GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

APLICACIÓN MÓVIL MULTIPLATAFORMA PARA UN SISTEMA DE MARKETING DE PROXIMIDAD BASADO EN BALIZAS BLE

Autor: Teresa González García

Director: David Contreras Bárcena

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Aplicación móvil multiplataforma para un sistema de marketing de proximidad basado en balizas BLE

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2020/21 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Teresa González García Fecha: 10/07/2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

CONTRERAS BARCENA DAVID - 53007209Y

Firmado digitalmente por CONTRERAS BARCENA DAVID - 53007209Y
Fecha: 2021.07.10 22:48:10 +02'00'

Fdo.: David Contreras Bárcena Fecha: 10/07/2021

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

APLICACIÓN MÓVIL MULTIPLATAFORMA PARA UN SISTEMA DE MARKETING DE PROXIMIDAD BASADO EN BALIZAS BLE

Autor: Teresa González García

Director: David Contreras Bárcena

Madrid

Agradecimientos

A mi familia, por su confianza y apoyo incondicional desde el inicio de la carrera.

A mis amigos, compañeros, profesores, y todas aquellas personas que hayan pasado por mi vida durante estos cuatro años, ya que gracias a todos ellos he conseguido llegar hasta aquí.

A mi director, por darme la oportunidad de trabajar en este proyecto tan interesante y por haberme ayudado durante el proceso.

A ICAI, por ayudarme a crecer tanto profesional como personalmente.

A Pau Fernández, por sus tutoriales tan útiles.

Muchas gracias.

APLICACIÓN MÓVIL MULTIPLATAFORMA PARA UN SISTEMA DE MARKETING DE PROXIMIDAD BASADO EN BALIZAS BLE

Autor: González García, Teresa.
Director: Contreras Bárcena, David.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

En este proyecto se diseña e implementa un sistema de marketing de proximidad basado en Balizas BLE, aplicado al sector minorista o retail. Para ello, se desarrollan los tres componentes principales del mismo: una app móvil multiplataforma con el framework Flutter para el interfaz de usuario, la lógica del back-end para gestionar todas las operaciones y peticiones del front-end mediante un servicio en la nube llamado Firebase, y la lógica para la comunicación entre dispositivos y Balizas BLE (Bluetooth Low Energy), tecnología clave en el Internet de las Cosas (IoT) debido que se utiliza en conexiones de baja potencia.

Además, en este proyecto se han explorado las tecnologías Flutter y Firebase más allá del desarrollo full-stack de la aplicación.

Palabras clave: App móvil multiplataforma, Balizas BLE, nube, IoT

1. Introducción

Hoy en día, nos encontramos en la era post-digital [1], en la que la tecnología avanza a paso agigantado. Estos avances aumentan la competitividad existente entre las empresas y las obliga a llevar a cabo transformaciones tecnológicas cada vez más disruptivas para mantener su posicionamiento en el mercado.

Sin embargo, empresas pequeñas y establecimientos minoristas o retailers en muchos casos no son capaces de seguir este ritmo de transformación frenético, por falta de recursos normalmente. Por ello, se quedan obsoletos, pierden clientela y beneficios económicos.

2. Definición del Proyecto

En este proyecto se desarrolla un sistema de marketing de proximidad para el sector retail, en el que los clientes o consumidores, mediante una aplicación móvil recibirán contenido exclusivo según su ubicación dentro de un establecimiento.[2]

La implementación de este sistema servirá como un impulso tecnológico para que establecimientos y comercios pequeños puedan hacer frente a los desafíos más urgentes del sector, agravados por la pandemia, como la omnicanalidad, digitalización de tiendas físicas o una visión 360º del cliente. [3]. Además, añadirá valor a la experiencia de compra de los clientes, ya que aporta unicidad, un aspecto muy valorado, lo que se resumirá en fidelización, mayor alcance, y beneficios económicos.[4]

Si bien las estrategias de marketing de proximidad llevan varios años utilizándose por empresas con distintas tecnologías, en este proyecto se utilizan las tecnológicas más demandadas del momento, cuya combinación resulta en un sistema económico, sencillo de utilizar y sostenible [5]: desarrollo móvil multiplataforma, servicios de Cloud Computing y dispositivos IoT (Balizas BLE, que se comunican mediante un protocolo inalámbrico de baja energía llamado Bluetooth Low Energy).

3. Descripción del sistema

El sistema desarrollado está formado por 3 componentes (además del usuario): aplicación móvil multiplataforma (Flutter) servicios back-end cloud (Firebase) y balizas (SensorTags).

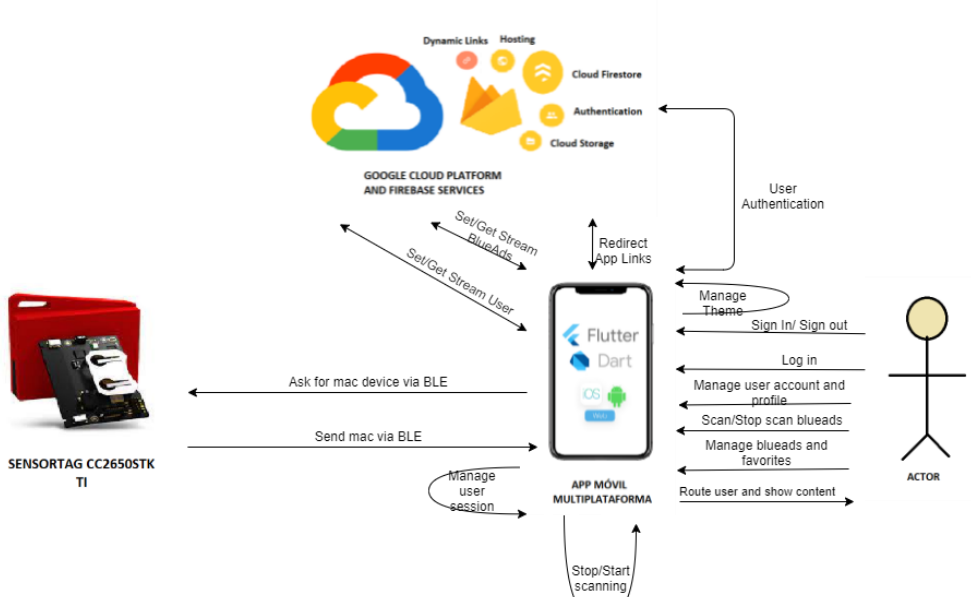


Figura 1 - Sistema desarrollado

- App móvil multiplaforma: es el core del sistema. Gestiona las interacciones con el usuario, back-end y con las balizas. Se encarga del interfaz de usuario y de la lógica para obtener o guardar datos de la nube, así como para detectar balizas y desencadenar una acción, en este caso mostrar contenido al usuario.

- Servicios cloud Firebase: se configuran los siguientes servicios de back-end para este proyecto: Autenticación, Hosting, Cloud Firestore (base de datos), Dynamic Links y Cloud Storage (bucket de almacenamiento).

- Balizas elecrónicas: tienen funcionalidad beacon (protocolo iBeacon) y soportan varios protocolos inalámbricos. Además, no cambian su Bluetooth MAC por defecto.

El proceso de scanning BLE de balizas realizado por la app , consiste en buscar de forma iterativa coincidencias entre la Bluetooth MAC de cada dispositivo encontrado con algún identificador de la base de datos cloud.

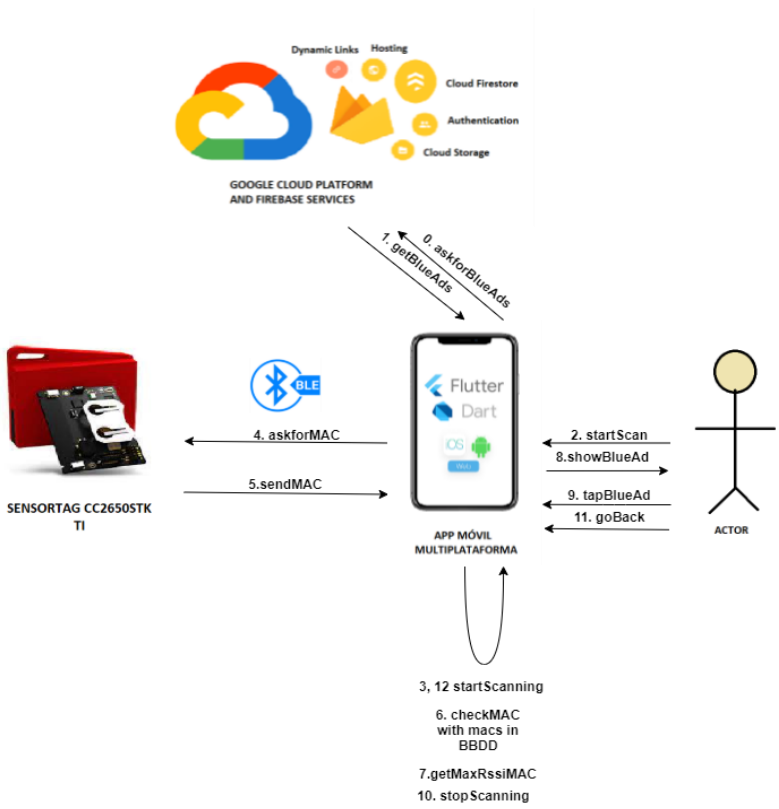
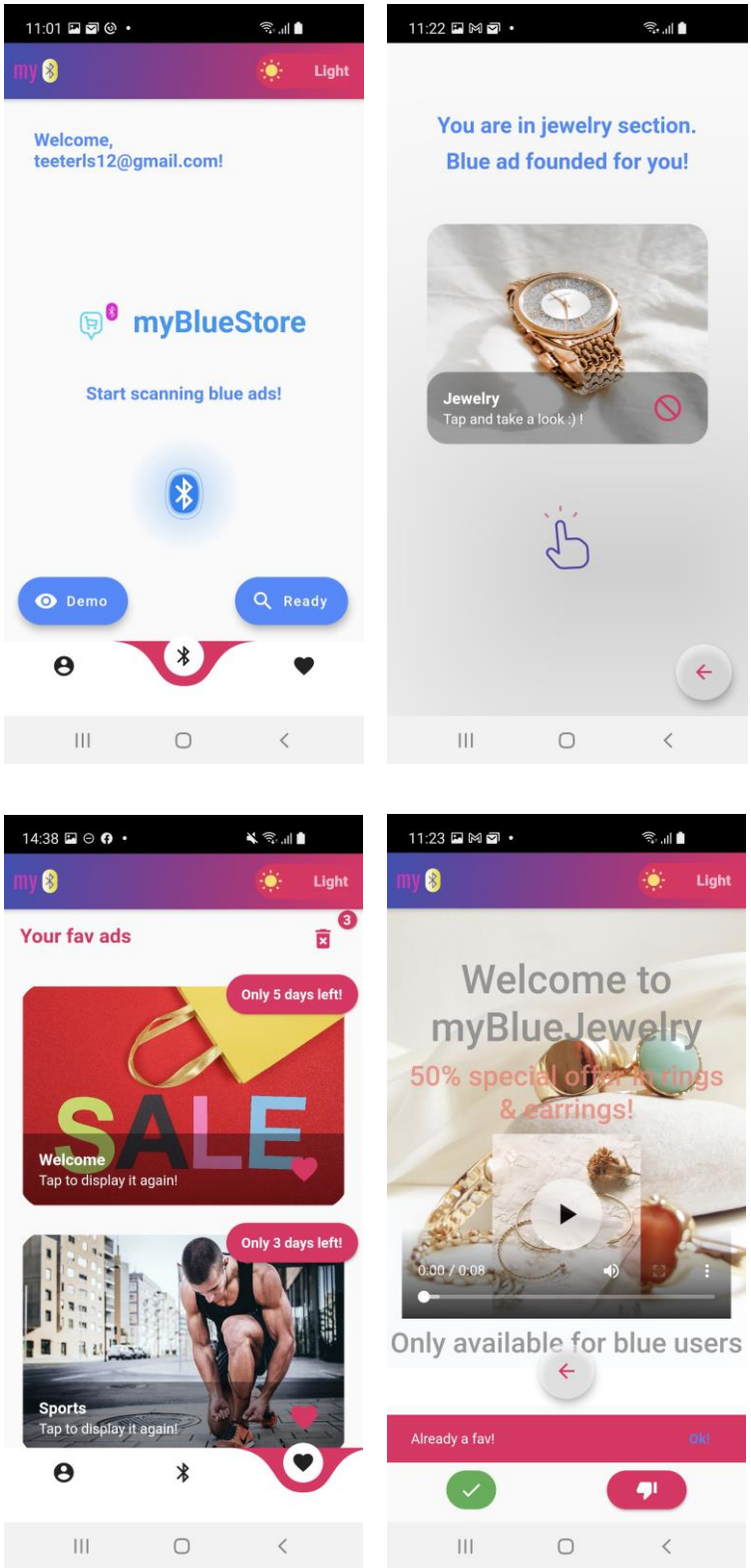


Figura 2 - Proceso scanning BLE

4. Resultados

A continuación, se muestra la UI de la aplicación, con los casos de uso más importante de un usuario autenticado con su email.

Figura 3 - Pantallas UI



5. Conclusiones

El presente trabajo recoge los primeros pasos de un proyecto más ambicioso: desarrollar un sistema de marketing de proximidad completo y establecer una red de establecimientos retail.

Con el alcance actual del trabajo, se han cumplido todos los objetivos iniciales, si bien por cuestiones de tiempo no se ha podido configurar la aplicación iOS integrar más servicios en el back-end de cloud, como herramientas de Machine Learning y tratamiento de datos. Sin embargo, la estructura del proyecto es perfectamente escalable para poder continuar de manera inmediata añadiendo más funcionalidades.

6. Referencias

[1] Daugherty, P. La era post-digital ya está aquí. ¿Estás preparado para lo que viene? *Technology Vision 2019, Accenture.* Recuperado de https://www.accenture.com/es-es/insights/technology/technology-trends-2019 el 21/1/21.

[2] Ligero, R. 8 cosas que deberías saber sobre los beacons, *Accent Systems*. Recuperado de https://accent-systems.com/es/blog/8-cosas-que-deberias-saber-sobre-los-beacons/ el 17/1/21.

[3] Refojos, M. El retail ya piensa en el 2021: ¿cuáles son los desafíos más urgentes? *El Periódico.com.* Recuperado de https://www.elperiodico.com/es/activos/empresas/20201118/retail-comercio-desafios-urgentes-8209155 el 10/7/21.

[4] Rodríguez, C. Situación y retos 2020 Mercado de gran consumo España., *Nielsen.* Recuperado de https://www.ioncomunicacion.es/wp-content/uploads/Tendencias-consumidor-2020.pdf el 15/5/21.

[5] Impacto de los dispositivos IoT en la sostenibilidad, *Envira IoT*. Recuperado de https://enviraiot.es/dispositivos-iot-sostenibilidad-impacto/ el 2/7/21.

CROSS-PLATFORM MOBILE APPLICATION USED IN A PROXIMITY MARKETING STRATEGY WITH BEACONS

Author: González García, Teresa.

Supervisor: Contreras Bárcena, David.

Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

This project designs and implements a proximity marketing system based on electronic beacons, applied to the retail sector. For this purpose, the three main components are developed: a multiplatform mobile app with the Flutter framework for the user interface, the back-end logic to manage all front-end operations and requests through a cloud service called Firebase, and the logic for communication between devices and electronic beacons through Bluetooth Low Energy (BLE), a key technology in the Internet of Things (IoT) used in low-power connections.

Keywords: Cross-platform mobile app, beacons, proximity, cloud, IoT

1. Introduction

Today, we find ourselves in the post-digital era [1], in which technology is advancing at a rapid pace. These advances increase the existing competitiveness among companies and force them to carry out increasingly disruptive technological transformations in order to maintain their position in the market.

However, small companies and retail establishments are often unable to keep up with this frenetic pace of transformation, usually due to a lack of resources. As a result, they become obsolete, lose customers and economic benefits.

2. Project definition

This project develops a proximity marketing system for the retail sector, in which customers or consumers, through a mobile application will receive exclusive content according to their location within an establishment.[2] The implementation of this system will serve as a technological boost for small establishments and businesses to

meet the most urgent challenges of the sector, aggravated by the pandemic, such as omnichannel, digitization of stores.

The implementation of this system will serve as a technological boost for establishments and small businesses to face the most urgent challenges of the sector, aggravated by the pandemic, such as omnichannel, digitization of physical stores or a 360º vision of the customer. [3]. In addition, it will add value to the customer shopping experience, as it brings uniqueness, a highly valued aspect, which will be summarized in loyalty, greater reach, and economic benefits.[4]

Although proximity marketing strategies have been used for several years by companies with different technologies, this project uses the most demanded technologies of the moment, whose combination results in an economical, simple to use and sustainable system [5]: multiplatform mobile development, Cloud Computing services and IoT devices (electronic beacons, which communicate through a low energy wireless protocol called Bluetooth Low Energy).

3. System description

The developed system consists of 3 components (in addition to the user): cross-platform mobile application (Flutter), back-end cloud services (Firebase) and beacons (SensorTags).

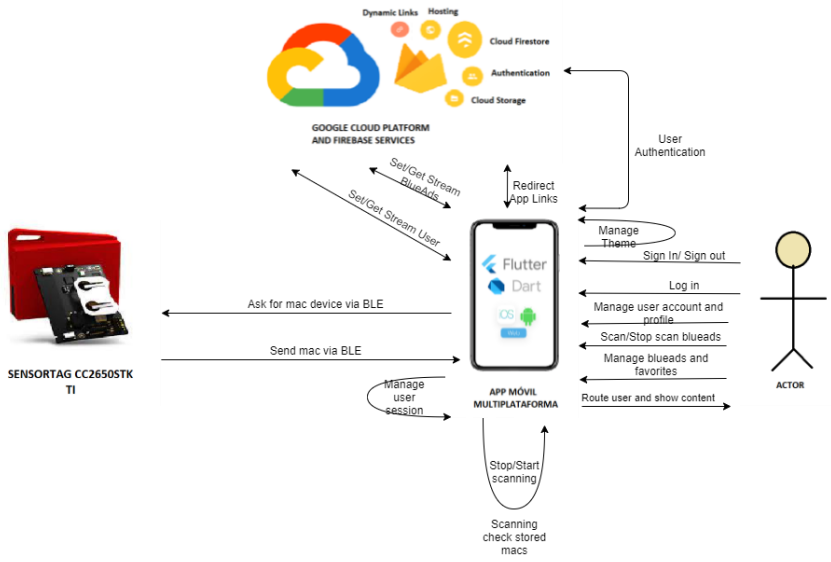


Figure 1 - System developed

- Multi-platform mobile app: it is the core of the system. It manages the interactions with the user, back-end and beacons. It is in charge of the user interface and the logic to get or save data from the cloud, as well as to detect beacons and trigger an action, in this case displaying content to the user.

- Firebase cloud services: the following back-end services are configured for this project: Authentication, Hosting, Cloud Firestore (database), Dynamic Links and Cloud Storage (storage bucket).

- Elecronics beacons: they have beacon functionality (iBeacon protocol) and support several wireless protocols. In addition, they do not change their Bluetooth MAC by default.

The BLE scanning process of beacons performed by the app consists of iteratively searching for matches between the Bluetooth MAC of each device found with an identifier in the cloud database.

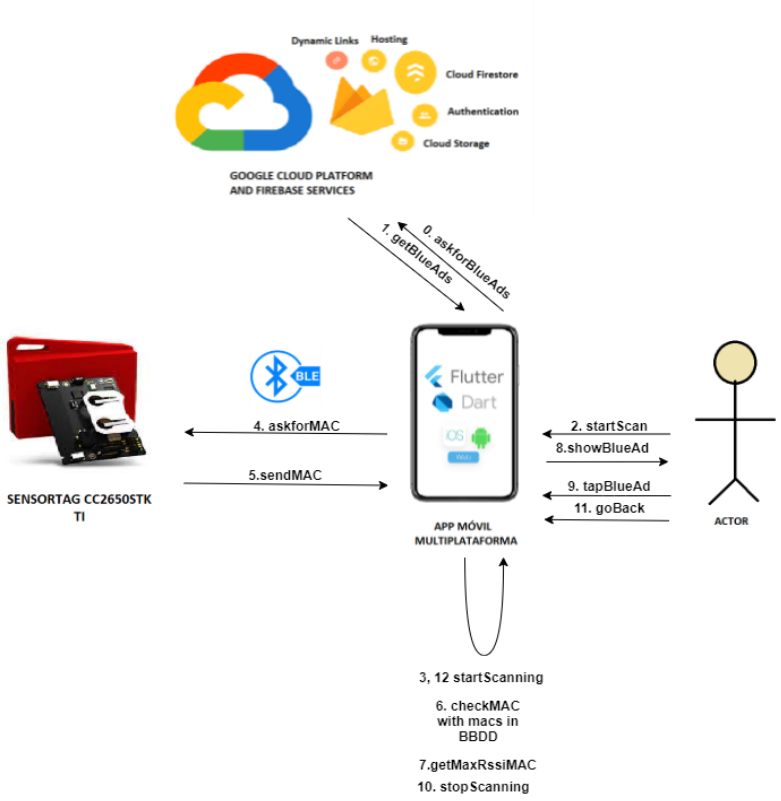
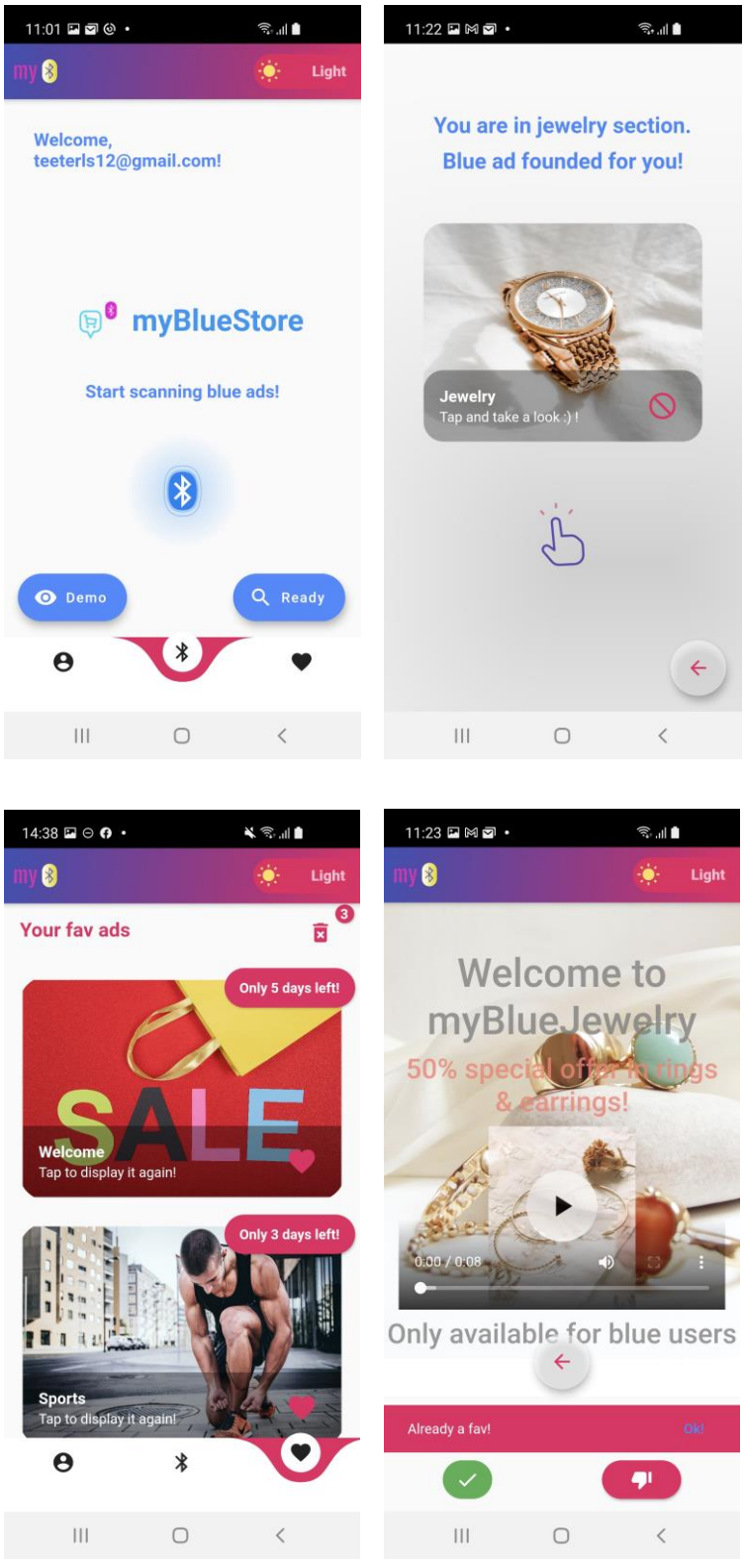


Figure 2 - Scanning process

4. Results

The UI of the application is shown below, with the most important use cases of a user authenticated with his email.

Figure 3 - UI Screens



5. Conclusions

This work is the first steps of a more ambitious project: to develop a complete proximity marketing system and establish a network of retail establishments.

With the current scope of work, all the initial objectives have been met, although due to time constraints it has not been possible to configure the iOS application to integrate more services in the cloud back-end, such as Machine Learning tools and data processing. However, the project structure is perfectly scalable to be able to continue immediately adding more functionalities.

6. References

[1] Daugherty, P. The post-digital era is here - are you ready for what's next? Technology Vision 2019, Accenture. Retrieved from https://www.accenture.com/es-es/insights/technology/technology-trends-2019 on 1/21/21.

[2] Lightweight, R. 8 things you should know about beacons, Accent Systems. Retrieved from https://accent-systems.com/es/blog/8-cosas-que-deberias-saber-sobre-los-beacons/ on 1/17/21.

[3] Refojos, M. Retail is already thinking about 2021: what are the main challenges? El Periódico.com. Retrieved from https://www.elperiodico.com/es/activos/empresas/20201118/retail-comercio-desafios-urgentes-8209155 on 10/7/21.

[4] Rodríguez, C. Situation and challenges 2020 Spanish FMCG market, Nielsen. Retrieved from https://www.ioncomunicacion.es/wp-content/uploads/Tendencias-consumidor-2020.pdf on 5/15/21.

[5] Impact of IoT devices on sustainability, Envira IoT. Retrieved from https://enviraiot.es/dispositivos-iot-sostenibilidad-impacto/ on 2/7/21.

Índice de la memoria

Índice de figuras

Índice de tablas

Índice de códigos

Capítulo 1. INTRODUCCIÓN

La sociedad ha experimentado una transformación tecnológica sin precedentes desde finales del pasado siglo hasta la actualidad. Hoy en día, nos encontramos en la era post-digital, avanzando hacia una sociedad inteligente, en la que, como se explica en [1], hemos integrado la tecnología en nuestras vidas de tal forma que la innovación digital en el sector empresarial se ha convertido en un requisito indispensable para poder competir en el mercado.

Por tanto, las empresas se encuentran en medio de una carrera tecnológica, en la que se busca implementar las tecnologías más disruptivas para aumentar su target (público objetivo y potencial) y mantener el posicionamiento en el mercado. Una tendencia tecnológica actual, impulsada por las necesidades de los clientes, es la personalización de productos y servicios, ya que según [2], los consumidores comienzan a comprar de forma más intencionada que compulsiva, prestando gran atención en la experiencia de compra y la unicidad.

Este ritmo frenético de cambios no solo lo experimentan las empresas grandes y medianas, sino también las pequeñas, y en particular el sector retail. Entre la competitividad existente y el impacto negativo de la pandemia, este sector se enfrenta a retos bastante urgentes, entre los cuales se pueden destacar, según [3]: obtener una visión 360º del cliente (conocerlo más allá de simples datos de formulario), gestionar la omnicanalidad, mejorar el consumer journey o digitalizar la tiena física. Todos estos retos requieren un cambio tecnológico, ajustado a los recursos disponibles de las tiendas y comercios.

1.1 DESARROLLO DE APLICACIONES MÓVILES

En los últimos años, desde la Revolución Industrial 4.0, han aparecido nuevas tecnologías que cambiarán en el futuro próximo la forma de vivir. El smartphone, dispositivo que se encuentra en constante evolución, ha alcanzado su madurez y se ha convertido en la tecnología más utilizada mundialmente. Según los estudios [4] y [5], en los últimos 3 años la penetración de mercado ha sido mayor en las aplicaciones móviles (68%) que en las web (53%), y además se ha podido concluir que en una gran lista de países se invierte más del 85% del tiempo total de uso de un smartphone en apps móviles y el tiempo restante en el navegador web.

Ilustración 1 - Penetración apps moviles (Informe Mobile Ditrendia 2020)

Del estudio de investigación sobre el uso de los smartphones durante el pasado 2020 realizado por [6] se pueden sacar varios datos relevantes: el consumo de datos se ha disparado (7.2 GB en datos móviles por usuario al mes) y el número de usuarios únicos en dispositivos móviles equivale al 67% de la población global total. Además, el mercado de

Ilustración 2 - Situación Global Mobile 2020 (Yiminshum)

las aplicaciones móviles está creciendo vertiginosamente, siendo las apps de compras las segundas más descargadas en la actualidad.

Esta explosión del mercado de las apps móviles ha acelerado el surgimiento de tecnologías y herramientas de desarrollo móvil para hacer la tarea lo más fácil y rápida posible, y hacer frente a las necesidades de los usuarios a un ritmo acelerado añadiendo nuevas apps al ecosistema.

Actualmente, los sistemas operativos móviles predominantes en el mercado son Android (74%) e iOS (25%). Estos utilizan Kotlin/Java y Swift/Objective-C como lenguajes nativos. Con estos y las APIs nativas, se pueden lograr desarrollos muy complejos en cuanto a diseño y funcionalidad para comunicarse directamente on el hardware, programando a nivel de sistema operativo.

Sin embargo, la programación móvil con lenguaje nativo es solo una opción de las múltiples existentes, entre ellas también se encuentran las apps web, híbridas y multipataforma.

Una app web es básicamente una página web que ha sido optimizada para poder utilizarse en un móvil. Utiliza los lenguajes web tradicionales HTML, CSS y JavaScript para el front-end. El mismo código es compatible para cualquier sistema operativo ya que se accede a ellas a través de un navegador web, por lo que no hace falta descargarlas. Hoy en día hay la posibilidad de adaptar las aplicaciones web a las llamadas app web progresivas, que no dejan de ser apps multiplataforma ejecutadas en el navegador web sin necesidad de descarga, pero con una experiencia muy similar a una nativa. No obstante, como ya se ha explicado, la penetración de las apps en navegadores web está disminuyendo progresivamente.

Por otro lado, una app híbrida puede considerarse como una combinación entre nativa y web y es a su vez muy similar a una app multiplataforma. Los frameworks tanto híbridos como multiplataforma trabajan sobre un único código nativo que se puede compilar o exportar en cualquier plataforma con unos cambios mínimos (ciertas configuraciones especiales que hagan falta, como para la comunicación Bluetooth). Además, tienen acceso a las APIs nativas.

Es importante matizar que los términos de app multiplataforma e híbrida se suelen utilizar indistintamente y pueden confundirse, pero no son sinónimos. Como se explica en [7], los frameworks híbridos como Ionic o Apache Cordova *"pintan webs embebidas en una carcasa nativa",* esto significa que incrustan elementos web (HTML, CSS, JavaScript) en un entorno nativo, obteniendo funcionalidades nativas. Sin embargo, los multiplataforma como React Native o Flutter al compilar el código producen apps totalmente nativas sin intervención de un navegador para su ejecución.

Ilustración 3 - Resumen desarrollo apps móvil (Freecodecamp.org)

A la hora de escoger entre desarrollar una app nativa o híbrida/multiplataforma, como se explica en [8] hay que tener en cuenta las principales similitudes y diferencias. Las similitudes son que para el usuario, el *look & feel* es prácticamente idéntico, además de que ambos desarrollos tienen acceso a las APIs nativas, por lo que se pueden implementar funcionalidades complejas.

Sin embargo, desarrollar una app multiplataforma supone un ahorro en tiempo y en costes. Con un mismo código se puede lanzar una app tanto en iOS como en Android, por lo que el *time-to-market* es mucho menor. El coste de aprendizaje de un único lenguaje es considerablemente inferior al que supone aprender los lenguajes nativos por separado, y además se puede reutilizar código más fácilmente. La única desventaja de los frameworks

híbridos con respecto a los nativos es que, ya que los primeros corren sobre un "entorno virtual", el rendimiento es algo inferior.

Ahora mismo, existe una tendencia entre desarrolladores de migrar hacia herramientas híbridas o multiplataforma por las ventajas que se han expuesto, pero sobre todo por la rapidez de desarrollo y la rentabilidad económica que supone para las empresas. Como se explica en [9] las aplicaciones multiplataformas son una herramienta de éxito para mantener la posición en un mercado tan acelerado, y las empresas están invirtiendo masivamente en ellas, ya que tienen mayor alcance y permiten obtener mejor retorno de la inversión.

1.2 IOT Y CLOUD COMPUTING

El Internet de las Cosas (IoT) y Cloud Computing han sido de las tecnologías más disruptivas de la 4ª Revolución Industrial. La combinación de ambas da lugar a soluciones innovadoras y escalables.

El número de dispositivos conectados a la red (normalmente Internet) se está volviendo inmanejable, y seguirá *in crescendo*, por lo que ha sido necesario desplegar nuevas tecnologías como 5G para evitar una congestión de la red (entre otros objetivos como disminuir la latencia e incrementar la velocidad de trasmisión). La interconexión o agrupación de varios dispositivos forma un ecosistema IoT, en el cual se comporten un volumen de datos de todo tipo gigantesco.

Las aplicaciones de un ecosistema IoT son infinitas, y nuevos casos de uso han fomentado la aparición de varias tecnologías de red. Como se indica en [10], en las *Smart cities* y *Smart homes* emergentes se utilizan simultáneamente tecnologías de acceso inalámbrico de distinto alcance y velocidad de transmisión, como Wi-Fi, Bluetooth, ZigBee o Z-Wave.

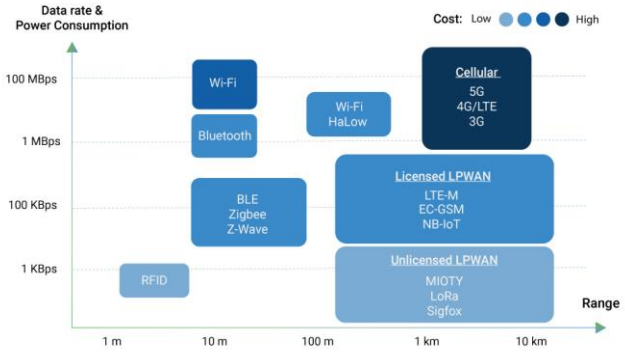


Ilustración 4 - Comparativa tecnologías inalámbricas IoT (SaftBatteries.com)

En conexiones que requieren de un bajo consumo de energía de los dispositivos participantes, entra en juego la tecnología Bluetooth Low Energy, que se explicará en el siguiente capítulo.

Por otro lado, la maduración de la tecnología IoT, y en consecuencia la gran cantidad de datos generados requiere en la mayoría de los casos de uso integrar una plataforma en la nube para la gestión y procesamiento de estos, así como para proveer seguridad, entre otras funcionalidades.

El Cloud Computing, o simplemente la "nube" ha revolucionado la forma en la que las empresas (y también particulares) se relacionan con las TIC (tecnologías de información y de la comunicación) y adquieren recursos tecnológicos. Con esta tecnología, las empresas ofrecen servicios y aplicaciones en la red (normalmente Internet) a otras empresas y particulares, mediante un modelo de pago por uso.

Cloud Computing supone un nuevo paradigma de la computación, ya que ha separado el hardware del software. Con cloud, las organizaciones o usuarios pueden acceder de forma remota a espacios de almacenamiento, bases de datos, servidores, etcétera, sin necesidad de infraestructura propia, dejando atrás los costes que supone el mantenimiento de servidores físicos.

Además de reducción de costes, cloud aporta múltiples ventajas a las empresas: flexibilidad, agilidad y velocidad, escalabilidad, alta disponibilidad, seguridad, o mayor capacidad de almacenamiento [11]

Actualmente, las empresas están migrando sus servicios y aplicaciones a alguna infraestructura en la nube. Para que una empresa siga siendo competitiva, migrar a la nube ya no es una opción. Según [12], Gartner predice que más del 80% de las empresas de todo el mundo habrá migrado todos los centros de datos locales a alguna infraestructura cloud.

Cloud Computing ofrece distintos modelos de servicios, que se resumen en: Infraestructure as a Service (IaaS), Platform as a Service (PaaS) y Software as a Service (SaaS). [11]

- IaaS: esta solución es la que se necesita para que los administradores y arquitectos de red puedan migrar de sus centros locales a la nube. Este modelo ofrece control absoluto, flexibilidad y escalabilidad. El proveedor de IaaS da acceso, mediante pago por uso, a máquinas virtuales, características de redes y espacios de almacenamiento. Además, es el proveedor el que se encarga de la seguridad de la infraestructura, pero no de las aplicaciones instaladas. Esa seguridad corre a cargo de las empresas. Un ejemplo de poveedor IaaS es Amazon Web Services (AWS).

- PaaS: es una solución pensada para desarrolladores de aplicaciones, proporcionándoles un entorno de desarrollo que facilita el despliegue de aplicaciones. Mediante pago por uso, estos tienen acceso a herramientas de gestión, distribución y testeo de apps. PaaS ofrece más flexibilidad y control.
- SaaS: es una solución destinada a los usuarios finales, con la que tienen acceso al software y aplicaciones de los proveedores mediante un modelo de pago por uso. La conexión se realiza a través de una API o de la red. El proveedor es el responsable del desarrollo y mantenimiento del software, mientras que el usuario tiene poco control, pero le aporta mucho valor.

Recientemente ha entrado en escena un modelo intermedio entre PaaS y SaaS, técnicamente construido sobre PaaS, que es el Backend as Service (BaaS, o mBaaS si es mobile). Con este

modelo, se ofrece una plataforma con los servicios de gestión del back necesarios para un rápido de aplicaciones web y móviles. Mientras PaaS simplifica el despliegue de una aplicación, BaaS simplifica el proceso de desarrollo. Ejemplos de este modelo son Firebase, AWS, Parse.

Ilustración 5 - Jerarquía Cloud Computing con nivel BaaS

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este capítulo se explicarán con detalle las tecnologías utilizadas para el desarrollo del proyecto, divididas en secciones.

2.1 TECNOLOGÍA BLUETOOTH

Bluetooth es una tecnología inalámbrica de acceso personal y corto alcance (WPAN), que permite la comunicación y transmisión de datos y voz punto a punto mediante un enlace de radiofrecuencia, normalmente entre 2 dispositivos. Como se explica en [13] esta transmisión se puede llevar a cabo sin conexión, es decir, se inicia aunque el destinatario no haya aceptado la transmisión todavía, y con conexión, en la que antes de empezar a transmitir datos reales se establece una conexión virtual entre los dispositivos.

La tecnología, considerada como estándar industrial, fue desarrollada e introducida a principios de los 90 por Bluetooth Specialist Interest Group (Bluetooth SIG), y opera en la banda ISM de 2.4 GHz, la cual comparte con la tecnología Wi-Fi.

La tecnología Bluetooth, a diferencia de otras tecnologías de red, está especializada en las conexiones sencillas y de bajo consumo y transferencia de datos en distancias cortas. Por esto, alcanza velocidades más bajas en las transferencias, requiriendo más tiempo para el envío de paquetes grandes. Se suele utilizar para enviar archivos individuales y servicios sencillos.[13]

Actualmente, prácticamente cualquier dispositivo es compatible con alguna versión de Bluetooth. La versión más nueva es Bluetooth 5.2, también conocida como Bluetooth LE Audio. Esta utiliza el estándar Low Energy, para conexiones de bajo consumo de energía, además de un nuevo codec (LC3) que mejora la calidad del audio considerablemente. El alcance también aumenta hasta los 200 metros. [14]

A nivel de enlace, como en cualquier red de acceso (Wi-Fi, Ethernet), se utiliza una dirección Bluetooth MAC de 48 bits, asignada por el fabricante del dispositivo, para identificar únicamente a un dispositivo Bluetooth. La mitad de estos bits identifican al fabricante y la otra mitad al modelo. Un dispositivo físico, como un smartphone, posee de forma simultánea varias tarjetas de red, cada una con una dirección MAC.

Estos identificadores son únicos y se pueden utilizar para seguir dispositivos, y permitir o denegar el acceso de estos a una red. Sin embargo, hay formas de cambiarlas para dificultar el tracking de dispositivos. En el caso concreto de Bluetooth, según las especificaciones, esta MAC se denomina BD_ADDR. Además, se encuentran dos tipos de Bluetooth MAC: públicas y random o aleatorias (privadas y estáticas, resolubles y no resolubles). Un dispositivo Bluetooth debe poseer alguna de estos dos grandes tipos, o incluso ambos. [15]

A continuación, se expone un subprotocolo de la versión Bluetooth 4.0, en el que la gestión de las direcciones MACs es determinante.

2.1.1 BLUETOOTH LOW ENERGY

Una de las tecnologías IoT más importantes, como ya se ha adelantado, es el protocolo Bluetooth 4.0, más conocido como Bluetooth Low Energy (subconjunto de dicho protocolo) o Bluetooth Smart. La diferencia con respecto al Bluetooth convencional es que está diseñado para que, aun operando en la misma banda (2.4 GHz) y manteniendo un alcance de similar, requiera un bajo consumo de energía. Esto se consigue gracias a que el BLE permanece en modo suspensión constantemente, hasta que se inicie una conexión, con una tasa de 1 Mb/s y un alcance de hasta 60 metros aproximadamente. El resultado es que la batería de los dispositivos dura más tiempo.

Como se explica en [16], este protocolo utiliza 40 canales de radiofrecuencia de 2MHz cada uno, de los cuales 3 se utilizan para anunciarse y los 37 restantes para intercambio de datos.

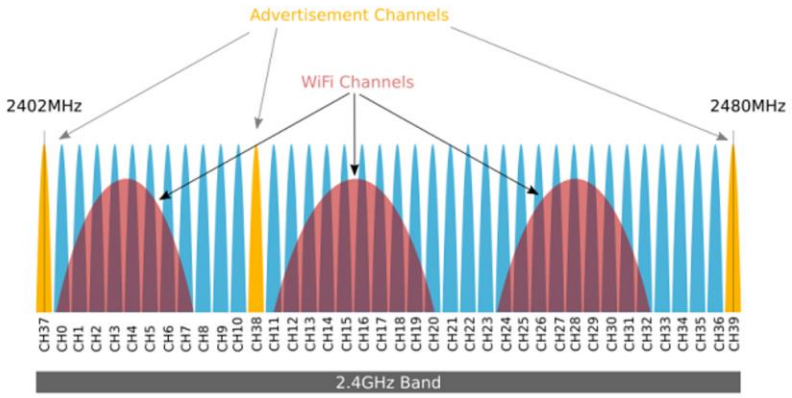


Ilustración 6 - BLE channels e interferencias (EDN Asia)

Como se puede ver en la Ilustración 6, los canales de advertising no se solapan con ningún canal Wi-Fi, para evitar interferencias en este proceso previo a la conexión. Estos canales (37, 28 y 39) son fijos, por lo que las peticiones de conexión no pueden realizarse en ningún otro canal disponible. Una vez el advertising ha tenido lugar, es cierto que según que canal se utilice, podrían darse interferencias durante la transmisión de datos debido al solape de algunos canales BLE con los de Wi-Fi.

En la pila de protocolo se encuentran dos subcategorías fundamentales: GAP (Generic Access Profile) Y GATT (Generic Attribute Profile). GAP se encarga de definir la topología general de la pila BLE, y los roles de dispositivos, mientras que GATT describe cómo se transfieren los atributos (datos) cuando se ha dado lugar el advertising y la conexión ha sido establecida. [16]

En [17] se explica el funcionamiento de redes BLE, cuyo concepto fundamental, recogido en el GAP es:

> *"En las redes Bluetooth de baja energía los dispositivos pueden ser centrales o periféricos. Los dispositivos centrales (teléfonos inteligentes, tabletas, computadoras, etc.) tienen mayor capacidad de procesamiento y son responsables de controlar los dispositivos periféricos. Los dispositivos centrales generalmente*

ejecutan software creado específicamente para interactuar con dispositivos periféricos. Estos últimos sirven como sensores que recopilan datos y los envían a dispositivos centrales para su procesamiento. La clave del bajo consumo es que no procesan datos, solo lo recogen."

Después de "pasar" por el GAP y establecer la conexión, el GATT es el encargado de definir los roles por conexión: cliente y servidor. Los servidores GATT son los que almacenan datos organizándolos en atributos (fragmento de datos etiquetados y accesibles). Como se explica en [16], algunos servicios o características Bluetooth estándar pueden representarse con identificadores de 16 bits, pero con el protocolo BLE se ofrece posibilidad a los fabricantes de definir servicios con un identificador único (UUID) de 128 bits para sus aplicaciones concretas.

Es importante mencionar que BLE no posee seguridad en el emparejamiento de dispositivos, como se explica en [17], aunque la comunicación si está cifrada y posee autenticación, entre otras características. Por lo tanto, en esa previa fase habrá que tomar alguna política de seguridad de las que se explica en el artículo para evitar escuchas de terceros en los intercambios de claves, que puedan derivar en un ataque *man in the middle*.

Por ejemplo, en relación con la seguridad, los fabricantes dispositivos BLE o Bluetooth Smart pueden hacer uso del Bluetooth LE Privacy. Como se explica en [18], los dispositivos periféricos están anunciando su presencia continuamente durante el proceso de advertising, cuyos paquetes contienen la dirección MAC Bluetooth para identificar al dispositivo, entre otros valores. Con LE Privacy, las direcciones MAC se reemplazan con un valor aleatorio que va cambiando en intervalos de tiempo determinados por el fabricante. Gracias a esto, cualquier dispositivo que esté escuchando de forma maliciosa no podrá distinguir si esa MAC random corresponde al mismo dispositivo físico a lo largo del tiempo. Parecerá que está siguiendo a muchos dispositivos BLE que solo están activos durante ciertos intervalos, por lo que no podrá trackear al dispositivo físico siguiendo solamente su MAC address.

Este protocolo tiene gran importancia en el proyecto, puesto que está presente en protocolos de tecnología beacon, la cual se explicará más adelante.

2.2 TECNOLOGÍAS SOFTWARE

Como ya se ha explicado anteriormente, los lenguajes de programación y frameworks (o entornos de trabajo) actualmente desarrollados suelen utilizarse para crear aplicaciones móviles para smartphones, tablets, wearables, etc. Por este motivo, estas herramientas deben cumplir con estos requisitos, como se explica en [19]: favorecer una buena experiencia de usuario, con la menor cantidad posible de sintaxis y al mismo tiempo, prestando atención a la memoria disponible en el dispositivo.

A continuación, se explican las tecnologías software escogidas para desarrollar una app móvil teniendo en cuenta contexto de rapidez y competitividad del mercado.

2.2.1 DART

Ilustración 7 - Logotipo Dart

Dart (originalmente llamado Dash), es un lenguaje de programación orientada a objetos open-source y gratuito creado por Google y presentado en 2011. Con este lenguaje, los desarrolladores de Google crean aplicaciones móviles (iOS, Android) y web de alta calidad, ya que tiene características dirigidas al desarrollo del lado del cliente. [20] Asimismo, este lenguaje tiene como objetivo facilitar a desarrolladores novatos la creación de aplicaciones móviles y web propias, lo que hasta no hace mucho era impensable.

Inicialmente, se creó este lenguaje como alternativa a JavaScript (lenguaje interpretado) para desarrollo web. Ya que los navegadores no pueden ejecutar este lenguaje, existe el compilador Dart2JS para llevar a cabo la traducción de forma directa y rápida, y así una app de Dart puede funcionar en cualquier navegador móvil y de escritorio. [19] Esta portabilidad también se encuentra en los sistemas operativos móviles, ya que Dart compila arm y código x86. [20]

Además, lenguaje comparte paradigmas de programación con otros lenguajes POO para modelar datos y comportamientos complejos, entre otros, resultando en una sintaxis muy similar. Dart dispone de variables, operadores, enunciados condicionales, bucles, funciones, clases, objetos, listas, herencias, y otros conceptos. Conociendo con anterioridad algún lenguaje POO, el aprendizaje de Dart es fácil, rápido y accesible. [19]

También tiene otras particularidades, que hacen de este lenguaje una herramienta productiva ,fuertemente tipada y con un código mucho más fácil de leer, acercándolo al lenguaje humano. La máquina virtual de Dart y la compilación Just-In-Time permiten que los cambios realizados en el código se ejecuten inmediatamente. [21] Además, se puede programar en Dart en cualquier editor de texto sencillo o IDE, y también en DartPad, editor online gratuito. Para fomentar esa rapidez, también este lenguaje dispone de un gestor de paquetes que se explicará más adelante.

Otra característica clave de Dart es su adaptación a la programación reactiva, que permite gestionar objetos de corta duración (como Widgets), mediante una rápida asignación de objetos y el recolector de basura generacional. Asimismo, Dart es que soporta programación asíncrona con APIs que utilizan objetos como Future y Stream. [20]

La única desventaja, es que es un lenguaje bastante nuevo, por lo que la documentación es escasa en comparación con otros lenguajes POO. Sin embargo, recientemente Dart está ganando en popularidad gracias a su framework Flutter, tendencia actual en el desarrollo de apps multiplataforma y que se explicará a continuación.

2.2.2 FLUTTER

Ilustración 8 - Logotipo Flutter

Flutter es el kit de software o framework open-source y gratuito presentado por Google en 2018 y programado en Dart. Es una herramienta utilizada en el desarrollo de interfaces de usuario para aplicaciones móviles multiplataforma (iOS, Android), con un código único. Además, con el actual SDK de Flutter 2 el soporte para apps web y de escritorio es estable.

En línea con los objetivos de Dart, este framework persigue crear apps lo más rápido y fácil posible, además de con un UI bonito y atrayente que pueda tener funcionalidades complejas propias de desarrollo nativo. Ya que el desarrollo de un único código es compatible con cualquier plataforma existente, y mediante el uso de herramientas como el gestor de paquetes de Dart, el time-to-market es muchísimo menor que con otras opciones de desarrollo de apps, como ya se explicó anteriormente. Básicamente, con Flutter un desarrollador novato puede crear una app más rápido y fácil que nunca y desplegarla “en todas partes”.

El SDK de Flutter, como es open-source, se encuentra en un repositorio abierto en Github, [23], con múltiples ejemplos, documentación, hilos, etcétera.

En este repositorio, nos podemos encontrar cuatro branches o ramas oficiales (entre muchas otras, ya que cualquiera puede solicitar una bifurcación de cualquier rama): [24]

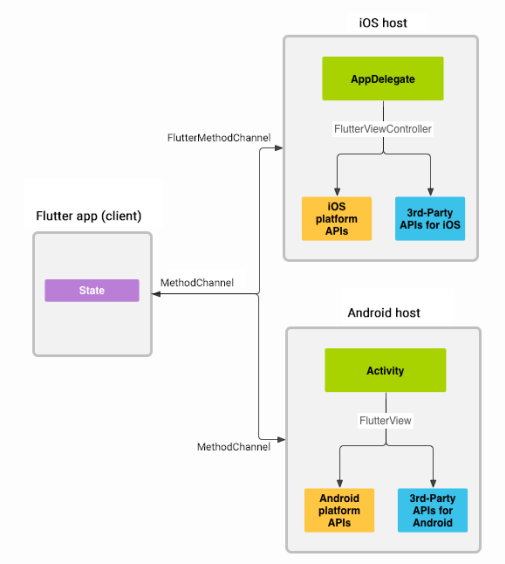


Ilustración 9 - MethodChannel Flutter
(Documentacion oficial)

- Master: es la rama principal, a partir de la cual se bifurcan las siguientes ramas, y corresponde la última versión absoluta del código. Es susceptible a fallos accidentales, por lo que no es el código estable.
- Beta: en esta rama cada vez se promociona la considerada mejor versión del mes anterior entre todas las ramas del repositorio. Las compilaciones se prueban en los codelabs(tutoriales).
- Dev: rama donde se aloja el código previo a la fusión con el código de la rama principal. Es la última versión, que necesita ser completamente probada antes de

pasar de una rama a otra. Se intenta que el paso sea lo más rápido posible, de forma casi continuada, pero muchas veces ocurren compilaciones incorrectas.

- Stable: en esta rama se encuentra el código definitivo considerado como "bueno". Al instalar el SDK de Flutter, por defecto se está en esta rama. Se vuelca código a esta rama de forma trimestral, para poder tener las máximas garantías de estabilidad.

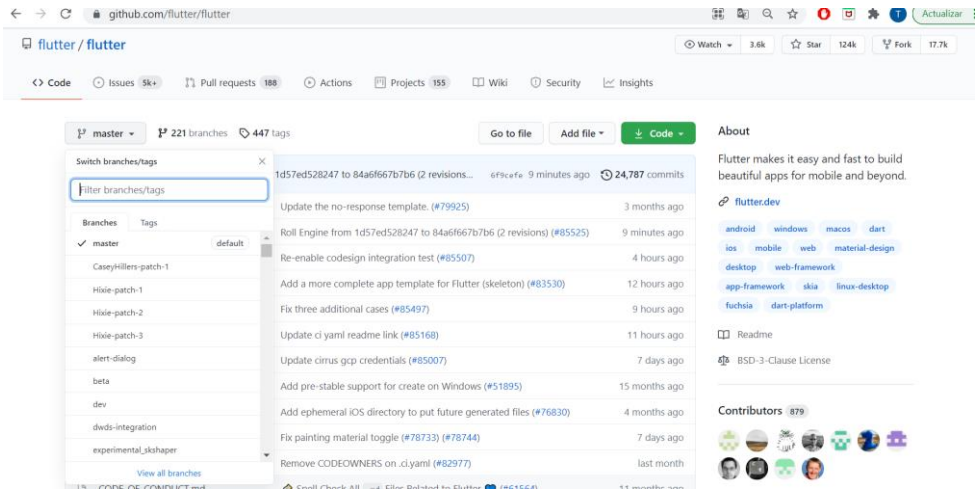


Ilustración 10 - Github Flutter SDK

En este proyecto, se utiliza el canal o rama dev debido a la incompatibilidad de algunos paquetes claves en el backend con la rama stable. Durante el desarrollo del proyecto, debido al uso de esta rama, se han sucedido algunos fallos relacionados con la inestabilidad del canal. Los pasos necesarios para instalar el SDK y cambiar de canal se explican en el Anexo II.

Actualmente, pese a que es una tecnología muy nueva, Flutter está ganando mucha popularidad por sus múltiples ventajas. Entre frameworks multiplataforma, su principal rival es React Native, gratuito y creado por Facebook. Aunque ambos sean multiplataforma, hay dos diferencias claves entre ellos, que se recogen en la siguiente imagen:

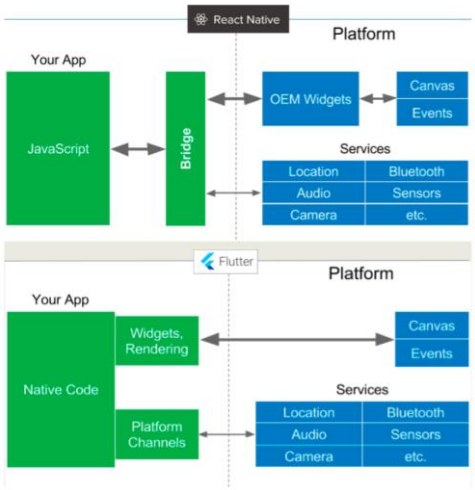


Ilustración 11 - Comparativa Flutter y React Native (Paradigma Digital)

Primero, Flutter, como está programado en Dart, compila muy rápido produciendo instrucciones nativas compatibles con arm64 y x86. Sin embargo, React Native compila en JavaScript de una forma menos efectiva, ya que como se puede ver en la imagen, es necesario un componente Bridge entre medias que penaliza el rendimiento. Como resultado, el time-to-market de Flutter es menor.

Por otro lado, React Native utiliza componentes UI nativos (OEM Widgets), por lo que la app se verá distinta según el sistema operativo o versión. Se pueden lograr funcionalidades complejas, pero el desarrollador debe estar pendiente de que estos componentes se visualicen o comporten correctamente en cada plataforma. Sin embargo, Flutter utiliza sus propios componentes, los widgets, por lo que la misma aplicación se verá igual en cualquier versión de cualquier sistema operativo, por más antigua que sea, algo que puede tranquilizar al desarrollador. [21]

Aunque con Flutter las apps tengan la misma UI y se comporten igual en cualquier plataforma, gracias a dos sets de widgets (Material Design y Cupertino), se pueden imitar los diseños de Android e iOS respectivamente. [25]

Como se explica en [7] Reactive Native en este momento tiene mayor posición en el mercado, pero se prevé un empate a medio plazo.

2.2.2.1 Widgets

Flutter aprovecha las ventajas de programación reactiva de Dart sobre asignación rápida de objetos, ya que este framework se fundamenta en elementos llamados Widgets. Un widget es coloquialmente, un trozo de pantalla. Este trozo de pantalla puede ser estático (o sin estado) o con estado.

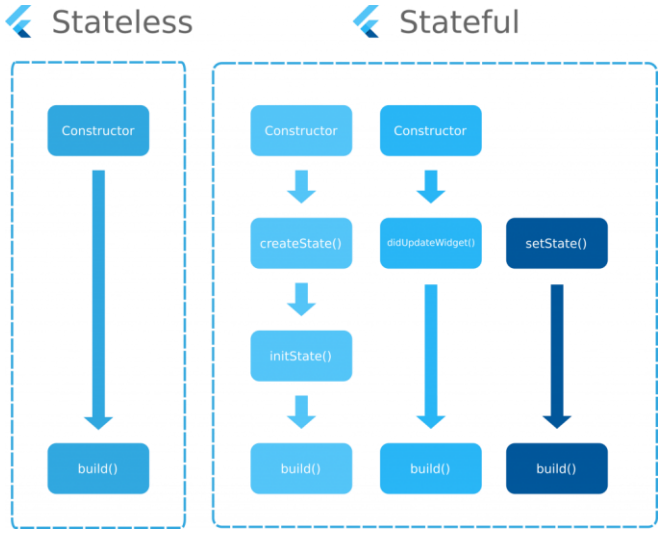


Ilustración 12 - Staless Widget vs Stateful Widget (Documentación oficial)

Un Widget estático es aquel que una vez que se crea no cambia nunca, independientemente de la acción del usuario. Ejemplos de este tipo son Text, Row, Column, Container.

Por otro lado, un Widget con estado es dinámico, puede cambiar (modificar su interfaz) por determinados eventos desencadenados por el usuario de la aplicación, o al recibir datos desde el backend. La primera vez que se construye (build), puede tener un estado inicial, (initState), pero una vez construido se puede manipular el estado (setState) cuando sea necesario. Se hace un rastreo o seguimiento del estado del widget. Ejemplos de este tipo son TextField, RadioButton o Checkbox.

Para construir el interfaz de cada pantalla que compone una aplicación, son necesarios varios widgets, que van montando unos sobre otros siguiendo un orden jerárquico, formando layouts. La traducción “visual” del código Flutter y su jerarquía configura el Widget Tree, en el que a partir del primer widget (padre) se van construyendo/ montando otros widgets

(hijos), que a su vez son padres de otros widgets, y así sucesivamente. De hecho, un desarrollador puede crear nuevos widgets con o sin estado compuestos por múltiples Widgets hijos.

Un ejemplo muy sencillo es el siguiente, sacado de la página oficial de Flutter:

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

En este código, se puede observar la jerarquía claramente. La app que se ejecutará en el main (MyApp), es un widget sin estado que contiene una jerarquía de Widgets de Flutter. En este caso, es importante destacar la diferencia entre propiedades/argumentos e hijos. Un widget tiene muchas propiedades o argumentos, entre las cuales algunos son obligatorios para ser construidos. Si se consulta la API de Flutter, en los constructores de cada Widget se pueden ver las propiedades que son obligatorias (parámetro required) y las que son opcionales. Las propiedades están al "mismo nivel" en el árbol de jerarquías. Ser hijo es una de las muchas propiedades del padre ("child" o "children"), que normalmente es opcional, pero que supone el siguiente nivel inferior en el árbol.

En este caso, el widget padre es MaterialApp, que es el widget raíz para crear aplicaciones acordes a Material Design. Dentro de este, nos encontramos la propiedad "title", que es un

String con el título de la app, y la propiedad "home" que es el widget principal sobre el que se muestra en la UI.

El widget Scaffold es muy utilizado como "plantilla" para las pantallas, por cómo distribuye los widgets en su interior. Dentro de este, nos encontramos la propiedad "appBar", de tipo widget, que construirá una barra de navegación en la parte superior de la pantalla, en este caso con un "title" Text. Además, en el "body" de Scaffold o parte central, se muestra un Center (widget de posición). Por último, este widget sí que posee un hijo, que depende de él y en este caso es un Text.

Sin embargo, los Widgets, trozos de pantalla, son inmutables, con instancia única. Si se quieren añadir cambios en el interfaz, habría que añadir un nuevo widget, lo que afecta negativamente a la rapidez de representación de la UI.

Por tanto, para conseguir el alto rendimiento de las apps construidas, Flutter optimiza la renderización o representación de los widgets mediante una estrategia declarativa, en la que cada widget se encarga de su propia representación según su estado interno. Entran en juego otras dos clases: Element y Render Tree, que ayudan al propio framework a calcular las diferencias en el estado de algún widget y realizar solo los mínimos cambios necesarios, es decir, "repintar" solo los trozos de pantalla que han cambiado. [7]

El Element Tree contiene objetos llamados Elements, que son mutables y pueden cambiar sus propiedades sin tener que crear un nuevo objeto. Estos contienen referencia o punteros a los Widgets o componentes de la UI, por lo que representan posiciones del árbol de Widgets. Además, se encargan de actualizar la UI ya que guardan los estados de los Widgets.

El Render Tree contiene objetos llamados renders, que poseen toda la información necesaria para calcular layouts (localización en la pantalla, tamaño y composición a nivel de pixel) y dibujar la UI. Efectivamente, cuando se representa la UI en Flutter, pese a que en el código se escriben Widgets, realmente se pintan RenderObjects.

Como se explica en [26] , cuando se inicia la aplicación, por cada widget en el Widget Tree se crea un objeto element en el Element Tree y un objeto render en el Render Tree.

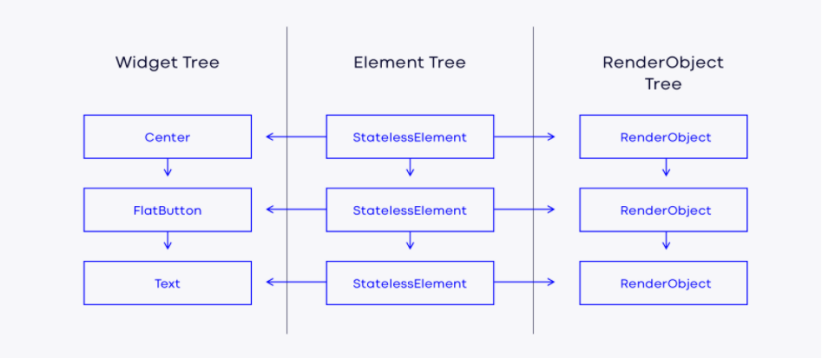


Ilustración 13 - Flutter Trees (Iteo.com)

Los Elements funcionan como nexos entre los Widgets y los RenderObjects, ya que tienen referencias de ambos. Cuando un Element detecta algún cambio en el estado de un Widget, compara el tipo de objeto en tiempo de ejecución de este con el RenderObject correspondiente. Solo si estos no coinciden, se crea un nuevo RenderObject, se calculará un nuevo layout y se repintará esa parte o trozo de pantalla únicamente.

Flutter posee un catálogo amplísimo de Widgets disponibles en el paquete "material" (incluido por defecto en el proyecto y necesario para construir la UI). Además, se pueden incorporar Widgets nuevos importándolos como dependencias desde una herramienta de paquetes de Dart llamado Pub, que se explicará posteriormente más en detalle.

En el capítulo 5 se explican con detalle los Widgets y métodos más relevantes en el proyecto.

2.2.2.2 Backend

Ya que Flutter es un framework enfocado al UI, es recomendable gestionar la lógica de peticiones y consultas desde el interfaz mediante alguna herramienta de back-end (si bien hay paquetes en Dart para alojar información en bases de datos locales, entre otras herramientas), sobre todo con vistas a escalar el proyecto.

Flutter es compatible con cualquier tecnología y lenguaje back-end (Node.js, PHP, Python, etc). No obstante, existe una tendencia reciente, como ya se ha explicado, a migrar las apps a algún servicio cloud, por motivos de escalabilidad, seguridad, flexibilidad entre otros. Por

esto, plataformas BaaS (o mBaaS) se están convirtiendo en primera opción para desarrolladores. [27]

Se ha decidido utilizar Firebase, ya que es también una tecnología propiedad de Google, por lo que todos sus servicios son compatibles e integrables con Flutter. Además, se está convirtiendo en una revolución dentro del stack tecnológico mobile.

2.2.3 FIREBASE

Ilustración 14 - Logotipo Firebase

Firebase es una plataforma BaaS (backend como servicio) de Google para el desarrollo de aplicaciones móviles multiplataforma, web y gaming. Esta ofrece múltiples servicios de gestión de aplicaciones en la nube para acelerar la tarea de creación de aplicaciones de alta calidad, supervisar su funcionamiento mediante informe y analíticas, así como para facilitar el proceso de escalada de los proyectos. Posee un kit de herramientas enfocadas a promocionar las aplicaciones (como AdMob), y ganar dinero. Por todas las comodidades que ofrece al desarrollador, hoy Firebase se ha convertido en una de las soluciones para el backend por excelencia.

En esta plataforma se puedan crear aplicaciones Android, iOS, Web y Unity, todas en un mismo proyecto. Además, al ser propiedad de Google, es el perfecto complemento para proyectos Flutter, ya que hay plugins compatibles para Flutter de todos los servicios.

Además, la consola o dashboard de Firebase es muy amigable y accesible, otra ventaja a tener en cuenta. Los servicios o herramientas se dividen en: compilación, lanzamiento y supervisión, analytics, participación y extensiones (la mayoría en versión beta).

Dentro de su amplia oferta, en este proyecto se han integrado los siguientes servicios en el proyecto Flutter:

- Autenticación

  Es un servicio de backend utilizado para gestionar el acceso de usuarios a la app de forma totalmente segura y privada, generando tokens únicos e inmutables que sirven como identificador de usuario mientras la sesión esté activa. Esta herramienta da soporte a múltiples métodos de acceso, como via usuario y contraseña, link por correo electrónico, móvil, servicios de terceros (Google, Facebook, Twitter, Github, etc) o anónimo. Además, gestiona otros eventos relacionados con la autenticación del usuario como eliminación de cuenta, cambios de contraseña o de correo electrónico.

- Cloud Firestore

  Es una de las dos soluciones cloud multiplataforma (junto con Realtime Database) que Firebase ofrece para almacenar datos no estructurados en tiempo real, independientemente de la latencia de la red o la conectividad a Internet.
  Cloud Firestore se estructura en colecciones (y subcolecciones) y documentos.

  Una colección es un conjunto de documentos, como la carpeta raíz.
  Los documentos son mapas clave (o campo)-valor, donde se almacenan los datos, con un identificador único dentro de la colección a la que pertenecen.
  Dentro de un documento se pueden añadir subcolecciones, que contendrán a su vez otra lista de documentos, creando así relaciones jerárquicas. Hay que tener en cuenta que cuanto más largas sean estas relaciones, más complejas serán las consultas.

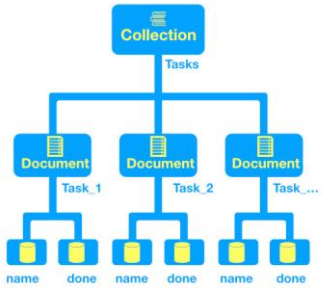


Ilustración 15 - Estructura Cloud Firestore (code.tutsplus.com)

Los tipos de datos que soporta Firestore son: String, boolean, número, vector, mapa, referencia (a un documento en una ruta concreta), geopoint, timestamp y null.

Esta base de datos NoSQL, como se explica en [28], se caracteriza por su flexibilidad y escalabilidad. Firestore ofrece un modelo de datos más intuitivo y accesible que Realtime Database, que organiza sus datos en un árbol JSON jerárquico, el cual puede volverse tedioso de escalar cuando el número de datos es grande y complejo. Además, Firestore permite consultas, clasificaciones y ordenaciones más completas. Por otro lado, Cloud Firestore posee reglas de seguridad más robustas y escala de forma automática sin limitación de usuarios simultáneos, a diferencia de RT Database, que solo permite 100.000 usuarios a la vez.

- Cloud Storage

  Es el servicio que permite alojar objetos en la nube. Normalmente, se utiliza para almacenar fotos, vídeos, archivos y otro tipo de documentos online, con copias de seguridad.

- Firebase Hosting

  Es el servicio que permite alojar contenido web en Internet. Gratuitamente, Firebase provee de un dominio finalizado en '.web.app' o '. firebaseapp.com'. Se pueden

utilizar targets o identificadores cortos para agrupar recursos estáticos únicos de Firebase Hosting, entre otras cosas.

- Dynamic Links

  Esta herramienta sirve para redireccionar al usuario a un link específico, llevándose consigo toda la información necesaria para poder acceder a él. Esta técnica se conoce como Deep linking en desarrollo web. En la consola de Firebase se pueden gestionar estos links mediante prefijos URL y URLs cortas, y consultar el número de clicks, entre otras cosas.

En cuanto a planes de facturación, Firebase tiene dos: el Spark, gratis, y el Blaze, pago por uso. Hay importantes diferencias entre el primero y el segundo: en el plan gratis hay cuotas máximas de uso para cada servicio, pero en el de pago no hay límite, si bien se incluyen cuotas diarias gratuitas. Además, con el plan de pago se incluye el proyecto en la plataforma de Google Cloud, de la que se hablará a continuación. En este proyecto se utiliza el plan Blaze, de lo que se habla con más detalle en el Capítulo 4.

Los pasos para crear un proyecto de Firebase y configurar distintas apps se encuentran en el Anexo II.

2.2.3.1 Google Cloud Platform

Es importante destacar que cuando creamos un proyecto en Firebase, lo estamos creando con la misma cuenta en la nube de Google, cuya plataforma es Google Cloud Platform (GCP). Esta ofrece infinidad servicios de todos los niveles de Cloud Computing (IaaS, PaaS, BaaS y SaaS), divididos en computación, almacenamiento, Big Data y Machine Learning. GCP es competidor directo de Amazon Web Services (AWS), Microsoft Azure, IBM Cloud, Oracle Cloud o Alibaba Cloud, entre otras infraestructuras cloud.

Ilustración 16 - Logotipo GCP

GCP integra la plataforma Firebase con el plan de pago por uso, y además de compartir infraestructura, también tienen en común. comparten los siguientes servicios: Cloud Firestore, Cloud Functions y Cloud Storage. El SDK de Firebase actúa como cliente en GCP, solicitando acceso a los distintos servicios conforme el proyecto va escalando. Se pueden construir flujos Firebase (SDK cliente)-GCP (SDK server) muy complejos, como el siguiente, donde se muestra el soporte de la infraestructura para el ecosistema IoT y cómo el usuario final puede recuperar de forma visual la información recogida de los SoCs:

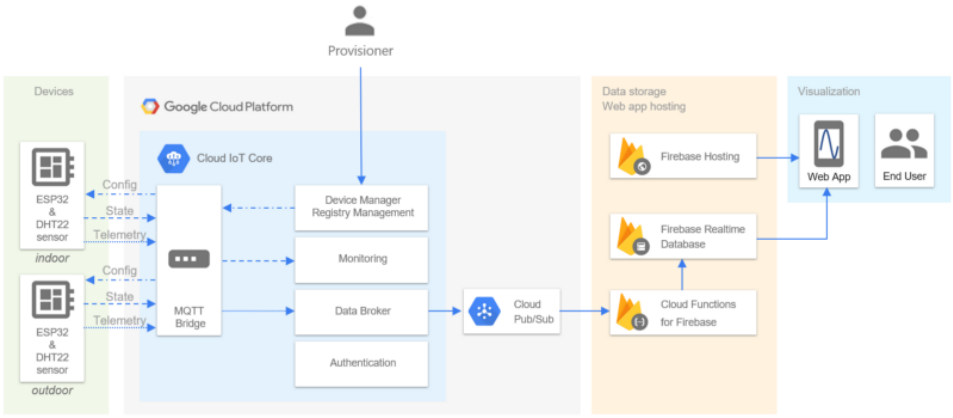


Ilustración 17 - Ejemplo proyecto Firebase & GCP (freecodecamp.org)

Los cambios realizados en una de las dos consolas se reflejan en la otra. Ya que la consola de Firebase resulta más amigable y fácil de usar, solo se ha utilizado esta plataforma durante el desarrollo, si bien se ha comprobado que el proyecto estuviera perfectamente desplegado en GCP, y la información de la BBDD sincronizada, entre otras cosas, como se tratará en el capítulo 5.

2.2.4 ENTORNOS DE DESARROLLO SOFTWARE

- Android Studio (versión 4.1.1)

Ilustración 18 - Logotipo Android Studio

Android Studio es el entorno oficial o IDE para el desarrollo de apps con el sistema operativo Android (propiedad de Google). Permite crear un emulador Android para probar la aplicación (AVD) a la vez que reconoce y permite ejecutar la aplicación via debug en un dispositivo hardware Android conectado al ordenador por USB, que es lo que se ha hecho en este proyecto.

Además, está desarrollado con el software de Intellij IDEA (JetBrains), por lo que el interfaz es muy parecido, y se utiliza de la misma forma. [29]
En este proyecto, se han utilizado ambos indistintamente.

- Intellij IDEA Ultimate (versión 2020.2.4)

Ilustración 19 - Logotipo Intellij IDEA

Intellij IDEA es un IDE desarrollado por JetBrains y da soporte a múltiples lenguajes y frameworks, por lo que es una de las herramientas más completas para desarrollo software del mercado.

Actualmente, se ofrecen dos versiones de este IDE: Community Edition, gratuita y open-source, incluye todos los servicios básicos para JVM y desarrollo Android, y Intellij IDEA Ultimate, versión comercial distribuida, incluye herramientas para web y desarrollo empresarial exclusivas (.NET, DevOps, entre otras).

En este proyecto se utiliza la versión Ultimate con licencia de estudiantes (se explica más en el Anexo II), y se ha ejecutado la app via debug con un dispositivo físico.

Ha sido utilizado en otras asignaturas anteriores para desarrollo web, por lo que no es necesario familiarizarse con el entorno en sí, sino con los plugins de las tecnologías.

- Sublime Text 3

Editor de texto gratuito y sencillo que admite código de múltiples lenguajes. Es especialmente útil para programación web. Se ha utilizado para crear los HTML y para realizar pequeños cambios en el código, en los que no fuese necesario ejecutar la app, ya que este programa es muchísimo más ligero que los otros IDEs.

2.3 *TECNOLOGÍAS HARDWARE*

Para el desarrollo del proyecto, se ha necesitado: un portátil para instalar los IDEs y ejecutar la app, dos smartphones Android con distintas versiones de la API para probar el funcionamiento de la app via debug (USB) y la tecnología beacon, con dos SensorTag.

El portátil utilizado es un HP modelo ENVY 13-d008na (2017) con un procesador Intel Core i5-6200U a 2.30 GHz y 8GB de RAM, entre otras características. Tras unas pruebas iniciales con el AVD de Android Studio, el emulador no era lo suficientemente rápido para detectar cambios en tiempo real del back-end, por lo que se optó por utilizar dispositivos Android físicos, y así hacer testing directo de experiencia de usuario.

Para ello se utilizaron dos móviles Samsung de distintos modelos, con versiones de Android distintas, para comprobar el correcto funcionamiento de la app. En ambos casos, se debe cumplir que el minSDKversion de la app Android, que es el nivel API mínimo al que la app se ejecutará (encontrado en la ruta android/app/build.gradle, en este caso es 19, que equivale a la versión Android 4.4) sea menor o igual que la version del SDK del dispositivo.

El primero y más moderno es un Samsung S10e, con procesador Exynos 9820. Inicialmente viene incluida la versión Android 9.0 (API 28), y actualmente se ha actualizado a Android 11.0 (API 30), que es la última versión disponible. El segundo y más antiguo es un Samsung Galaxy S6, con procesador Exynos 7420. Inicialmente viene incluida la versión Android 5.0 (API 21), y se ha actualizado a Android 7.0 (API 24). Como se puede observar, todas estas APIs son mayores que la mínima requerida.

2.3.1 TECNOLOGÍA BEACON

De acuerdo con [30], una baliza electrónica o beacon es un dispositivo fijo de tamaño pequeño (hasta del tamaño de una moneda), económico (precio medio menor de 10 euros, depende de las especificaciones) y que posee una pequeña pila de botón como fuente de alimentación, la cual puede durar años. Asimismo, los beacons son compatibles de forma nativa con la mayoría de sistemas operativos del mercado, si bien estos deben ser compatibles con la versión Bluetooth 4.0, o superior.

Ilustración 20 - Estructura Beacon

Estos dispositivos utilizan principalmente tres protocolos basados en BLE: iBeacon (Apple, compible solo con el ecosistema Apple), EddyStone (Google, open-source), y AltBeacon

(Radius Network). Como se explica en [31] independientemente del protocolo que se utilice, un beacon debe emitir las siguientes dos características, controladas por el firmware: potencia de transmisión fija, y periodo de emisión (advertising Interval). En el caso de iBeacon, se define el UUID como parámetro identificador, mientras que con EddyStone, el identificador se llama EddyStone-UID. También este protocolo define un identificador encriptado que cambia de forma periódica (EddyStone-EID).

La comunicación entre la baliza y otros dispositivos se realiza mediante el envío constante de una señal broadcast BLE, y previamente a iniciar la misma es necesario configurarla. Esta señal es de corto alcance (entre 40-70 metros, pero hasta 100 metros en pruebas exprimentales) y puede ser recibida e interpretada por otros dispositivos inteligentes, como smartphones o tablets que tengan soporte para Bluetooth 4.0 o superior. Cabe destacar que los beacons son elementos pasivos que no emiten más información que un identificador universal (UUID), entre otros bytes, que sirve para que el receptor-que tiene papel de oyente-pueda localizarlo con gran exactitud debido al corto alcance de la señal.

Su principal función es "despertar" a otros dispositivos que estén a la escucha, y para ello suele utilizarse una app móvil (beacon mobile SDK) que reconozca la señal que emiten, y se desencadene una acción determinada en ella, dando un caso de uso a la comunicación, como por ejemplo calcular la posición exacta del dispositivo o mostrar al usuario cierto contenido desde el servidor. Además, ya que esta comunicación BLE es de una sola vía, es muy común analizar y gestionar las señales mediante una infraestructura cloud como back-end de la app.

Como ya se ha explicado, la tecnología BLE comparte banda de 2.4 GHz con otras tecnologías inalámbricas, lo que podría derivar en interferencias en la transmisión de datos. Sin embargo, como se explica en [32], ya que los beacons tienen únicamente un rol pasivo, no se crean conexiones ni transmisiones de datos bidireccionales, y solo hacen uso de los canales de advertising para "despertar" a otros dispositivo, los cuales están estratégicamente situados para no solapar con otras tecnologías inalámbricas coexistentes en la banda. Por tanto, un ecosistema beacon no crea interferencias en sí mismo.

Las aplicaciones de la tecnologías beacon son infinitas y se pueden utilizar en cualquier espacio, tanto indoor como outdoor (espacios públicos, restaurantes, museos, kioskos, cines, tiendas, etc). Las aplicaciones más comunes, como se indica en [9] son:

- Indoor tracking: permite controlar el acceso de los trabajadores a una empresa o monotorizar el inventario. Según [10], la empresa Cisco, entre otras, ha comenzado a implementar este sistema hace unos años.
- Asset tracking: sirve para localizar las pertenencias con bastante precisión.
- Retail: mejora la experiencia de usuario en los establecimientos comerciales enviando información exclusiva. Además, con la información aportada de los beacons se podría generar un *heat map* para entender cómo se comporta el cliente. Esta técnica tiene como resultado una fidelización del cliente. En este proyecto, se utiliza este caso de uso para llevar a cabo una estrategia de marketing de proximidad.

Esta tecnología, presente en el mercado desde 2013 con la presentación del protocolo iBeacon, ha ido creciendo en popularidad con la aparición de otros protocolos, y su uso en diversas aplicaciones ha ido en gran aumento.

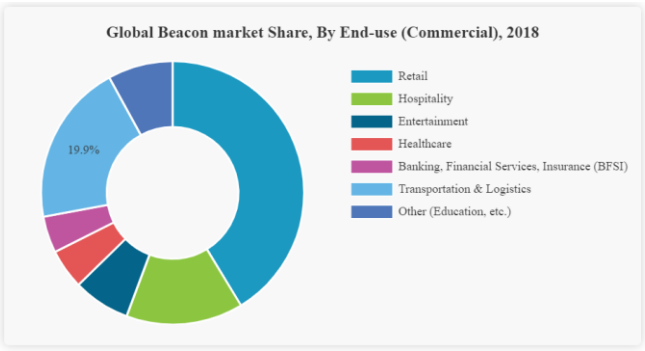


Ilustración 21 - Global Beacon Market Share 2018 (Fortune Business Insights)

Según datos de [33], el mercado global de beacons en 2018 era de 1.36 mil millones de dólares, y se estima un crecimiento compuesto anual del 48.9%, llegando a los 31.61 mil millones en 2026. Además, el segmento de beacons gestionados en la nube, crecerá exponencialmente de 242.8 millones de dólares en 2018 a ganancias de 23.3 mil millones en 2026.

Esta tecnología está creciendo especialmente en el sector industrial y empresarial, tanto en verticales no orientadas al retail (especialmente IoT, industrias y casas conectadas), como verticales retail (pequeños comercios, centros comerciales) y con un especial auge en el tracking de posesiones personales. [34],

Hoy en día, la tecnología beacon se puede encontrar no solo en un dispositivo físico dedicado, si no integrada en otros, como por ejemplo wearables, en formato pulsera, colgante o llavero. Este año, Apple ha sacado AirTag, la cual es una baliza de localización de pequeño tamaño, en formato llavero, que sirve para localizar los dispositivos del ecosistema Apple de un usuario, de forma segura y cifrada.

2.3.1.1 CC2650STK SensorTag

Ilustración 22 - CC2650STK SensorTag (Texas Instruments)

El CC2650STK SensorTag es un dispositivo desarrollado por Texas Instruments y perteneciente al ecosistema IoT. Dispone de un chipset con 10 sensores: luz, micrófono digital, sensor magnético, humedad, presión, acelerómetro, giroscopio, magnetómetro, temperatura de obeto y de ambiente.

Además, permite conectividad cloud mediante BLE para gestionar los datos online de los sensores a través de una app móvil compatible tanto para iOS como para Android.

Sus características principales son las siguientes:

- Wireless MCU CC2650

  Este microcontrolador [35] es parte de una familia de dispositivos de potencia ultrabaja que operan en la banda 2.4GHz, que soporta múltiples protocolos

inalámbricos como Bluetooth (5.1), ZigBee y 6LoWPAN. Además, tiene un procesador ARM Cortex M3 de alto rendimiento y un controlador Bluetooth Low Energy, entre otras características. El modo bajo consumo de este microcontrolador permite el funcionamiento con baterías de tipo botón, ya que les proporciona una larga vida útil.

Se ha comprobado experimentalmente que el dispositivo se apaga a los 2 minutos aproximadamente. Con este temporizador también se fomenta las aplicaciones de bajo consumo y aumenta la vida útil de la batería.

- Batería CR2032

  Esta batería [36]es una coin-cell de litio, ideal para soluciones IoT debido a su reducido tamaño, su larga vida útil (90% de la capacidad inicial incluso después de 10 años), y su diseño de potencia ultracompacta.

- Funcionalidad beacon

  Este dispositivo [37] además de soportar comunicación BLE, incluye el protocolo iBeacon para crear aplicaciones móviles que puedan gestionar los datos del SensorTag y la localización física.

2.4 OTRAS HERRAMIENTAS

En este apartado se explican otras herramientas utilizadas para llevar a cabo el desarrollo del proyecto. Además de las que se exponen a continuación, la documentación de Dart, Flutter y Firebase (FlutterFire, Cookbook, Flutter Gallery, CodeLabs, Material Design) ha servido como guía durante todo el desarrollo. Sin embargo, debido a que en muchos casos esta era escueta o incompleta para las necesidades particulares que iban surgiendo, se ha hecho uso de blogs dedicados (Medium), foros (StackOverflow), videotutoriales (Youtube, Udemy o Codeacademy), páginas web y otros repositorios de Github públicos.

2.4.1 PUB.DEV

Pub es una herramienta online donde se pueden buscar o filtrar paquetes programadas en Dart compatibles con proyectos de Flutter, AngularDart y programas Dart en general.

En Pub.dev se pueden encontrar tanto plugins de Dart como paquetes en general. Como se indica en [38], es importante puntualizar que los conceptos de paquete y plugin son distintos, ya que un plugin es un tipo de paquete con unas características específicas.

Un paquete Dart es un directorio con un archivo llamado pubspec, el cual puede contener dependencias como librerías, otras apps, imágenes, tests, etcétera. Este paquete puede integrarse como dependencia en otro proyecto.

Por otro lado, un plugin es un paquete especial que habilita una funcionalidad de un sistema operativo concreto en la app (Kotlin, Swift, web, etc), como por ejemplo utilizar la cámara del dispositivo.

Como ya se ha explicado, una de las ventajas de Flutter es la rapidez con la que se pueden construir apps con su sistema de paquetes/librerías. Con esto, el framework está abierto a las aportaciones de la comunidad de desarrolladores, por lo que de forma colaborativa se van creando paquetes que satisfacen las necesidades de la comunidad. Como resultado, el framework implementa mejoras y "crece" rápidamente.

Estos paquetes compatibles con Flutter se encuentran en el "catálogo" Pub.dev. En esta página hay paquetes publicados por el equipo de Flutter y también por desarrolladores independientes. Para que un paquete sea publicado, el equipo de Flutter hace un análisis de este, puntuándolo (sobre 130 puntos) según métricas como ritmo de actualizaciones, nivel de documentación o calidad de código. Los paquetes con mayor puntuación se muestran como Flutter Favorites. Normalmente, suelen ser paquetes desarrollados por el equipo de Flutter.

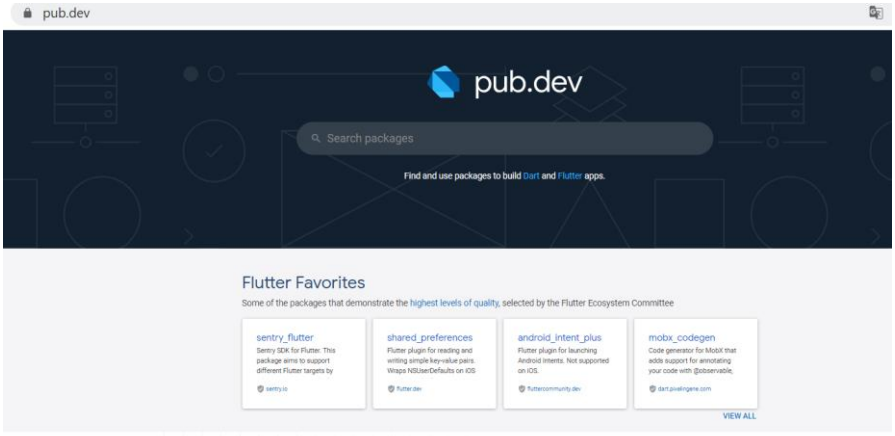


Ilustración 23 - Pantalla Pub.dev

Para cada paquete, además de la puntuación, se puede ver la popularidad en porcentaje, el número de likes y el repositorio Github donde se aloja el código.

Un ejemplo de un paquete de un desarrollador independiente, muy relevante en este proyecto es el siguiente:

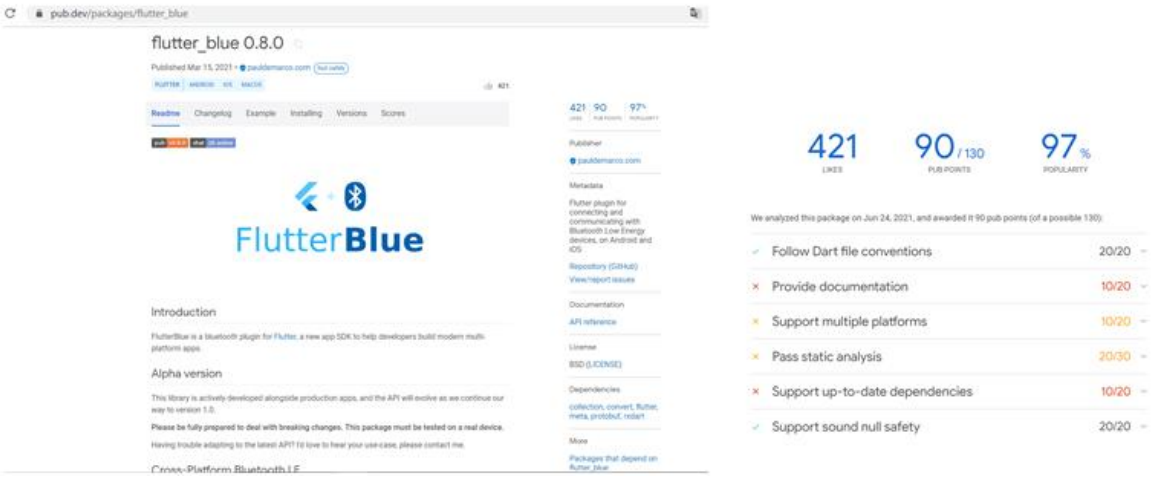


Ilustración 24 - Paquete FlutterBlue en Pub.dev

Como se puede observar, tiene menos puntuación. Ha tenido que pasar los tests del equipo de Flutter, los cuales se van actualizando. Con los resultados de los tests, el desarrollador puede mejorar los puntos débiles del paquete, ya que se le da feedback de los fallos existentes.

2.4.2 GIT

Ilustración 25 - Logotipo Git

Git es un software open-source de control de versiones distribuido, diseñado para desarrollar desde pequeños a grandes proyectos con rapidez y eficiencia. [39]

Es comúnmente utilizado en proyectos de desarrollo software colaborativos, en los que los miembros de un equipo trabajan sobre el mismo código simultáneamente, a veces conectados desde distintas redes. Este sistema permite que los miembros puedan seguir el flujo o secuencia de modificaciones realizadas por otros, y así tomar decisiones sobre su código que no afecten al trabajo ya realizado.

El funcionamiento de Git es el siguiente: al crear un repositorio (se sube el código a Git por primera vez), se crea una rama o branch principal (master). Desde esta, se pueden crear bifurcaciones o subramas, con el código duplicado de la rama master, pero independientes a esta (sistema descentralizado). Con esto, un miembro del equipo solicita una bifurcación de la rama principal con el código conjunto, y trabaja sobre este haciendo las modificaciones pertinentes sin afectar al código principal. Una vez se comprueba que funciona, se fusiona la subrama en la rama master, y se integra el código. El resto de miembros del equipo puede ver todos estos movimientos en orden cronológico.

2.4.3 GITHUB

Ilustración 26 - Logotipo GitHub

Github es uno de los proveedores de hosting de repositorios en la nube más utilizados. Se define técnicamente como una forja, que significa plataforma de desarrollo colaborativo, ya que, ofrece una interfaz gráfica sencilla de utilizar para la gestión de repositorios por parte de equipos y también de usuarios individuales.

Mediante el sistema de versiones Git, se pueden consultar rápidamente solicitudes de bifurcación de la rama principal, los commits y modificaciones de estas, comentarios, ratings, etcétera. Se considera como una red social, ya que los usuarios pueden abrir hilos de conversación o de fallas (issues) para colaborar entre ellos. Cualquier persona puede inscribirse y alojar repositorios públicos (o de acceso restringido) de forma gratuita, lo que hace de Github una herramienta tan popular.

En el caso particular de este proyecto, esta herramienta ha sido muy útil, para que el avance del desarrollo se pudiese seguir fácilmente por parte del director. Solo se ha utilizado la rama principal, ya que es un proyecto de carácter individual.

Además, familiarizarse con esta herramienta es muy necesario ya que el SDK de Flutter y sus distintos canales se encuentran en alojados en repositorios Github, así como las librerías y múltiples repositorios con códigos de ejemplo. Los mismos desarrolladores de Flutter van añadiendo hilos de issues cuando se encuentran algún problema al ejecutar código, lo cual es muy habitual, sobre todo cuando se utiliza el canal developer de Flutter, como es el caso de este proyecto.

El código completo de este proyecto se encuentra en el siguiente repositorio público:

https://github.com/teeterls/myBlueAd

Capítulo 3. ESTADO DE LA CUESTIÓN

Se define marketing de proximidad como [40] un ***"conjunto de técnicas en las que se utilizan las nuevas tecnologías para contactar con un usuario en función de su ubicación concreta y llevar a cabo una acción determinada."*** Esta estrategia experiencial ha evolucionado progresivamente a un tipo de marketing móvil, ya que se opta frecuentemente por desarrollar aplicaciones móviles como medio de comunicación o interacción directa entre clientes y establecimientos.

Debido al contexto de competitividad tecnológica expuesto antes y el boom de las aplicaciones móviles, muchos comercios y empresas optan por implementar una estrategia de proximidad, que fomenta la personalización y unicidad con el objetivo de dar solución a retos como digitalizar la tienda física, mejorar el costumer journey, la omnicanalidad o conocer mejor a los clientes.

3.1 TECNOLOGÍAS APLICADAS AL MARKETING DE PROXIMIDAD

El marketing de proximidad se inicia con el reparto de folletos, y evoluciona al uso de distintas tecnologías. Para llevar a cabo esta estrategia, las tecnologías más comunes son: Wi-Fi, Bluetooth (beacons), QR, NFC, geolocalización (GPS) o SMS. A continuación, se explican sus principales características y se ofrece una breve comparativa entre ellas y los beacons, con información sintetizada de [40], [41] y [42].

- El envío de SMS a clientes con contenido exclusivo es una forma de contactar con ellos rápida y sencilla, pero antes de hacerlo, los mismos deben haber rellenado algún tipo de formulario web o haber comprado una tarjeta de fidelización. Además, para que se pueda localizar al cliente y enviarle información según su ubicación hay que combinar esta técnica con otras.

- Los códigos QR o bidimensionales permiten que los usuarios accedan a cualquier página o contenido estático en la web sin tener que introducir la URL, entre otras acciones. La mayoría de los dispositivos inteligentes soportan lectura de QR. Sin embargo, para escanearlos el dispositivo debe estar a escasos centímetros y es más incómodo para el usuario, puesto que o debe abrir la cámara o descargarse una app específica de escaneo de QR.

- Tarjetas NFC: tecnología inalámbrica de corto alcance (menos de 20 cm) que deriva de las etiquetas RFID, las cuales funcionan por radiofrecuencia. Están pensadas para los dispositivos móviles, pero no todos las soportan. Las ventajas con respecto a los QR es que ser diseñados con cualquier forma e integrarla en objetos, por lo que es muy útil para casos de uso de indoor tracking, como inventarios, o asset tracking. Además, la conexión es instantánea sin emparejamiento previo, una ventaja sobre la tecnología beacon. Son idóneas como método de pago con datáfonos, identificación en transporte público o sincronización de dispositivos. Sin embargo, su corte alcance es una gran desventaja para la experiencia del cliente.

- GPS: técnica de geolocalización que puede conocer la ubicación de cualquier dispositivo en la Tierra, gracias a la trilateración y el uso de satélites. Este sistema es el más utilizado en el marketing, en concreto la técnica *geotargeting*, el recurso favorito de las redes sociales, que utiliza los datos GPS y la dirección IP para enviar anuncios y contenidos según la ubicación del usuario (con su previo consentimiento). El sistema de posicionamiento global posee gran exactitud en los espacios abiertos, sin embargo, en los espacios *indoor* es recomendable combinarlo con alguna otra tecnología LBS (location based service). Con la tecnología beacon se puede localizar dispositivos en aquellas zonas donde GPS pierde cobertura, como por ejemplo un túnel.

- Wi-Fi: es la tecnología de acceso inalámbrico más utilizada hoy en día, con un alcance de hasta kilómetros si se utilizan repetidores, los cuales son muy

recomendados para espacios *indoor* debido a la presencia de obstáculos. Aporta un alto nivel de seguridad, ya que la red Wi-Fi puede ser privada (o pública), y solo tener acceso a ella clientes que se hayan registrado a una app web o móvil de la tienda, o que hayan rellenado algún tipo de formulario de fidelización.

Con esta red los usuarios no gastan datos móviles, lo cual aporta valor añadido al cliente, y cuando se conecten a la red se les podrá enviar cualquier información. Para la localización en interiores se puede utilizar Wi-Fi Location, pero es muy sensible al tráfico de la red, por lo que si hay muchos clientes conectados, la comunicación puede ser de baja calidad. Una desventaja con respecto a BLE es el mayor consumo de energía, y que el despliegue de dispositivos Wi-Fi tiene un coste mucho más elevado. Cabe destacar que hay beacons que combinan BLE y Wi-Fi, pero son más caros.

3.2 *EJEMPLOS*

Como se ha explicado, tanto el marketing de proximidad como el uso de los beacons están más extendidos. Grandes empresas tecnologías ofrecen sistemas de marketing de proximidad y de posicionamiento indoor.

Ilustración 27 - Principales proveedores de beacons

Beaconstac es una empresa que vende hardware para estrategias de marketing de proximidad: beacons, QR, Geofencing y NFC tags. Además de ser proveedor de dispositivos físicos, también ofrece servicios software de pago para la gestión de estas estrategias (dashboards), así como apps WhiteLabel para crear aplicaciones personalizadas compatibles con beacon.[43]

Es además la empresa desarrolladora de la aplicación móvil NearBee, disponible en iOS y Android, que pretende sustituir a las notificaciones Google Neaby. Utiliza el protocolo EddyStone-URL para detectar balizas próximas y mostrar notificaciones en el dispositivo. Es una solución enfocada al retail. Sin embargo, no ha tenido mucha penetración en el mercado. Además, el UI no es muy atrayente, y los contenidos asociados a los beacons detectados se muestran como elementos en una lista, en vez de aparecer como un pop up o ventana emergente, algo que no llama la atención. Cuando se pulsa uno de estos elementos, la app te redirige a una página web.

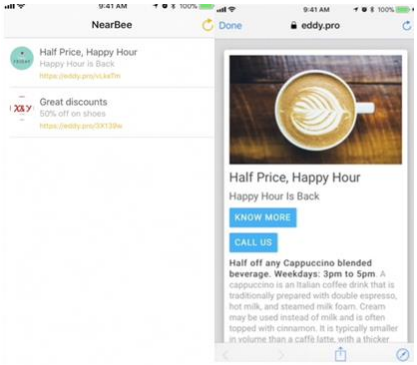


Ilustración 28 - Capturas App Nearbee

Las empresas Quuppa y Mildmac, líderes en posicionamiento de interiores, se han asociado para desarrollar una plataforma de pago RTLS (Real Time Location Service) llamada BTrazing [44] para localización de cualquier tipo de activo (objetos y personas), basada en beacons BLE gestionados en cloud. Para lograr una localización exacta, se utiliza la tecnología beacon en dos dispositivos: tags portables por los activos, y locators que tienen el rol de antena receptora, la cual detecta hasta el ángulo de llegada de la señal BLE.

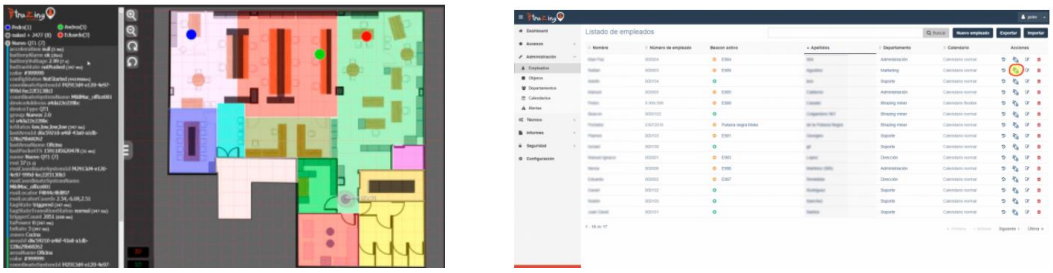


Ilustración 29 - RTLS BTrazing

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Con este proyecto, se exploran las tecnologías del momento, las cuales son tendencia en el proceso de transformación tecnológica y digital: desarrollo de aplicaciones móviles multiplataforma, IoT y Cloud. Por esto, se ofrece un producto económico, que satisface las necesidades tecnológicas del presente, y servirá como un impulso para aquellos establecimientos del sector retail, que no están muertos, si no obsoletos. La pandemia ha acelerado este proceso de cambios, y muchos de estos comercios que no pueden seguir este ritmo, se encuentran en situación de quiebra y cierre próximo.

Además, actualmente, se han observado cambios de paradigma en el comportamiento del consumidor. Según [45] el concepto de tienda está cambiando con la tendencia offline-to-online. Los consumidores, que ahora tienen el control, buscan experiencias omnicanales, en los que la experiencia emocional que supone comprar en una tienda física sea complementada y mejorada por medios digitales.

Asimismo, la pandemia ha fomentado la vuelta a los comercios locales y pequeños establecimientos próximos (comercio de proximidad), si bien también prácticas como el cocooning (pasar más tiempo en casa) han disparado las compras online. La proximidad y la experiencia son una oportunidad, y las barreras de retailer – no retailer se están desdibujando gracias a la incorporación de nuevas tecnologías al sector. [46]

Asimismo, las múltiples aplicaciones y ventajas del beacon, como son el uso de BLE y su eficiencia energética, el precio económico y la sencillez de integrar en una solución mobile, hacen de esta una herramienta ideal para atraer clientes e impulsar las ventas del sector retail.

En el presente proyecto, se resuelve la problemática del contenido spam, una de las críticas hacia la tecnología beacon más recurrente y por el cual Google canceló en 2018 el soporte

de Google Nearby Notifications (EddyStone protocol) en dispositivos Android, ya que en todo momento el usuario puede elegir si quiere recibir contenido, cuando recibirlo, y qué hacer con dicho contenido. Una vez el usuario sale de la aplicación (no necesariamente cerrando sesión), no recibe ninguna notificación push.

Por otro lado, como se explica en [47], llevar a cabo esta estrategia en un negocio, supone, además de un elemento diferenciador tecnológico, una mejora experiencia del cliente, ya que además de proporcionarle información útil en el momento adecuado, se aporta un valor añadido al pensar en sus intereses y su comodidad a la hora de comprar. A través de la app móvil, los clientes tienen acceso a promociones y ofertas especiales de forma rápida y sencilla, además de personalizada según su ubicación. Esto puede facilitar enormemente su decisión a la hora de comprar un producto u otro, o ir a la sección de artículos que tenga mejores descuentos. También los clientes pueden dar feedback al establecimiento de lo que les interesa y lo que no, mejorando la comunicación cliente-establecimiento dentro de una estrategia de marketing omnicanal.

En definitiva, la intención de este proyecto es desarrollar una estrategia de marketing *win-win*, que por un lado mejore la experiencia de compra de los clientes y por otro lado aumente los beneficios de los establecimientos.

4.2 OBJETIVOS

Este presente trabajo es el inicio de un proyecto con un alcance mucho más amplio, a cumplir en el medio-largo plazo, cuyo propósito es desarrollar un sistema de marketing de proximidad completo basado en balizas BLE y su aplicación en el sector retail.

El alcance actual del trabajo se ha definido a partir de los siguientes objetivos, que han marcado la línea temporal de trabajo a lo largo de los meses.

4.2.1 OBJETIVOS GENERALES

- Aprendizaje del uso del framework Flutter para desarrollar aplicaciones móviles multiplataforma:
    - Conocer el funcionamiento del árbol de widgets y el uso de los widgets para el UI.
    - Conocer arquitecturas y patrones de Flutter.
    - Aprender sobre la gestión de estados.
    - Aprender a utilizar las dependencias e incluir librerías de terceros.
    - Aprender a integrar servicios del backend a la app móvil.

- Aprendizaje del lenguaje de programación orientado a objetos Dart, que es utilizado por Flutter.
    - Aprender las particularidades de la sintaxis con respecto a otros lenguajes de POO, como las variables nombradas.
    - Aprender a programar de forma asíncrona y utilizar elementos como Streams o Futures, muy necesarios para desarrollar apps en Flutter.

- Aprendizaje de la tecnología Cloud Firebase y sus múltiples servicios.
    - Aprender sobre la tecnología Cloud y sus ventajas.
    - Aprender cómo crear un proyecto y configurar aplicaciones (móvil, web), en concreto una app Android.
    - Conocer los servicios y herramientas para desarrolladores y cómo implementarlas en el proyecto Flutter : Autenticación, Cloud Firestore, Cloud Storage, Hosting, etc.

- Aprendizaje de la tecnología BLE y la comunicación entre dispositivos a través de la misma
    - Conocer las ventajas de BLE frente a otras opciones de tecnologías inalámbricas y de marketing de proximidad.
    - Cónocer como se comunican las balizas BLE con dispositivos móviles.

- Aprender a integrar la app móvil en un sistema de comunicación BLE.

❖ Crear un sistema de navegación mostrando contenidos personalizados con beacons.

❖ Aplicación de conocimientos y herramientas de Ingeniería de Software: metodología ágil, arquitecturas y patrones de diseño, diagramas UML y buenas prácticas de programación (técnicas clean code y testing)

4.2.2 OBJETIVOS ESPECÍFICOS

❖ Desarrollo de una aplicación móvil multiplataforma en Flutter para un sistema de marketing de proximidad mediante Balizas BLE.
- La app debe detectar las balizas mediante comunicación BLE, y mostrar al usuario un contenido específico asociado a la baliza más próxima.
- El contenido estará en una BBDD en la nube, y podrá sufrir modificaciones en tiempo real, los cuales serán alertados a la app.
- Se probará el funcionamiento de la app en distintos espacios físicos con tráfico Bluetooth variado.

❖ Desarrollo de una aplicación funcional, con un interfaz de usuario atractivo y user-friendly, así como accesible.

❖ Configuración del proyecto móvil en Firebase para otorgarle escalabilidad y utilizar algunas de las herramientas disponibles para desarrolladores: Autenticación, Cloud Firestore (BBDD en tiempo real), Cloud Storage, Hosting y Dynamic Links.

4.3 METODOLOGÍA Y PLANIFICACIÓN TEMPORAL

En proyectos de desarrollo software como este, la planificación temporal está muy influida por la metodología a seguir, como se explicará a continuación.

4.3.1 METODOLOGÍA

En este trabajo se sigue el enfoque **de metodología ágil**, forma de trabajo que es una práctica extendida en proyectos de software. Esta es muy útil en el desarrollo de proyectos en equipo, ya que se basa en agilizar los proyectos y completarlos con éxito dentro del *deadline* a través de la interacción humana y la coordinación efectiva de todos los miembros.

Además, estas metodologías (Scrum, XP, Kanban, etc.) siguen un ciclo de desarrollo adaptativo y en cascada, esto significa que el desarrollo del proyecto se va amoldando a las necesidades del cliente o a las condiciones del entorno, con el fin de obtener los mejores resultados y la mayor satisfacción del cliente.

Pese a que este proyecto es de carácter individual, se irán mostrando los avances en reuniones con el director, y para ello es necesario dividir el trabajo en tareas con una duración más o menos fija. La flexibilidad en la planificación es clave debido a la inexperiencia inicial en el desarrollo de aplicaciones móviles y en el uso de las tecnologías escogidas. Por ello, el uso de Scrum no es el más adecuado, ya que está muy orientado a proyectos en equipo y a dividir las tareas en *sprints* periódicos y cortos.

Por estos motivos, se ha decidido seguir la metodología Kanban, la cual es muy fácil de implementar, además de la más visual. Esta metodología se caracteriza por dividir el proyecto total en tareas muy específicas, cuyos resultados son fácilmente medibles. Consiste en el uso de un tablero en el cual se divide la lista de tareas principalmente en: pendientes, en proceso y completadas. Las tareas se van "empujando" desde el backlog o pila de tareas por el tablero Kanban a través de las distintas fases hasta ser completadas.

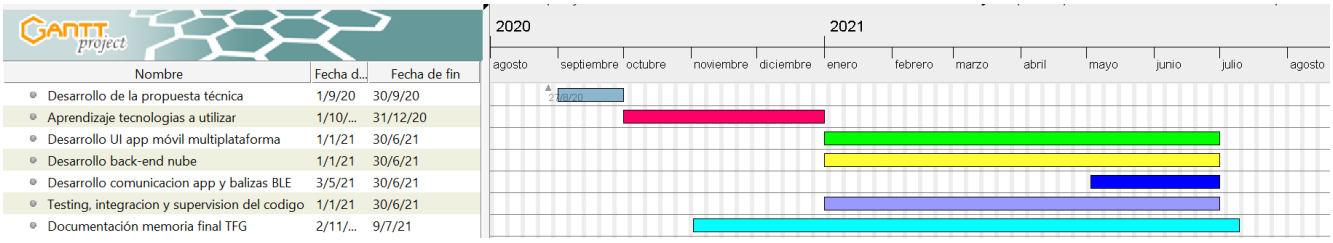
Se ha utilizado la herramienta Trello para crear tableros rápidos y manejables y organizar las tareas en las que se divide el proyecto. En dichos tableros se van añadiendo listas que reflejan el estado actual de las tareas. De esta forma es muy intuitivo ver la línea de trabajo. Las tareas se representan en tarjetas visuales de colores con su título, comentarios y duración. Un ejemplo del estado del tablero correspondiente al proyecto en el mes de abril es el siguiente:

Ilustración 30 - Tablero Trello Proyecto

4.3.2 PLANIFICACIÓN TEMPORAL

En cuanto a la planificación temporal, el proyecto ha tenido como duración un curso académico (aproximadamente 9 meses), dado el carácter anual de la asignatura del Trabajo Fin de Grado, y como se ha explicado su desarrollo se ha dividido en tareas específicas.

La planificación temporal global del proyecto se recoge en el siguiente Diagrama de Gantt, la cual es una forma muy visual de exponer el cronograma de un proyecto:



Nombre	Fecha d...	Fecha de fin
Desarrollo de la propuesta técnica	1/9/20	30/9/20
Aprendizaje tecnologias a utilizar	1/10/...	31/12/20
Desarrollo UI app móvil multiplataforma	1/1/21	30/6/21
Desarrollo back-end nube	1/1/21	30/6/21
Desarrollo comunicacion app y balizas BLE	3/5/21	30/6/21
Testing, integracion y supervision del codigo	1/1/21	30/6/21
Documentación memoria final TFG	2/11/...	9/7/21

Ilustración 31 - Diagrama Gantt Proyecto

Como se puede observar en el diagrama, se han englobado las tareas específicas en 7 etapas generales, con el fin de poder organizar el proyecto con una visión más amplia. A continuación se explican las mismas:

I. Desarrollo de la propuesta técnica

En esta etapa se recogen los primeros pasos, desde el primer contacto con el director hasta la elección del tema, los objetivos iniciales y alcance del proyecto, la decisión de qué tecnologías utilizar y la redacción del Anexo A.

II. Aprendizaje tecnologías a utilizar

Esta etapa ha sido una de las más importantes en el proyecto. En ella se ha realizado la búsqueda por distintas herramientas de soluciones a las necesidades el proyecto. En concreto, se ha aprendido a programar en Dart y utilizar Flutter con un nivel avanzado, y a integrar los servicios de Firebase en el proyecto.

Cabe destacar que esta etapa realmente ha sido continuada durante todo el desarrollo del proyecto, puesto que iban surgiendo nuevas necesidades y problemas a resolver conforme iba avanzando el trabajo. La curva de aprendizaje ha sido larga, debido a la inexperiencia inicial en el desarrollo móvil y tecnologías cloud para el back-end. Esto ha requerido flexibilizar la duración de algunas etapas para el tiempo necesario al aprendizaje.

III. Desarrollo UI app móvil multiplataforma

En esta etapa se diseña y desarrolla la vista de la aplicación con el framework Flutter y Dart. Se decide el diseño, los casos de uso y navegación del usuario, los widgets a utilizar en cada página acorde a la normativa Material Design, entre otros aspectos, con el fin de lograr una UI fácil de usar, funcional, a la par que bonita y atractiva.

IV. Desarrollo back-end nube

En esta etapa se configuran los servicios a utilizar de Firebase en la aplicación, y los métodos necesarios (en Dart) para acceder a ellos, así como recuperar la información y mostrarla con Flutter. Los servicios configurados son: Autenticación de usuarios para gestionar los distintos accesos y proveer

autenticación 3rd party, Cloud Firestore como BBDD en tiempo real, Cloud Storage para almacenar archivos e imágenes, Firebase Hosting para proveer de hosting a los establecimientos y desplegar páginas web con contenido a mostrar al usuario, y Dynamic Links para redirigir a los usuarios a ciertas página.

V. Desarrollo comunicación app y balizas BLE

En esta fase se despliega el sistema completo de comunicación BLE entre dos dispositivos integrando la app móvil con un sistema de balizas físicas. Se harán distintas pruebas de funcionamiento en el contexto de marketing de proximidad definido en el proyecto.

VI. Testing, integración y supervisión del código:

El propósito de esta tarea es llevar a cabo buenas prácticas de programación, entre ellas la práctica TDD (test driven development) que permite comprobar el correcto funcionamiento del código, por lo que a medida que va avanzando el desarrollo del mismo se realizarán distintos tipos de pruebas.

Además, se desplegará continuamente el código en un repositorio Github y se irán subiendo las modificaciones y ampliaciones (en la rama branch ya que es un proyecto individual).

VII. Documentación memoria final TFG

Esta etapa recoge el proceso de documentación de la memoria final del TFG y los distintos anexos, además de la presentación y preparación de la defensa del proyecto en la convocatoria fechada en Julio.

4.4 ESTIMACIÓN ECONÓMICA

En este apartado se analiza de forma aproximada y desde una visión global el coste económico aproximado del desarrollo del proyecto. En el futuro, se podrá estudiar la viabilidad de este a partir de esta estimación resultante, entre otros datos. Se ha estimado un coste total de **23366.01€**, el cual se ha dividido en: personal, hardware y software.

4.4.1 COSTES EN PERSONAL

El coste de personal se ha calculado sobre las horas totales del proyecto. Para este proyecto se han invertido aproximadamente 360 h. A partir de los datos de [48] sobre el salario mensual medio en España de un desarrollador software (experiencia media) y del salario de un jefe de proyectos, se han calculado los costes que se recogen en la siguiente tabla:

Concepto	**Detalle**	**Importe**
Desarrollador software full-stack perfil junior	20€/h x 360h	7200 €
Jefe de proyectos	40€/h x 360h	14400 €
		TOTAL: 21600 €

Tabla 1 - Costes de personal

En esta estimación se sugiere un jefe o supervisor de proyecto, si bien no es estrictamente necesario debido al alcance actual de este, pero de cara al medio plazo y pensando en la escalabilidad, será necesario contratar más desarrolladores, y es recomendable una persona dedicada a supervisar y organizar al equipo, así como el estado de las tareas y sprints que se vayan planificando.

4.4.2 COSTES EN HARDWARE

Para calcular los costes de los dispositivos hardware utilizados se necesitan los precios actuales de los mismos, puesto que las tecnologías se vuelven obsoletas y van perdiendo valor con el paso del tiempo.

En este proyecto, como se explica en el capítulo de tecnologías, se han utilizado un portátil (HP modelo Envy 13-D008na) , dos móviles Samsung (S10e y S6) y dos SensorTags (CC2650STK) con funcionalidad de baliza electrónica. Se desglosan los costes en la siguiente tabla:

Concepto	**Detalle**	**Importe**
HP Envy 13-D008na	950.42€	950.42€
Samsung Galaxy S10e/2SD	506.74€	506.74€
Samsung Galaxy S6	209.99€	209.99€
SensorTag CC2650STK	2* 49.43€	98.86€
		TOTAL:1766.01€

Tabla 2 - Costes de hardware

En cuanto al portátil, actualmente se encuentra fuera de stock en la mayoría de las tiendas de electrónica, y el precio utilizado en la tabla se ha encontrado en Amazon.

Como se ha comentado anteriormente, se han utilizado estos dos dispositivos móviles con versiones o APIs de Android de distinta antigüedad para comprobar el funcionamiento correcto de la aplicación vía debugging. Si se utilizan distintos modelos de la misma marca o de cualquier otra, el coste cambiará. Los precios de la tabla se han encontrado en Amazon, si bien dependiendo del distribuidor o proveedor de dispositivos estos pueden cambiar ligeramente.

En el futuro, si se agrega una app con sistema operativo iOS (Apple) al proyecto de Firebase, para probar dicha aplicación vía debug sería necesario utilizar un iphonw, cuyo coste, según el modelo utilizado, habría que incluir al total de costes hardware.

En cuanto a los dispositivos utilizados como balizas, los SensorTag, cuyo precio se ha encontrado en la página oficial de Texas Instruments [37] son unos sensores inteligentes con conectividad inalámbrica BLE, que además de tener incluida la tecnología iBeacon para funcionar como baliza electrónica, son dispositivos con soporte para múltiples sensores, conectividad a la nube, y otras muchas funcionalidades de IoT. Por tanto, su coste es mayor a una simple baliza electrónica.

Para llevar a cabo este proyecto, no es necesario un dispositivo tan completo como este, sino que se puede utilizar un dispositivo con un único módulo BLE y compatibilidad con algún protocolo de tecnología beacon (firmware). Los precios varían según el tamaño de este, la duración de la batería, el alcance, entre otras características, pero se pueden encontrar desde dispositivos de pila de botón por menos de 5€ (AliExpress), hasta kits económicos como los que ofertan empresas como Accent Systems.

4.4.3 COSTES EN SOFTWARE

Para el desarrollo software de este proyecto, se han necesitado: el framework Flutter (y lenguaje Dart) para desarrollar la app, la tecnología Firebase para el back-end en la nube, entornos de desarrollo (IDEs) para escribir el código y ejecutar la app, editor de texto ligero para hacer cambios rápidos , y la tecnología Github (y Git) para el despliegue continuo del código un repositorio.

A continuación, se desglosan los costes de estas tecnologías:

Concepto	**Detalle**	**Importe**
Framework Flutter (Dart incluido)	Open-source	0€
Firebase	Plan de pago por uso Blaze	0€ *
Android Studio versión 4.1	Gratis	0€
Intellij IDEA Ultimate versión 2020.2	Licencia para estudiantes anual con cuenta JetBrains	0€ *
Sublime Text 3	Gratis	0€
Github	Gratis	0€
		TOTAL: 0€

Tabla 3 - Costes de software

Como se ha explicado ya, Flutter es un SDK open-source creado por Google, por lo que desarrollar apps con esta herramienta es gratis.

En cuanto a Firebase y sus planes de facturación, en este proyecto se ha utilizado el plan Blaze de pago por uso para incluir el proyecto en Google Cloud Platform, pero debido al alcance de este no ha supuesto ningún coste, ya que las cuotas gratuitas establecidas son más

que suficientes. Con lo cual, hay que tener en cuenta que cuando el proyecto escale y se integren nuevos servicios, podría suponer un nuevo coste para el proyecto. Sin embargo, en el medio plazo se estima que el uso continúe siendo gratis ya que según la calculadora de pagos de la documentación oficial, las prestaciones que incluye el coste gratis mensual son más que suficientes (por ejemplo 1 GiB para la base de datos Cloud Firestore o 5GB en Cloud Storage).

En cuanto a los IDEs utilizados, cualquier versión Android Studio se puede descargar gratuitamente. Sin embargo, en el caso de Intellij IDEA, como ya se ha explicado hay dos ediciones con diferentes prestaciones: Community Edition (gratis) e Intellij IDEA Ultimate (comercial con periodo gratis de 30 días). En el caso de este proyecto se ha utilizado la edición Ultimate 2020.2, dentro del paquete completo de productos JetBrains, con una licencia para estudiantes a renovar anualmente, por lo que su uso ha sido gratuito. Sin embargo, según la documentación oficial, este paquete sin licencia saldría a 64.90€, coste a incluir el total.

Por otro lado, tanto Sublime Text como Github son herramientas gratuitas.

Capítulo 5. SISTEMA DESARROLLADO

En este capítulo se describirá el proyecto en detalle y el desarrollo de sus tres partes fundamentales: front-end, back-end y comunicación BLE con balizas. Se dividirá el capítulo en análisis, diseño e implementación.

5.1 ANÁLISIS DEL SISTEMA

En este apartado se explicará la arquitectura del sistema en detalle, cada uno de los componentes que lo forman y cómo se relacionan entre ellos.

Como ya se ha adelantado, el sistema de marketing de proximidad desarrollado se compone de: aplicación móvil multiplataforma (Flutter), actor/usuario de la app, conjunto de Balizas BLE SensorTag y lógica o back-end en la nube (Firebase) para gestionar la comunicación móvil-baliza y la interacción usuario-aplicación.

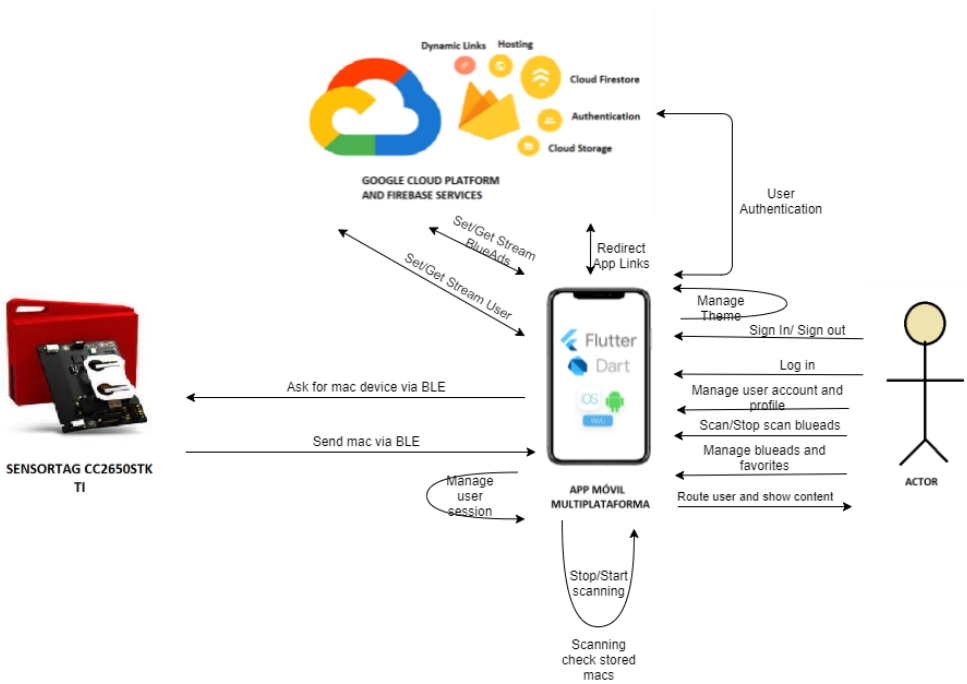


Ilustración 32 - Sistema desarrollado

❖ Actor

El usuario que accede satisfactoriamente a la aplicación móvil (cliente/consumidor actual o potencial) podrá disfrutar de contenido exclusivo (promociones, ofertas) según su ubicación en el espacio de un establecimiento. Según el método de acceso (con email o anónimo), podrá gestionar sus preferencias y crearse un perfil, entre otras funcionalidades. Los casos de uso se explican en detalle en el siguiente apartado.

❖ App móvil multiplataforma

Esta aplicación desarrollada en Flutter (disponible para iOS y Android con un UI prácticamente idéntico) interacciona con el usuario y con la/las Balizas BLE. Es la encargada de gestionar la sesión del usuario, el tema del UI, la navegación del usuario, el proceso de scanning y la comunicación con el back-end y sus servicios integrados. Los casos de uso se explican en detalle en el siguiente apartado.

❖ Back-end en Firebase (y GCP)

En la Plataforma de BaaS Firebase se configuran los siguientes servicios de back-end únicamente para una aplicación Android:

- Autenticación: se habilitan distintos tipos de acceso (email y contraseña, 3rd party, email link, teléfono móvil, anónimo y registro de nuevo usuario).
- Cloud Firestore: se crea una base de datos en la nube, con sincronización en tiempo real para gestionar colecciones de usuarios y de establecimientos (y contenidos).
- Cloud Storage: se habilita la subida de imágenes y archivos a la nube.

- Firebase Hosting: se crean dominios para el rol de administrador y de usuario. Este último se utiliza para mostrar contenidos en página web al usuario.
- Dynamic Links: se redireccionan a los usuarios mediante links cortos según determinados eventos.

Se profundizará en los aspectos clave de la implementación de estos servicios más adelante.

❖ Balizas BLE SensorTag

En este proyecto se utilizan 2 SensorTags con funcionalidad beacon. Como ya se ha adelantado, estos dispositivos inteligentes son compatibles con el protocolo iBeacon. Por ello, además de anunciar el nivel de potencia transmitida o el intervalo de anuncio (firmware), también anuncian el UUID (16 bytes, identificador de un gran grupo de balizas), el major (2 bytes, subconjunto más pequeño de balizas dentro del UIID) y el minor (2 bytes, identificación de baliza individual). En este caso particular, la red de Balizas BLE del mismo proveedor utilizada en todos los establecimientos tendría un UUID común. Dentro de un establecimiento concreto, el conjunto de balizas formaría un major. De este subgrupo, una baliza individual tendría su minor único.
Además, en cada comunicación BLE, estos dispositivos por defecto no utilizan LE Privacy, por lo que no cambian su Bluetooth MAC. En este proyecto se va a utilizar este valor como identificador único de cada dispositivo, ya que es muy fácil de obtener y de manipular.

En un sistema de marketing de proximidad basado en Balizas BLE, se reparten un conjunto de balizas por todo el espacio de un establecimiento (normalmente indoor), teniendo cada baliza una ubicación fija durante un periodo determinado. La ubicación de las balizas es estratégica, ya que la finalidad de este sistema es mostrar al cliente determinado contenido en función de la zona del establecimiento en la que se encuentre. Por lo tanto, habrá una baliza (o varias) ubicada en cada zona o sección de interés.

Ya que el usuario (y su dispositivo móvil) se irá moviendo por la tienda, su posición irá cambiando, y en cada momento se encontrará más próximo de una o unas balizas que de otras. El contenido que se le mostrará será el asociado a la baliza más próxima.

Un buen medidor de proximidad es el valor RSSI de la señal, que mide la intensidad de esta(valores negativos entre 0 y 120 decibelios). En este caso, cuanto mayor sea el valor RSSI (más cercano a 0) de la señal de una baliza detectado por la app móvil, más próximo estará el móvil de dicha baliza.

Intensidad de la señal (dBm)	Calificación
-30 dBm	Asombroso
-67 dBm	Muy bueno
-70 dBm	De acuerdo
-80 dBm	No es bueno
-90 dBm	Inutilizable

Ilustración 33 - Tabla intensidad RSSI (Speedcheck.org)

La comunicación entre app móvil y los SensorTags se recoge en un proceso llamado scanning. El scanning consiste en encontrar de forma iterativa dispositivos compatibles con BLE, obtener sus Bluetooth MACs y buscar alguna coincidencia entre las MACs encontradas y las Bluetooth MACs guardadas en la BBDD en la nube. Los pasos de esta búsqueda son los siguientes:

1. Se carga en tiempo real todo el contenido disponible de la base de datos (colección blueads, que se explicará más adelante) y se obtiene una lista con las Bluetooth MACs correspondientes.
2. El actor pulsa el botón para empezar la búsqueda, que está habilitado solo si el móvil tiene Bluetooth activado.
3. La aplicación escanea o descubre mediante BLE todos los dispositivos con Bluetooth activado (versión 4.0 o mayor) en un rango de aproximadamente 50 metros.

4. Por cada dispositivo escaneado o encontrado, obtiene su Bluetooth MAC (entre otra información anunciada) y busca dicha MAC dentro de la lista de MACs cargadas de la base de datos.
5. Si encuentra una coincidencia, la app mostrará al usuario un contenido asociado a dicha MAC. Si se encuentran varias coincidencias, se mostrará el contenido asociado a la MAC del dispositivo más cercano (señal BLE con mayor RSSI). Si no encuentra coincidencias, sigue buscando indefinidamente.

   Este proceso se repite en bucle hasta que el usuario haga click en el contenido mostrado o vuelva a la página principal.

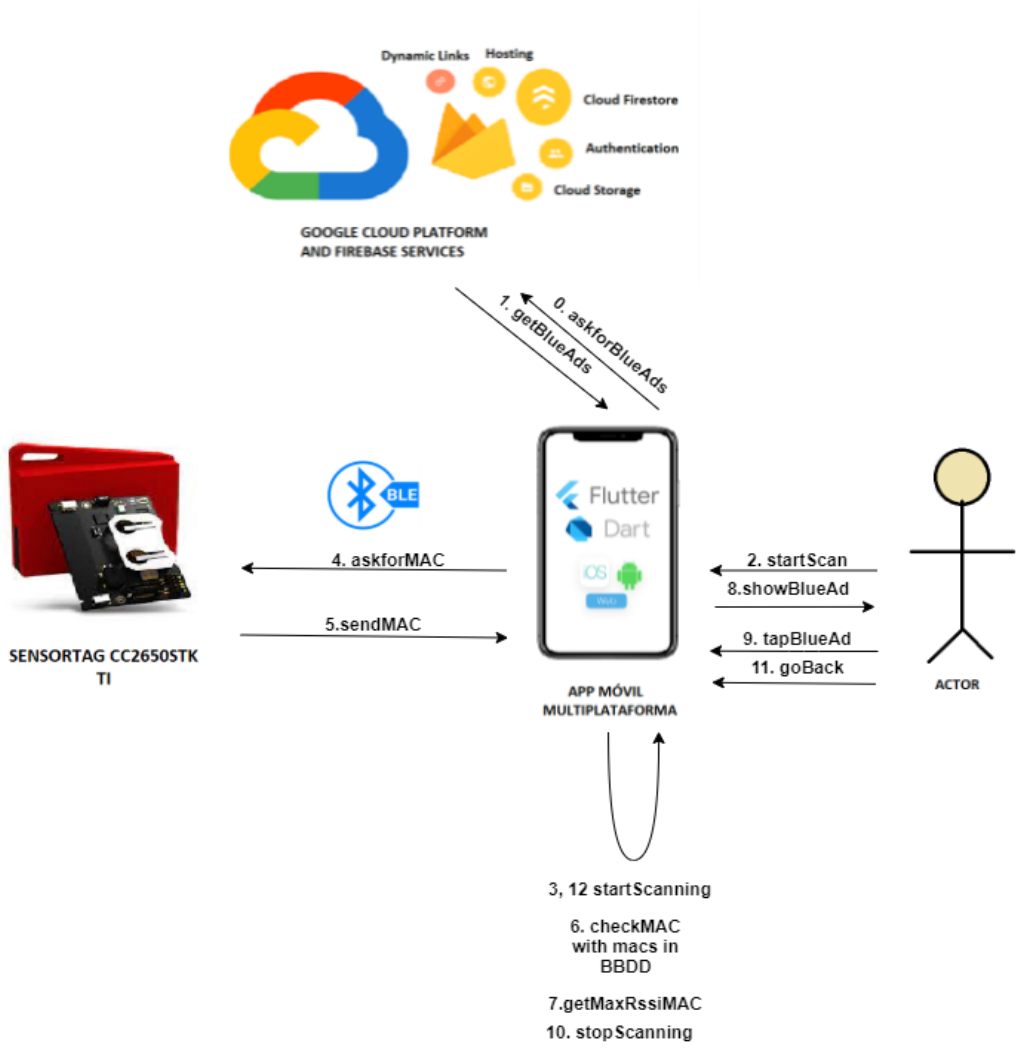


Ilustración 34 - Proceso Scanning

Para llevar a cabo las pruebas experimentales y comprobar el funcionamiento correcto de este sistema, se ha pensado en un tipo de establecimiento del sector retail en particular: tiendas departamentales (distintas secciones) con una superficie de como máximo 100 m2. El tamaño y distribución del espacio indoor condiciona la posición de las balizas, ya que el alcance de la señal broadcast que emite cada una puede llegar a los 50 metros. Entre las señales BLE procedentes de las balizas y señales inalámbricas de otros dispositivos (móviles de los clientes/usuarios, puntos de acceso Wi-Fi, ordenadores), en un momento determinado puede haber un gran tráfico de red inalámbrico en estos espacios indoor.

Ya que la app realiza el proceso scanning en bucle para buscar los beacons "guardados" en la base de datos, y el usuario está el movimiento constante, puede ocurrir que el usuario en un momento determinado no esté lo suficientemente cerca de una baliza determinada, o lo que es lo mismo, esté lo suficientemente cerca de varias balizas. En ese instante, la app puede "tardar" en calcular cual es la baliza más próxima, ya que le estarán llegando señales de varias o muchas balizas (además de las señales Bluetooth del resto de dispositivos). En ese corto periodo, la experiencia del usuario podría verse afectada ya que la app "dudaría" cual el contenido final a mostrar.

Por otro lado, en este proyecto no se han cubierto las posibles interferencias de las señales BLE en el proceso de scanning, las cuales son muy sensibles a obstáculos.

La implementación del scanning con Flutter y los SensorTag se explica más adelante.

5.2 DISEÑO

En este apartado se explicará el diseño y modelado de la arquitectura del sistema mediante múltiples diagramas. Se hace énfasis en la aplicación móvil, que es el core de este sistema y en los servicios back-end.

5.2.1 MODELO

Ya que Dart es un lenguaje POO, se fundamenta en clases y objetos o instancias para estructurar proyectos y modelar comportamientos (definiendo atributos o variables y métodos para obtener y cambiar estos atributos). En este proyecto, se utiliza este paradigma de programación para organizar la información y manipularla de forma sencilla a lo largo del código.

Las clases que modelan o estructuran el proyecto desarrollado son: Usuario, RetailStore y BlueAd. La base de datos en la nube (Cloud Firestore), es no relacional y orientada a objetos, ya que cada documento se trata como un objeto, en el que campos guardan el valor de los atributos de dicho objeto. Por tanto, en la base de datos se guardan objetos o instancias de estas 3 clases. Tanto para guardar información en la base de datos, como para recuperarla, se hará el mapeo información-objeto que se explicará más adelante.

- Usuario

  Usuario modela a aquellas personas (clientes actuales, potenciales, o no-clientes) que hayan accedido satisfactoriamente a la aplicación via email (no anónimo ni via móvil). Los objetos de esta clase se guardan en la colección de usuarios en la base de datos. Los atributos son los siguientes:

  - Uid: Identificador del usuario. Este identificador es el token de acceso generado por el servicio de autenticación de Firebase que se explicará más adelante. El valor no puede ser null.
  - Datos e información de acceso: username, email, lastaccess. Email y lastaccess no pueden ser null, mientras que username sí.
  - Datos personales: name, surname, gender, country, city, state, maritalstatus, age, phone, address, photoURL, favads. Favads es un mapa con los contenidos favoritos del usuario (zona – url). Los valores de todos estos atributos pueden ser null.

    El usuario podrá modificar todos sus datos personales y de acceso (menos la fecha de último acceso), además de eliminar su cuenta.

❖ RetailStore

RetailStore modela a los establecimientos retail que implementen este sistema. Los objetos de esta clase se guardan en la colección de retailstores en la base de datos. Los atributos son: nombre del establecimiento y ubicación geográfica (en coordenadas latitud, longitud).

❖ BlueAd

BlueAd modela el contenido o información a mostrar en un establecimiento según la ubicación determinada del usuario. Los objetos de esta clase se guardan como subcolección dentro de un documento retailstore en la base de datos. Los atributos son los siguientes:

- Uid: Identificador del contenido. Corresponde a la Bluetooth MAC de una baliza ubicada dentro del espacio de un establecimiento. Se entrará en detalle en este identificador posteriormente.
- Fecharegistro: Fecha en la que un establecimiento, con rol de administrador, "crea" el contenido para poder ser mostrado al usuario.
- Expiration: Fecha de expiración del contenido, el cual no solo es exclusivo en cuanto a ubicación sino limitado en el tiempo (como una oferta)
- Zona: Representa una sección temática dentro del espacio de un establecimiento (deportes, zapatería). Como se ha explicado, este sistema se ha pensado para tiendas por secciones o departamentales.
- Url: URL de una página web perteneciente a un dominio de usuario creado mediante Firebase Hosting. Esta página web es el contenido audiovisual completo (oferta, anuncio) que se muestra al usuario dentro de la misma aplicación.
- Image: URL de una imagen almacenada en el servicio Cloud Storage, asociada a un BlueAd determinado. Se utiliza para mostrar una previsualización de la oferta o anuncio al usuario.

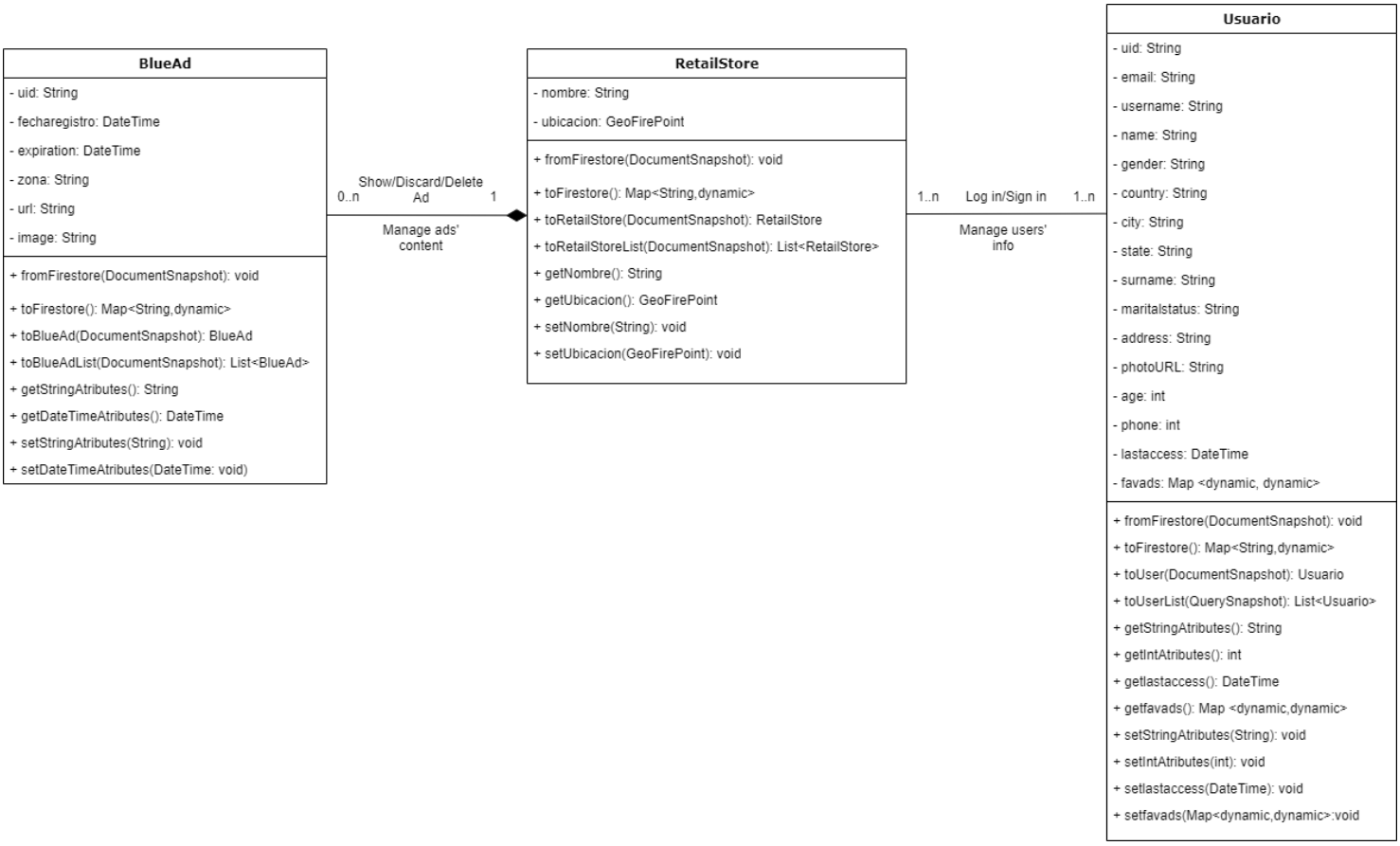


Ilustración 35 - Diagrama de Clases

Como se puede observar en la Ilustración 35, existen relaciones entre estas 3 clases. Un usuario o usuarios (hasta n) pueden "estar" en un establecimiento o establecimientos (hasta n), sin necesidad de ser usuarios exclusivos. Todos los establecimientos interesados en utilizar este sistema formarán una red retail. Cualquier usuario que accede a la app puede usarla en cualquiera de los establecimientos dentro de esta red para disfrutar de contenido exclusivo. Esta perspectiva sirve para tener una visión global de cuántos usuarios utilizan la app y cuantos establecimientos forman parte de esta red.

Un establecimiento sí posee o contiene contenido exclusivo (de 0 a n), que son los objetos BlueAds, los cuales no se encuentran en el resto de las tiendas. Si un objeto BlueAd "existe", está asociado a una ubicación específica dentro de un único establecimiento. Para que un

usuario acceda a este, tendrá que estar en dicha ubicación. El establecimiento en cuestión es el administrador de los contenidos a mostrar a los usuarios, así como del número de balizas y su ubicación en el espacio disponible. Estas relaciones se muestran de forma muy visual (mediante colecciones, subcolecciones y documentos) en la estructura de base de datos.

Por otro lado, estas clases además del constructor para inicializar un objeto, getters y setters, tienen métodos específicos de mapeo información-objeto tanto para guardar información a la base de datos cloud como para obenerla. También poseen métodos para mapear listas de información a listas de objetos.

5.2.2 APLICACIÓN MÓVIL

En este apartado se explica el diseño de la estructura y lógica de la aplicación móvil mediante múltiples diagramas (casos de uso, flujos de usuario y arquitectura/diseño de navegación). El interfaz de usuario resultante se muestra en el Capítulo 6.

Esta aplicación se ha creado para el rol usuario (cliente), si bien se han desarrollado muchas funcionalidades de rol administrador (simulando las acciones de un establecimiento) para gestionar los servicios cloud del back-end.

Es muy importante destacar que, la lógica de la app distingue al usuario en dos tipos: usuario que accede vía email, y usuario anónimo o que accede vía móvil. Los casos de uso para cada uno son distintos. La explicación a esta distinción se explica en el apartado siguiente.

- Casos de uso

  Mediante los casos de uso, se modelan las acciones de un actor en un sistema. En este caso, se distingue entre aplicación y usuario, ya que se comportan de distinta forma, si bien la lógica de la aplicación es la encargada de gestionar los casos de uso de un usuario determinado.

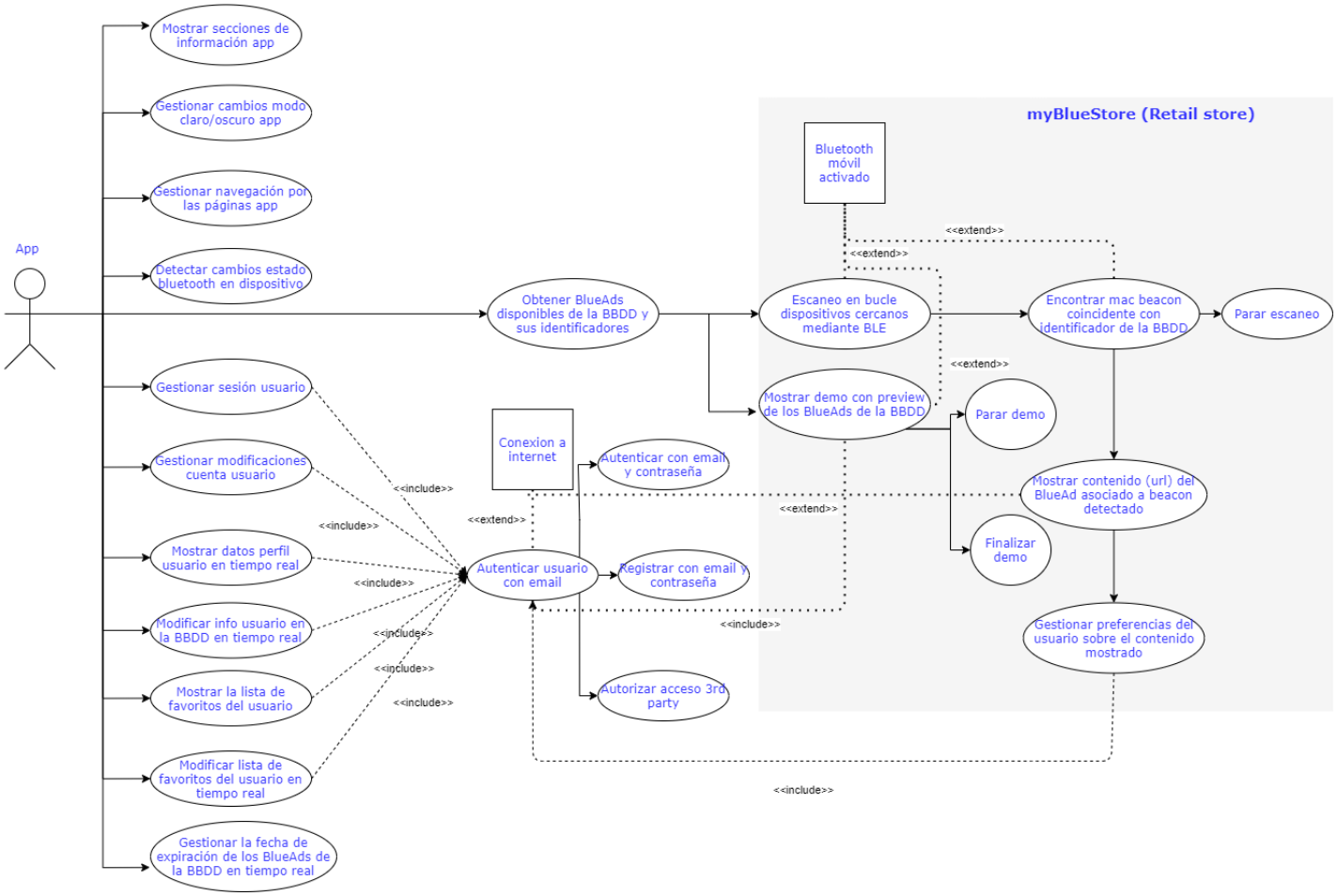


Ilustración 36 - Diagrama Casos de uso App

En el caso de la aplicación como actor, se modelan tanto acciones puramente de interfaz de usuario, como mostrar opciones del menú lateral, las url de los BlueAds o cambiar el UI según el tema elegido por el usuario (claro/oscuro), como de lógica o capa de negocio, como detectar el estado del Bluetooth, el tipo de autenticación de usuario, gestionar la sesión de un usuario, la información de la base de datos o llevar a cabo el scanning.

Como se observa en el diagrama UML, algunas acciones se pueden realizar fuera del espacio de un establecimiento (myBlueStore), como gestionar el acceso del usuario

o mostrar información de su perfil, mientras que las relacionadas con el scanning y búsqueda de contenido solo pueden realizarse dentro de un establecimiento que implemente el sistema desarrollado. Asimismo, para comenzar a escanear y encontrar coincidencias entre identificadores mac necesariamente el Bluetooth debe estar activado. Previamente a empezar el scanning, la app guarda "en local" una lista con los BlueAds disponibles en la base de datos, la cual se modifica en tiempo real si el contenido de dicha BBDD cambia. La app también controla el estado Bluetooth en todo momento para habilitar la navegación a distintas secciones, como se explicará en el diagrama de navegación de usuario.

Además, determinadas acciones incluyen una previa autenticación de usuario via email (registro, acceso usuario-contraseña o con proveedores de autenticación). Dichas acciones se detallan en el diagrama de Usuario.
Por otro lado, es importante destacar que para acceder a la aplicación es necesaria la conexión a Internet, ya que no se ha configurado la persistencia en el servicio Cloud Firestore, además de que para autenticarse mediante el servicio de un proveedor se necesita estar online. También se requiere conexión para visualizar las URL.

En el caso del usuario como actor, se distinguen dos tipos, según el método de acceso a la aplicación utilizado: Usuario y Usuario anónimo o móvil. El primero es aquel que se registra o accede con email y contraseña, via email link o con algún proveedor de acceso (Google, Facebook o Twitter). El segundo es aquel que accede de forma anónima o mediante su teléfono móvil (tras recibir un código de verificación). La gestión validación de accesos se lleva a cabo por el servicio cloud Autenticación, que se explica en el próximo apartado.

Como se recoge en el siguiente diagrama UML, hay diferencias notables en los casos de uso de estos dos tipos de usuario, las cuales se pueden resumir en: perfil, preferencias y demo.

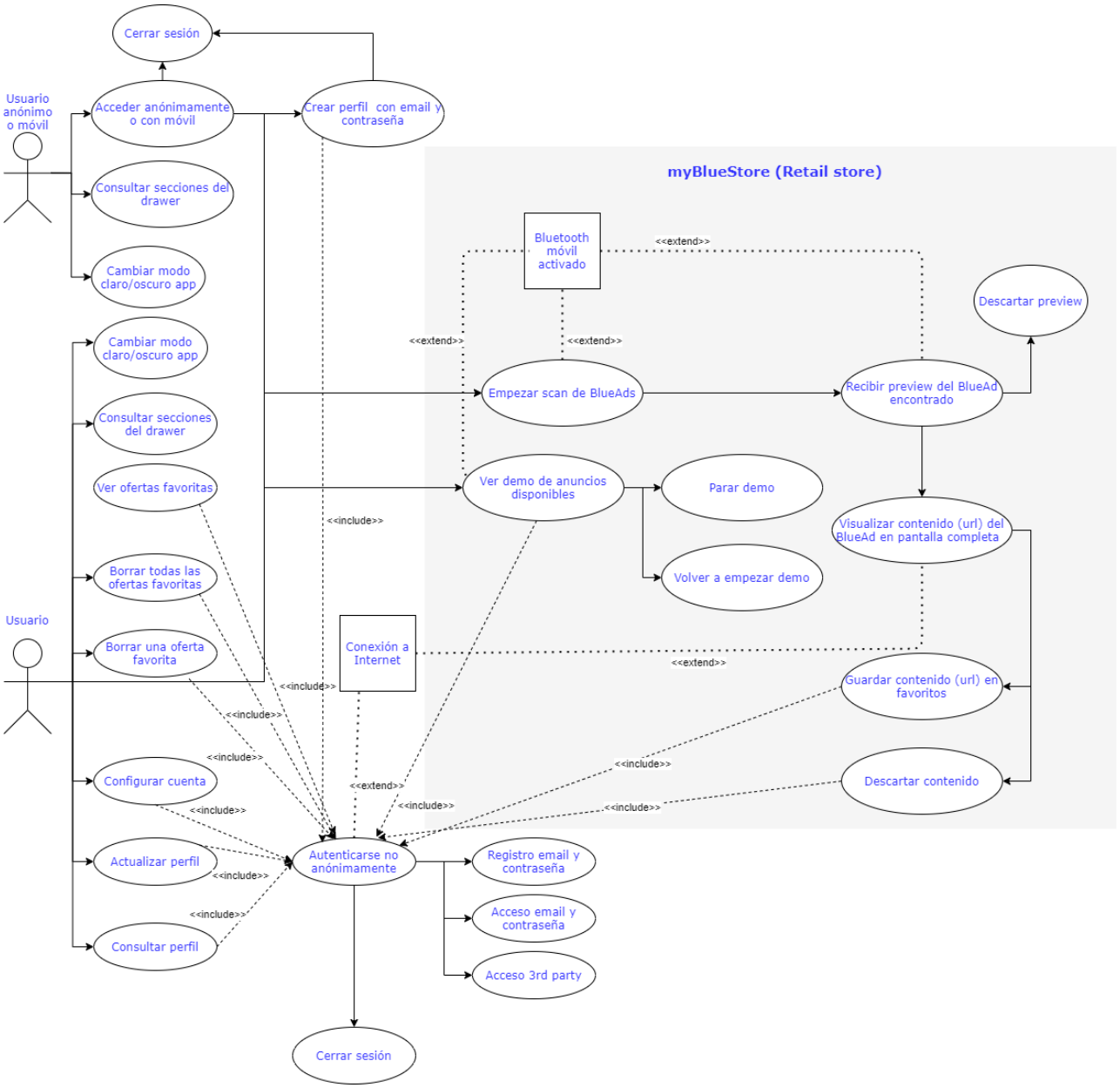


Ilustración 37 - Diagrama Casos de uso Usuario

El tipo Usuario, si tiene conexión a Internet, una vez accede (o se registra) a la aplicación con su cuenta de email, puede consultar su perfil de usuario y modificar sus datos personales y su foto en cualquier momento. También puede modificar su cuenta (cambiar email o contraseña, o borrar la cuenta de usuario).

Además, si el tipo Usuario se encuentra dentro de un establecimiento (myBlueStore), se le habilitan dos opciones, siempre y cuando tenga el Bluetooth activado: Demo y Scanning.

La opción Demo consiste en una versión simulada del proceso de scanning de balizas físicas. En esta, se reproduce el recorrido de un usuario dentro de un establecimiento, mostrándole cada 20 segundos una previsualización (en pantalla completa) de un BlueAd distinto, pero no el contenido exclusivo (url) que se muestra en el scanning. La demostración finaliza cuando ya se hayan mostrado todos los BlueAds de la base de datos. El usuario puede parar la demo en cualquier momento, o repetirla una vez finalizada.

En el caso de la opción Scanning ya explicada, cuando el tipo Usuario visualiza la URL del BlueAd encontrado, puede guardarla como favorita o descartarla. Además, puede consultar su lista de BlueAds favoritos, gestionarla (borrar uno o todos, visitar la página web de nuevo) y ver el tiempo que falta para que sus favoritos dejen de estar disponibles (tiempo que queda hasta la fecha de expiración)

Por otro lado, el tipo Usuario anónimo/móvil no tiene cuenta email, por lo que no puede consultar su perfil. Tampoco se le muestra la opción Demo con el Bluetooth activado, solamente la de Scanning. Al no tener perfil, no podrá guardar contenidos favoritos. Sin embargo, este usuario puede registrarse y crearse una cuenta nueva con toda la funcionalidad de un tipo Usuario.

- Flujo interacción usuario-app

Este diagrama de flujo representa de forma visual la ruta o los pasos que sigue este a lo largo de la app. Es muy útil para diseñar el UI y la lógica de forma que el usuario pueda ir avanzando a lo largo la ruta de la forma más accesible y sencilla posible.

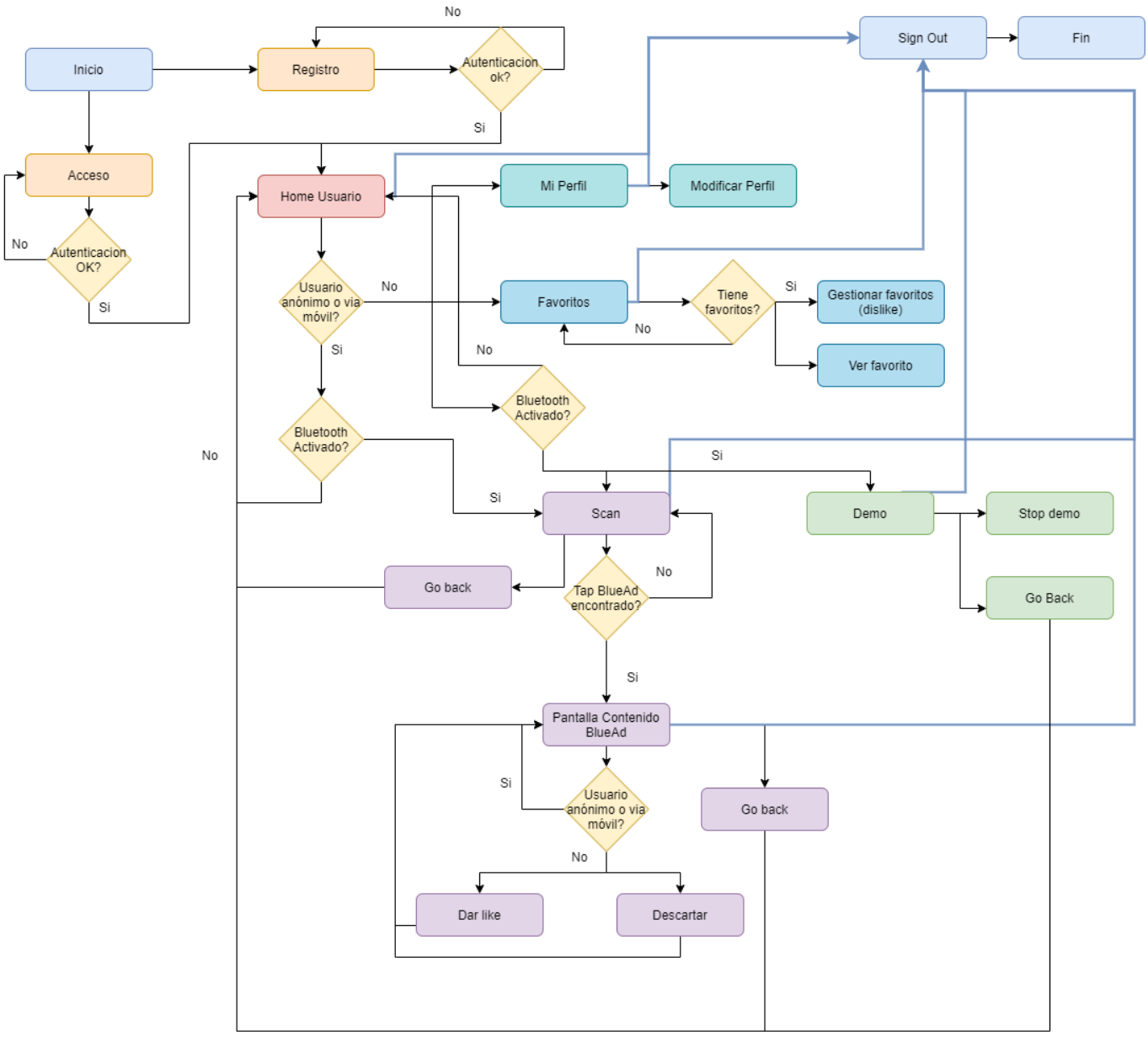


Ilustración 38 - Diagrama Flujo Usuario

Como se observa en el diagrama, la lógica de la aplicación controla el estado de autenticación, el tipo de usuario (tipo de acceso) y el estado del Bluetooth del dispositivo y otras interacciones del usuario para guiar al usuario por la ruta a seguir. Este diagrama es un buen resumen de todas las formas posibles de interacción entre usuario y aplicación. Varios ejemplos son los siguientes:

- Si la autenticación de un usuario es incorrecta, no podrá acceder a la página principal o home de la aplicación.

- Si el usuario no tiene el Bluetooth activado, no podrá acceder a la sección de Scan ni buscar contenido.

- Para guardar un favorito, un usuario que acceda satisfactoriamente con su email, debe tener el Bluetooth activado, navegar a la sección de Scan, esperar a que se encuentre un BlueAd, darle click a la previsualización y pulsar el botón de "me gusta" una vez se muestra la página web del BlueAd en pantalla completa.

- Si un usuario quiere ver el contenido de un favorito, debe acceder satisfactoriamente con su email y tener algún favorito en su lista.

- Un usuario que accede de forma anónima o con su móvil no puede guardar favoritos.

❖ Flujo de navegación app

El diseño de la navegación a lo largo de la app se ha dividido en varios bloques, como se muestra en la Ilustración 39. Estos bloques o tareas representan las pantallas del UI a desarrollar en un orden determinado y cómo se deben relacionar entre ellas dentro del proyecto Flutter. Cada pantalla es responsable de controlar determinados procesos o estados para permitir el avance del usuario al siguiente bloque. Por ejemplo, en la tarea Landing, la pantalla MainScreen es la que se encarga de gestionar si hay alguna sesión de usuario abierta. La implementación de cada bloque/tarea se explica posteriormente.

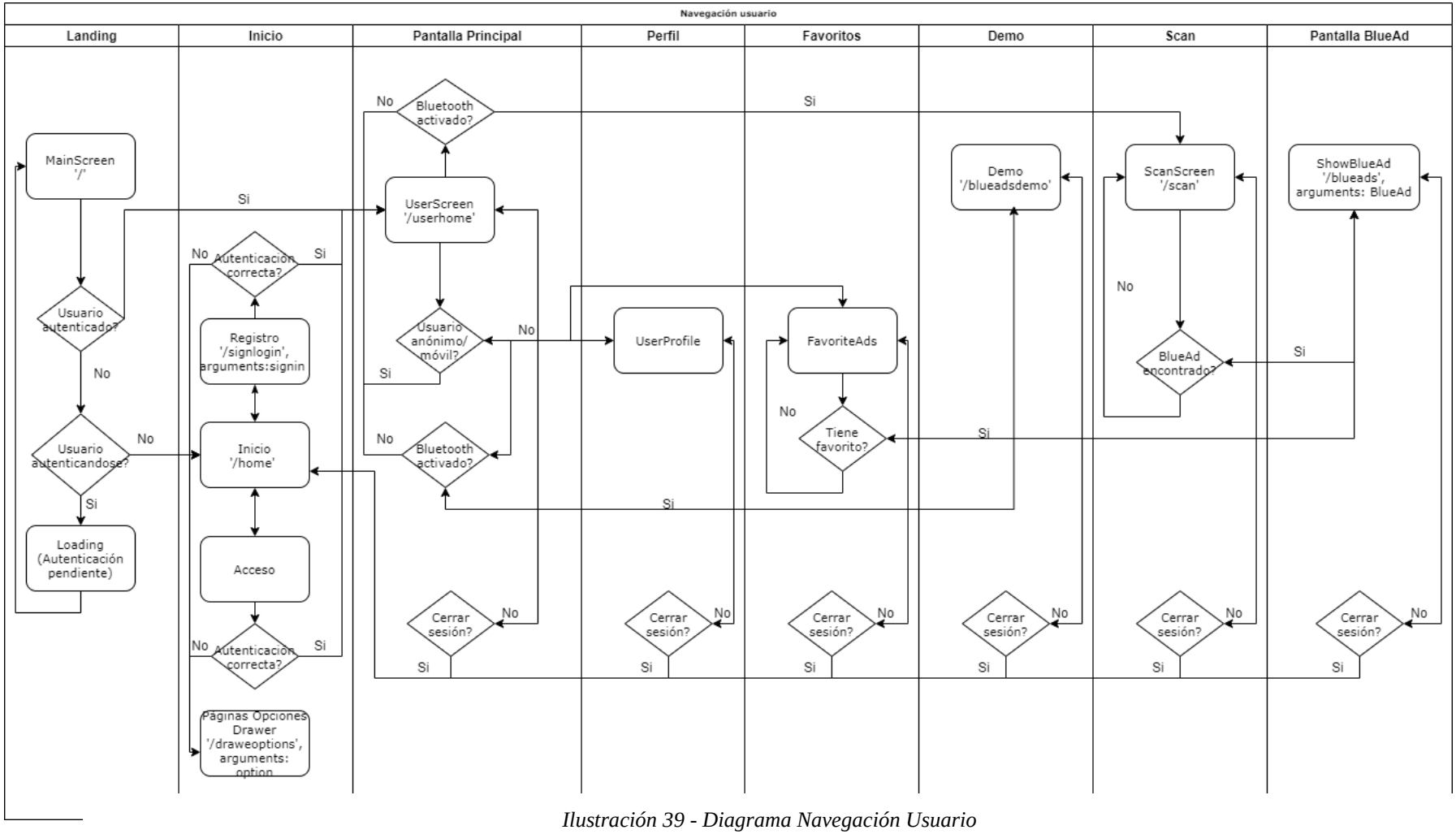


Ilustración 39 - Diagrama Navegación Usuario

5.2.3 SERVICIOS CLOUD

En este apartado se explicará el diseño de la estructura de los servicios de Firebase: Autenticación, Cloud Firestore, Cloud Storage, Firebase Hosting y Dynamic Links.

- Autenticación

  Este servicio de back-end gestiona el acceso seguro y privado (sin guardar contraseñas) de los usuarios a la aplicación, otorgándoles un token UID de acceso si la autenticación ha sido satisfactoria. Este token es inmutable mientras exista una cuenta de usuario, y solo se borra si se elimina una cuenta. Ofrece múltiples formas de acceso, de los cuales se han configurado las siguientes: correo/contraseña, vínculo o link por correo, teléfono móvil, Google, Twitter, Facebook y anónimamente.

  Si el usuario se autentica de forma anónima, se le crea una cuenta temporal o de sesión con un token UID. Una vez cierra sesión, la próxima vez que se autentique (de la forma que sea) se le creará otro token distinto al anterior. Por lo tanto, la información perteneciente a un usuario anónimo en una sesión determinada se perderá al finalizar la misma, a no ser que se vinculen nuevas credenciales (email y contraseña) al token de acceso antes de cerrar sesión.
  En este proyecto se ha optado por crear al usuario anónimo una nueva cuenta con un nuevo token de acceso, a partir de la cual pueda iniciar sesión, en vez de vincular credenciales.

  Por otro lado, para que un usuario se autentique via móvil y se genere un token UID, es necesario verificar previamente el número, al cual se le envía un código mediante SMS. Sin embargo, como ocurre con los usuarios anónimos, este token es temporal, por lo que no existe permanencia en la sesión. Se da el mismo tratamiento a este usuario que al anónimo.

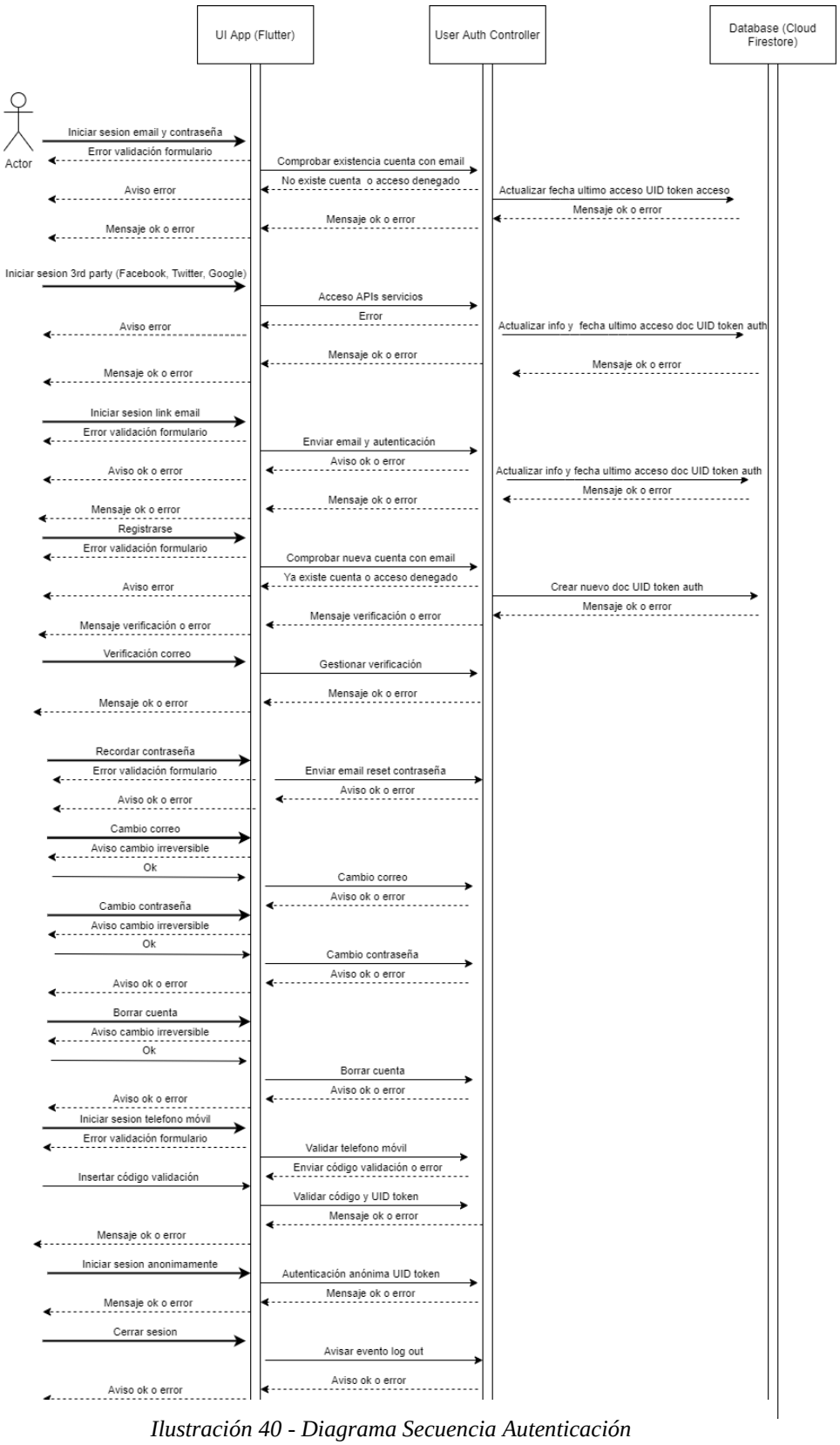


Ilustración 40 - Diagrama Secuencia Autenticación

En esta Ilustración 40 se representa el proceso de autenticación de un usuario. En este participan:

- El actor/usuario, el que decide cómo quiere acceder a la aplicación
- El interfaz de usuario (Flutter), que muestra las pantallas correspondientes a cada método de acceso, y se comunica con una capa controladora, de la que recibe el estado de autenticación y da feedback visual al usuario sobre el mismo y los posibles errores ocurridos en el proceso. Si la autenticación ha sido satisfactoria, dirige al usuario a la página principal.
- Capa controladora intermedia, donde se encuentra la lógica (métodos) para comunicarse con el servicio Authentication de Firebase y gestionar el estado de autenticación y los errores producidos, así como para avisar a la UI del resultado de autenticación.
- La base de datos Cloud Firestore, donde se actualizan la información relativa a los usuarios autenticados. Por los motivos expuestos, solo se guarda la información relativa a usuarios con cuenta email en la base de datos, como la última fecha de acceso o cambios en las credenciales, lo que resulta en dos tipos de usuario con casos de usos distintos, como ya se ha explicado.

Además, el servicio de Autenticación también se encarga de otros procesos como la verificación de dirección de correo, cambio de dirección de correo o contraseña, recordatorio de contraseña o eliminación de cuenta. Los cambios en credenciales o eliminación de cuenta son irreversibles, por lo que hay que avisar al usuario.

En este proyecto, se utilizan un Dyamic Links para redirigir al usuario desde su correo electrónico a la app en el caso de verificación de correo y acceso mediante link al correo.

❖ Cloud Firestore

Como ya se ha adelantado, esta base de datos no relacional y orientada objetos contiene dos colecciones principales en las que se guardan en forma de documentos únicos los objetos tipo Usuario y RetailStore. Dentro de cada documento de esta última, se encuentra una subcolección para los objetos tipo BlueAd, ya que un objeto RetailStore posee varios objetos BlueAds . Por tanto, el modelo y la relación entre clases ya explicada ha estructurado la jerarquía de la base de datos.

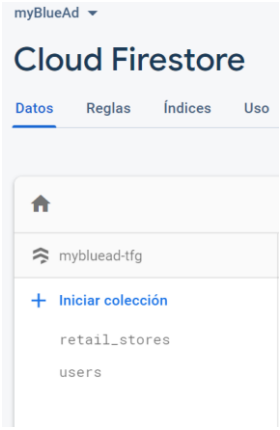


Ilustración 41 - Estructura raíz Cloud Firestore

Las rutas para acceder a los elementos de la jerarquía son las siguientes:

- mybluead-tfg: '/'. Base de datos, raíz principal.
- Colección retail_stores : '/retail_stores' . Colección de la raíz principal.
- Colección blueads: '/retail_stores/retail_stores_ID1/blueads': Subcolección dentro del primer documento de la colección retail_stores.
- Documento bluead: '/retail_stores/retail_stores_UID1/blueads/bluead_IDn' Dentro de la subcolección blueads, con el identificador correspondiente.
- Coleccion users: '/users'. Colección de la raíz principal.
- Documento user: '/users/user_IDn' . Dentro de la colección users, con el identificador correspondiente.

Como se puede observar por la ruta para acceder a la colección blueads, solo se ha creado un establecimiento (primer documento de la colección retail_stores). para mostrar el funcionamiento de la aplicación. Sin embargo, la estructura de la BBDD perfectamente escalable para en el futuro añadir nuevos establecimientos y poder tener una visión global desde la colección retail_stores.
Además, con esta estructura también se tiene una perspectiva general de todos los usuarios que utilizan la aplicación, en una colección aparte, ya que como se ha explicado, un usuario puede "estar" en cualquier establecimiento, no es exclusivo. En el futuro se podría integrar otra subcoleccion de usuarios por cada establecimiento.

En cuanto a los documentos de las 3 colecciones, estos corresponden a los objetos de las clases RetailStore, BlueAd y User. Cada documento/objeto tiene un identificador único dentro de una colección, que se puede crear aleatoriamente o con un valor predeterminado. Para cada documento, se puede crear una estructura en forma de mapa clave-valor distinta. Siguiendo el modelo, todos los documentos/objetos de una colección poseen una estructura de mapa con claves comunes, que son los atributos, y para cada uno, el valor en cuestión.
Un identificador de un documento no se puede cambiar, por cuestiones de seguridad de la BBDD. Para modificarlo, hay que sobreescribir el documento, es decir, borrarlo y crear otro nuevo con un nuevo ID.

Los documentos de la colección de usuarios se identifican con los tokens únicos generados por el servicio de autenticación (solo de los usuarios que acceden via email), que también son inmutables, y solo se borran si se borra una cuenta. Por este motivo, se utiliza este token como ID de documento, ya que solamente se elimina un usuario de la base de datos, si este elimina su cuenta. Se explica más en detalle en el apartado de implementación.
Por otro lado, el identificador de un documento de la subcolección blueads se genera automáticamente, cuando el administrador guarda cada BlueAd. Una vez guardados

todos los BlueAds, el administrador escanea las balizas y guarda sus Bluetooth MACs como valores de los atributos o campos "uid" de los documentos pertinentes.

No es una buena práctica utilizar las Bluetooth MAC, aunque en el caso de los SensorTags sean únicas, como identificadores, ya que las balizas pueden quedarse sin batería, o dejar de funcionar. Si ocurre esto, el identificador ya no es válido, y para no perder la información "valiosa" de un BlueAd, que es su URL (o el contenido a mostrar al usuario), habría que guardar la información del documento antes de borrarlo. Cuando hay muchos documentos, es un proceso tedioso.

Asimismo, cuando se haya alcanzado la fecha de expiración de un BlueAd, este deja de estar disponible para el usuario, es decir, ya no se muestra. Para esto, se cambia el valor del uid a null para no asociarlo a ninguna baliza física, pero no se borran los otros valores (url o zona) ya que, el contenido podría volver a estar disponible después de un tiempo.

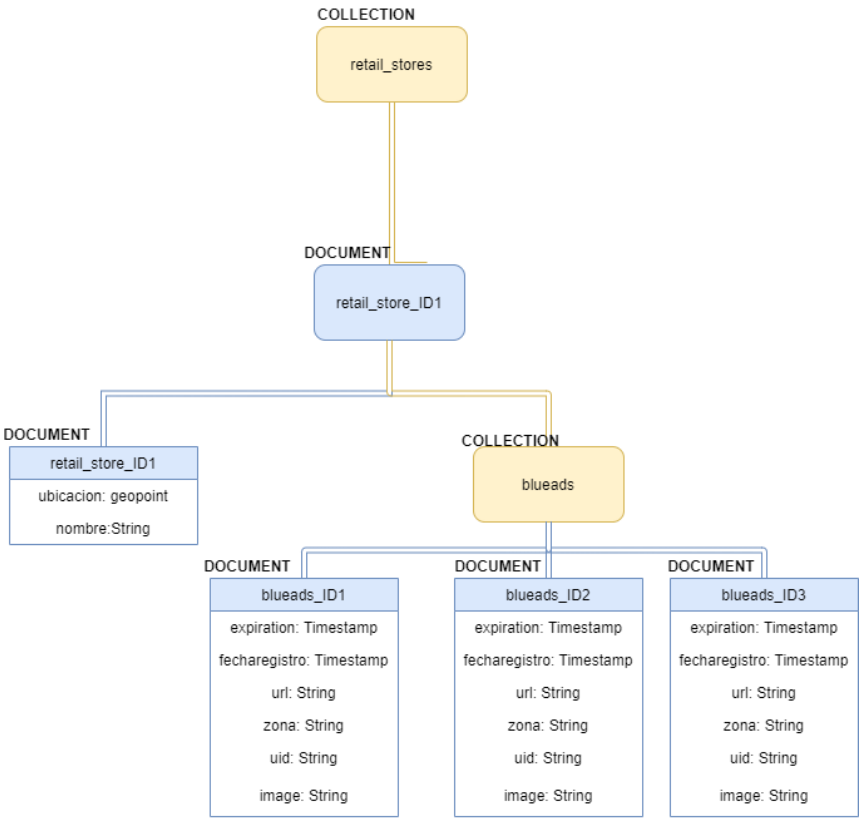


Ilustración 42 - Estructura BBDD

❖ Cloud Storage

Este servicio consiste en buckets o contenedores para almacenar cualquier tipo de objeto. En este proyecto, se utiliza el bucket por defecto, para almacenar imágenes correspondientes a los usuarios (foto perfil) y a los establecimientos (imagen para la previsualización de un BlueAd). Por tanto, se estructura en dos carpetas que coinciden con las dos colecciones:

Ilustración 43 - Bucket Cloud Storage

Actualmente, dentro de cada carpeta hay una subcarpeta, donde se guardan las imágenes. Un ejemplo de ruta de acceso a una de estas subcarpetas es la siguiente: gs://mybluead-tfg.appspot.com/retail_stores/ads_img. El administrador del sistema puede subir o descargar imágenes desde dicha ruta.

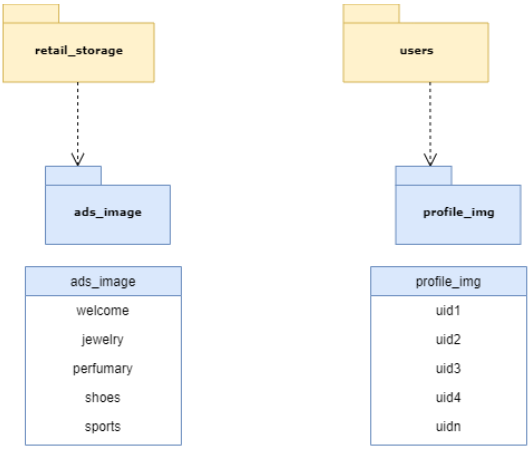


Ilustración 44 - Estructura Cloud Storage

❖ Firebase Hosting

Como se explica en la documentación oficial, se puede añadir un dominio personalizado en un proyecto y desplegarlo con este servicio. Para ello, Firebase Hosting busca registros DNS apropiados/libres mediante un lookup al proveedor DNS. Finalmente, despliega el contenido con un certificado SSL a través de un CDN global, la cual es una red de entrega de contenido rápida, próxima al usuario final.

En este proyecto se han creado dos dominios (además del creado por defecto) para el rol de administrador y el de usuario. El primero no se ha utilizado en el proyecto.

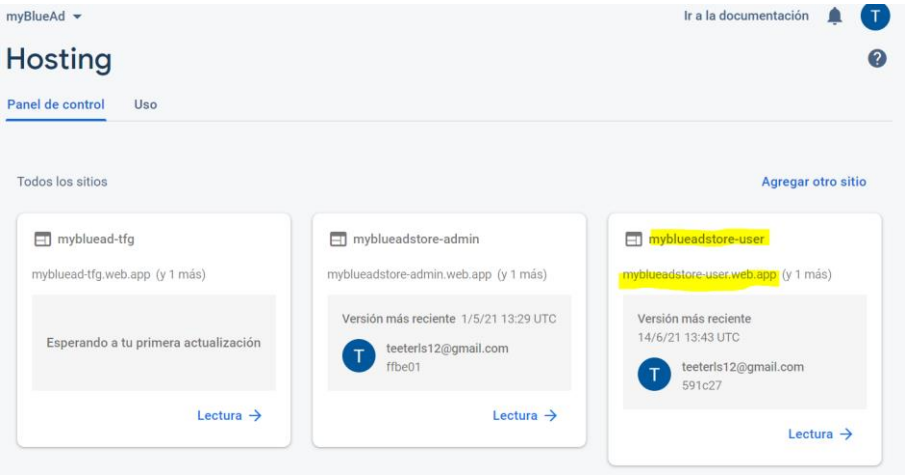


Ilustración 45 - Consola Firebase Hosting

A partir de este dominio de usuario, se utilizan Dynamic Links o rutas cortas para generar nuevas URL. La configuración e implementación se explica más adelante.

Como se recoge en el siguiente diagrama, una vez se ha alojado el contenido web (URLs de los BlueAds) a Internet con el dominio de usuario que provee Firebase Hosting, la aplicación móvil es la encargada de mostrar ese contenido al usuario tras el proceso de scanning. A partir de la URL a mostrar al usuario, si hay conexión a Internet, la app lleva a cabo el DNS lookup pertinente para obtener la IP, y mediante el componente WebView le muestra al usuario el contenido web sin salir de la misma. WebView por tanto sirve como "carcasa

web", que permite la navegación web embebida, dentro de la propia aplicación, sin necesidad de utilizar un buscador externo. Este componente se utiliza mucho en la programación nativa.

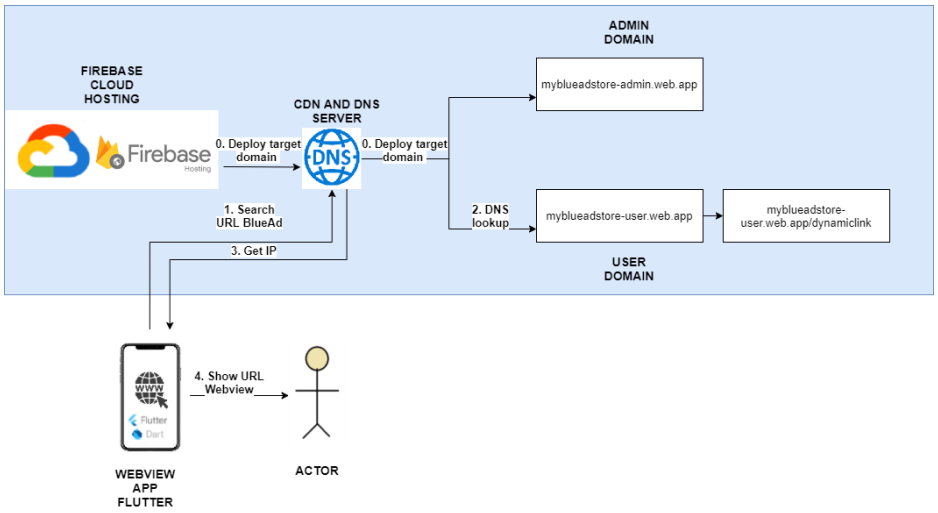


Ilustración 46 - Diagrama Firebase Hosting

5.3 IMPLEMENTACIÓN

En este apartado se explica con detalle cómo se ha estructurado el sistema y la interacción entre los 3 componentes. Asimismo, se resaltarán las partes más importantes del código.

En cuanto al proyecto de la app en Flutter, se estructura en varias carpetas y archivos, algunos creados por defecto o autogenerados, de los que se pueden destacar:

- android: carpeta con archivos de configuración de una aplicación Android.
- iOS: carpeta con archivos de configuración de una aplicación iOS.
- build: carpeta autogenerada donde se encuentra el código compilado de la app.
- test: carpeta donde se encuentran las herramientas para ejecutar tests de widgets.
- integration test: carpeta donde se ejecuta un test completo de la app.
- pubspec.yaml: archivo donde se incluyen las dependencias del proyecto (packages).

- assets: carpeta donde se guardan los archivos e imágenes necesarios para ejecutar la app.
- firebase: carpeta creada por Firebase CLI (interfaz de comandos) para integrar servicios y herramientas del backend en el proyecto a través de a consola. Se entrará en detalle más adelante.
- lib: carpeta donde se encuentra todo el código ejecutable de la app (se ejecuta el método main.dart). El código se ha organizado en otras 3 carpetas para formar una estructura o arquitectura de 3 capas: modelo, servicios y vista. Se considera por tanto que el proyecto Flutter sigue un patrón arquitectónico MVC (modelo-vista-controlador).
    - model: código relacionado con las clases del Modelo ya explicadas, y otras dos clases que se explican a continuación. Es la capa modelo.
    - services: métodos para acceder a los servicios de back-end, gestionar peticiones, etcétera. Es la capa lógica o de negocio.
    - view: código relacionado con el UI. Se divide a su vez en screens (pantallas completas) y en widgets (trozos de UI reutilizables). Es la capa de presentación.

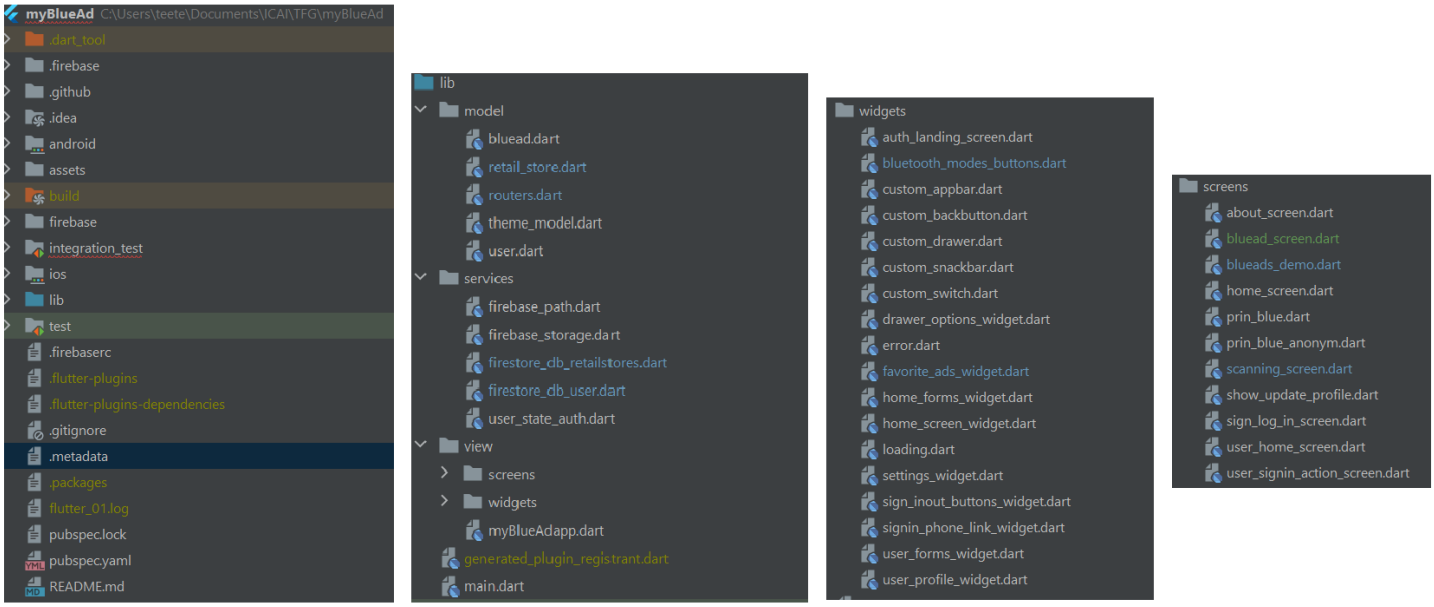


Ilustración 47 - Estructura Proyecto Flutter

5.3.1 PATRÓN DE DISEÑO PROVIDER

Ya se ha comentado anteriormente que un Widget, además de ser un elemento UI, posee un estado y lógica para gestionarlo. Por tanto, el código de interfaz y lógica se mezcla en un mismo widget. Incluso, como se explica en [49] , en grandes aplicaciones con mucho código, la lógica puede estar tan dispersa en varios Widgets que sea difícil de encontrar. En estos casos, para facilitar la tarea de desarrollo y que el código resultante sea más limpio y claro (técnicas clean code), es necesario utilizar algún patrón de diseño. En el caso de Flutter, hay varios (MVC, BLoC, Clean, Provider). Para estructurar este proyecto en carpetas y separar la lógica del UI lo más posible, se ha utilizado el MVC. Además, para manipular estados a lo largo del Widget Tree y detectar determinados tipos de cambios en el modelo se utiliza el patrón Provider, el cual es un paquete que se instala desde pub.dev y se añade como dependencia en el archivo pubspec.yaml proyecto Flutter.

Provider permite la centralización de los cambios de estado en una clase. Esto consiste en que solo una clase (Widget) es la encargada de manipular un estado, y proveer el valor de este al resto de Widgets que lo soliciten. En el momento en el que la clase con rol de Provider inicializa o modifica el estado, aquellos widgets que están escuchando podrán acceder a su valor actual desde cualquier posición del Widget Tree en la que estén, y actuar en consecuencia (normalmente repintarse). Por tanto, no es necesario ir pasando el estado de clases padres a hijas como un atributo. Provider avisa a los Widgets oyentes de los últimos cambios en el estado del cual es responsable.

Para acceder al valor de un estado, es necesario instanciar el Provider de la clase o modelo determinada, lo cual se hace con la siguiente sintaxis:

```
Provider.of<CounterService>(context, listen: false);
```

Codigo 1 - Ejemplo escucha Provider

Es necesario hacer explícito si el Widget que accede al valor del estado del Provider no va a repintarse mediante listen:false, ya que por defecto se repinta.

La clase Provider posee varios elementos, los más relevantes en este proyecto son:

- ChangeNotifier: clase desde la que se avisa a los listeners cambios en el estado del modelo mediante el método notifyListeners. Las clases que contienen el modelo deben heredar de esta.

- ChangeNotifierProvider: clase que "crea" el Provider con la instancia del modelo determinado mediante un método llamado create. Los widgets por debajo de esta clase (child) serán avisados de los cambios con notifyListeners.

- Consumer: clase para acceder al valor de un estado. Es una alternativa a la instancia del Provider ya explicada.

- StreamProvider: clase que "crea" un Provider que está escuchando permanentemente cambios asíncronos del propio modelo y los pasa a los Widgets descendientes.

- MultiProvider: clase que concentra a todos los widgets que van a escuchar cambios de algún Widget Provider. Normalmente, este elemento se incluye en el main como primer nivel de la aplicación, para que todos los Widgets que estén debajo de este en el árbol puedan ser oyentes de los Providers. Esta clase admite varios providers.

En este proyecto, se utilizan Providers para centralizar el estado de 3 clases del modelo: UserState, ThemeModel y List<BlueAd> (Lista de BlueAds). Se utiliza un MultiProvider en el main (nivel más alto, runApp) para que cualquier Widget pueda escuchar a lo largo de la app cambios en alguno de estos 3 Providers.

Las clases UserState y ThemeModel se explican a continuación, ya que los cambios en el modelo surgen de la interacción usuario-app, mientras que los cambios asíncronos en el modelo BlueAd son propios de la interacción app-base de datos.

```
1. void main() {
2.   //método para asegurar la inicialización de los Widgets
```

```
3.   WidgetsFlutterBinding.ensureInitialized();
4.   runApp(
5.
6.    //multiprovider
7.    MultiProvider(
8.     providers: [
9.         //escucha cambios documentos subcoleccion blueads
10.         StreamProvider<List<BlueAd>>(
11.             create: (_) => db.getBlueAds(), initialData: []
12.                  ),
13.         //escucha cambios tema claro/oscuro
14.         ChangeNotifierProvider <ThemeModel> (
15.             create: (_) => ThemeModel()
16.             ),
17.         ChangeNotifierProvider <UserState> (
18.           //singleton. Escucha cambios estado usuario
19.             create: (_) => UserState.instance(),
20.             ),
21.             ],
22.     child: InitializeApp(),
23.     ),
24.   );
25.   }
```

Codigo 2 - Main y Multiprovider

En este Codigo 2 se muestra como se inicializan múltiples Providers para notificar de cambios a los Widgets hijos (InitializeApp). El StreamProvider notificará de forma asíncrona los cambios en la base de datos, el Provider de ThemeModel notificará el cambio en el modo claro/oscuro de la UI, y el Provider de UserState notificará cambios en el estado de autenticación del Usuario.

5.3.1.1 Dart programación asíncrona

Como ya se adelantado, el lenguaje Dart soporta programación asíncrona. En este tipo de programación, Dart utiliza objetos como Future, que es un tipo de objeto cuyo valor (del tipo que sea) se espera en el futuro, y expresiones como async, para indicar que una función es asíncrona y devuelve inmediatamente un Future, o await, que fuerza la espera (pone en pausa el threat de ejecución) para obtener un resultado completado.

Cuando es necesario manipular una secuencia de eventos asíncronos, se utiliza la clase Stream. Esta funciona como un Iterable asíncrono, está escuchando en bucle una “fuente”

de datos de cualquier tipo que no sabe cuando terminará. Un Stream se compone de Snapshots (o AsyncSnapshots), que son capturas del último contenido actualizado de un Stream. Para devolver los valores de un Stream de utiliza la variable yield, que no se sale de dicho bucle, a diferencia de return. Además, las funciones que devuelven Streams, utilizan la expresión async* para indicar que se esperan "más de un Future".

La clase Stream tiene gran importancia en este proyecto, porque en Flutter se utiliza de forma muy frecuente un Widget llamado StreamBuilder. Este constructor de Streams, crea una UI para el flujo de eventos asíncronos cambiantes. Se construye basada en la última actualización del Stream (snapshot). Un ejemplo es el siguiente:

```
1. StreamBuilder<Usuario>(
2.         stream: db.getUserProfile(userstate.user.uid),
3.         builder: (context, AsyncSnapshot<Usuario> snapshot) {
4.           //si tengo un error se muestra en el widget aparte
5.               if (snapshot.hasError)
6.         {
7.             return ErrorContainer(snapshot.error.toString());
8.               }
9.         //todavía no hay datos
10.              if (!snapshot.hasData)
11.        {
12.                return Center(child: Loading());
13.         }
14.              //hay datos del perfil del usuario identificado
15.              return ShowUpdateProfile(snapshot.data);
16.   })
```

Codigo 3 - Ejemplo StreamBuilder Usuario

En el StreamBuilder del Codigo 3, recibe un stream o fuente asíncrona de datos pertenecientes al modelo Usuario. Con el builder, se puede controlar si la última actualización (snapshot) de la fuente asíncrona esta vacía o tiene un error, y mostrar una UI determinada. Este stream en cuestión es un método que recibe datos de un usuario desde Cloud Firestore, actualizados en tiempo real.

También, como se ha visto, se puede proveer de Streams (StreamProvider) a otros Widgets.

5.3.2 APLICACIÓN MÓVIL – USUARIO

La primera interacción app-usuario tiene lugar cuando este abre la aplicación. Como se recoge en la Ilustración 39, hay dos aspectos en el acceso de un usuario que la aplicación debe gestionar, además de comunicarse con el servicio de autenticación cloud:

El primero es el estado de autenticación, para determinar si hay alguna sesión abierta previamente y redirigir al usuario. El segundo es el tipo de acceso, que es importante para distinguir los distintos casos de uso ya comentados. Se crea un modelo UserState y un Provider que notifique los cambios en dicho modelo para controlar ambas cuestiones.

Dentro de este modelo, se enumeran 4 tipos de estados (Status) posibles: No inicializado, Autenticando, Autenticado, No autenticado. Como se puede observar en el código, esta clase modelo hereda de ChangeNotifier para notificar cambios a otros Widgets mediante el método_onAuthStateChanged (dentro del cual se encuentra notifyListeners). Además, se utiliza el patrón singleton para crear una única instancia de Firebase Auth y notificarla sobre los cambios del estado del usuario.

```
1. class UserState with ChangeNotifier {
2.   FirebaseAuth _auth;
3.   User _user;
4.   String _verificationId;
5.
6.   //estado inicial
7.   Status _status = Status.Uninitialized;
8.
9.   //listen for changes in auth status -> SINGLETON ONE AND ONLY
   FIREBASEAUTH INSTANCE
10.     UserState.instance() : _auth = FirebaseAuth.instance {
11.       _auth.authStateChanges().listen(_onAuthStateChanged);
12.     }
13.
14.     //getters
15.     Status get status => _status;
16.
17.     User get user => _user;
```

Codigo 4 - UserState Class

```
1. //metodo avisar listeners cambios FirebaseAuth
```

```
2.   Future<void> _onAuthStateChanged(User firebaseUser) async {
3.     if (firebaseUser == null) {
4.       //wrong
5.       _status = Status.Unauthenticated;
6.     } else {
7.       _user = firebaseUser;
8.       //ok
9.       _status = Status.Authenticated;
10.       }
11.       notifyListeners();
12.     }
13.   }
```

Codigo 5 - Método onAuthStateChanged

Inicialmente, el usuario se encuentra no inicializado. Si se autentica satisfactoriamente al usuario con algún método de acceso, el estado pasará a ser autenticado, se creará una instancia de la clase User (Firebase Auth, objeto que representa el usuario autenticado actual) con el email de acceso y el nombre de usuario si procede y se avisará a los Widgets. Si el proceso no ha sido satisfactorio, el estado será no autenticado. Un ejemplo de un método de autenticación es el siguiente:

```
1. //3rd party
2.   Future <String> signInWithGoogle() async {
3.     //authenticating
4.     _status = Status.Authenticating;
5.     notifyListeners();
6.     try {
7.       //ok, se crea user
8.       await _auth.signInWithCredential(await
   _credentialGoogle()).
9.       then((currentUser) async
10.         {
11.           User us= currentUser.user;
12.           Usuario _usuario = Usuario(us.uid, email: us.email);
13.          String e= await db.signUser(us.uid, _usuario);
14.           return e;
15.         });
16.
17.       } on FirebaseAuthException catch (e) {
18.         _status = Status.Unauthenticated;
19.         notifyListeners();
20.         if (e.code == 'account-exists-with-different-
   credential')
21.         {
22.           //devuelve string con error o no
```

```
23.           (_handledifferentCredentials (e.email, e.credential))
   .then((value) => value);
24.         }
25.         return e.code;
26.       }
```

Codigo 6 - Ejemplo GoogleSignIn método

Gracias a los getters, otros Widgets a través del Provider pueden obtener el estado, la instancia de User asociada al UserState, y su email si tiene, para distinguir entre los dos tipos de acceso ya comentados. Un ejemplo para distinguir entre tipos de usuario con el Provider es el siguiente:

```
1. final userstate =Provider.of<UserState>(context,listen: false);

...

2.  if (userstate.user.email!=null) {
3.      return  Stack(
4.          children: [
5.               Padding(
6.                padding: const EdgeInsets.only(left: 30.0),
7.                child: Align(
8.                     alignment: Alignment.bottomLeft,
9.                     child: DemoButton(true)),),
10.              Align(
11.               alignment: Alignment.bottomRight,
12.               child: ScanButton(true)),
13.                     ],
14.                     );
15.                              }
16.   else
17.     return ScanButton(true);
```

Codigo 7 - UserHome Provider UserState

Este Codigo 7 se encuentra en la página UserHome, donde una vez se obtiene la instancia de User (Firebase Auth), se comprueba si este tiene email asociado, para mostrarle los botones tanto de Demo como de Scan. Si el valor del atributo email es nulo, significa que el usuario ha accedido via móvil o anónimamente, por lo que, en concordancia con el diagrama de casos de uso, solo se le muestra el botón de Scan.

La navegación del usuario es gestionada por una clase llamada Routers, que forma parte del modelo. En esta, se generan y concentran todas las rutas nombradas (Router Widget). A partir de cada ruta, se construye un Widget o pantalla (builder MaterialPageRoute), y se puede navegar a través de ellas mediante el elemento Navigator.

```
1. class Routers {
2.   static Route<dynamic> generateRoute(RouteSettings settings) {
3.     switch (settings.name) {
4.       case '/':
5.         //pagina main que es la que redirige segun el estado
6.         return MaterialPageRoute(builder: (_) => MainScreen());
7.
8.       case '/home':
9.       //pagina inicio app
10.          return MaterialPageRoute(builder: (_) => HomeScreen()
   );
11.
12.      case '/signlogin':
13.          //pagina registro o acceso
14.          var options = settings.arguments as String;
15.          return MaterialPageRoute(builder: (_) =>
16.                                    SignLogInScreen(options));
17.
18.      case '/draweroptions':
19.        //pagina opcion drawer lateral
20.          var options = settings.arguments as String;
21.          return MaterialPageRoute(builder: (_) =>
22.                                  DrawerOptionsWidget(options));
23.
24.        case '/userhome':
25.        //pagina principal usuario
26.          return MaterialPageRoute(builder: (_) =>
27.                                         UserHomeScreen());
28.
29.        case '/credentials':
30.         //gestion conflicto credenciales auth 3rd party.
31.          var credentials = settings.arguments as List;
32.          return MaterialPageRoute(builder: (_) =>
33.                                   PwdCredentials(credentials));
34.
35.        case '/signinoptions':
36.          //pagina acceso con link o telefono
37.          var options = settings.arguments as String;
38.          return MaterialPageRoute(builder: (_) =>
39.                                    SignInPhoneLink(options));
40.
41.        case '/useraction':
```

```
42.           //pagina cambio cuenta
43.           var options = settings.arguments as String;
44.           return MaterialPageRoute(builder: (_) =>
45.                                       UserActionScreen(options));
46.
47.         case '/scan':
48.           //pagina scanning
49.           return MaterialPageRoute(builder: (_)=>ScanScreen());
50.
51.         case '/blueads':
52.           //pagina visualizacion BlueAd encontrado
53.           var bluead=settings.arguments as BlueAd;
54.           return MaterialPageRoute(builder: (_) =>
55.                                           ShowBlueAd(bluead));
56.
57.         case '/blueadsdemo':
58.           //pagina demo
59.           return MaterialPageRoute(builder: (_) =>
60.                                            BlueAdsDemo());
61.         default:
62.           //error
63.           return MaterialPageRoute(builder: (_) {
64.             return Error('No route founded for
65.                                   ${settings.name}');
66.           });
67.       }
68.     }
69.   }
```

Codigo 8 - Clase Routers

La ruta inicial es '/', que corresponde al Widget (MainScreen) que redirige al usuario según su estado de autenticación. Como se puede observar en algunos casos, las rutas admiten argumentos para pasarlos a los Widgets o pantallas "construidas" durante la navegación. Lo más habitual es que el usuario navegue entre pantallas al pulsar determinados botones o ventanas emergentes.

```
1. class MainScreen extends StatelessWidget {
2.   @override
3.   Widget build(BuildContext context) {
4.     return ChangeNotifierProvider(
5.       create: (_) => UserState.instance(),
6.       child: Consumer <UserState>(
7.         builder: (context, user, _) {
8.           switch (user.status) {
9.             //primera vez
```

```
10.                 case Status.Uninitialized:
11.                   return HomeScreen();
12.                   //no se ha autenticado
13.                 case Status.Unauthenticated:
14.                   return HomeScreen();
15.                 //log in
16.                 case Status.Authenticating:
17.                   return Loading();
18.                 case Status.Authenticated:
19.                   {
20.                     //ok entra directamente en su perfil. esto es
   si hay alguna sesion abierta
21.                     return UserHomeScreen();
22.                   }
23.               }
24.             },
25.           ),
26.       );
27.     }
28.   }
```

Codigo 9 - MainScreen Class

Esta ventana accede al estado del Usuario mediante los elementos ChangeNotifierProvider y Consumer, que son equivalentes a la instancia de Provider. Según dicho estado, dirige al usuario a una pantalla u otra. Con esta lógica, se garantiza que la navegación del usuario coincida con la mostrada en la Ilustración 39.

El usuario también puede modificar el modo claro/oscuro de la aplicación al presionar un switch en la barra superior, que cambia el modo con la función toggleMode(). Para poder repintar todas las pantallas con el modo, se crea una clase ThemeModel que hereda de ChangeNotifier, que gestiona el tema del interfaz global (MaterialApp) y mediante un Provider avisa al resto de Widgets. Los widgets pueden acceder al tema con el getter.

```
1. class ThemeModel with ChangeNotifier {
2.
3.   ThemeMode _mode;
4.
5.   //constructor
6.   ThemeModel({ThemeMode
   mode = ThemeMode.light}) : _mode = mode;
7.
8.   //getter
9.   ThemeMode get mode => _mode;
```

```
10.     //flutter avisa a los listeners de que el modelo a cambiado
   de un tema a otro
11.     toggleMode() async{
12.       //cambio de thememode de uno a otro
13.       //patron observer
14.       _mode = _mode == ThemeMode.light ? ThemeMode.dark : Theme
   Mode.light;
15.       notifyListeners();
16.     }
17.   }
```

Codigo 10 - ThemeModel Class

El tema del UI global se construye a nivel de MaterialApp, con la propiedad theme. Se construye la misma mediante un Consumer que escuche los cambios del ThemeModel. En este nivel, también se indica cómo se generan las rutas (clase Routers).

```
1. class myBlueAdApp extends StatelessWidget {
2.
3.   final String _title = "myBlueAd";
4.   bool isswitch= false;
5.   @override
6.   Widget build(BuildContext context) {
7.     return ChangeNotifierProvider<ThemeModel>(
8.       create: (_) => ThemeModel(),
9.       child: Consumer<ThemeModel>(
10.           builder:(context, theme ,_) =>MaterialApp(
11.             debugShowCheckedModeBanner: false,
12.             title:_title,
13.             theme: ThemeData(
14.                 primaryColor: Colors.pink[500],
15.                 visualDensity: VisualDensity.adaptivePlatformDe
   nsity
16.             ), // Provide light theme.
17.             darkTheme: ThemeData.dark(), // Provide dark theme.
18.             themeMode:  theme.mode,
19.             initialRoute: '/',
20.             //generador de rutas
21.             onGenerateRoute: Routers.generateRoute,
22.
23.           ),
24.         ),
25.       );
26.     }
```

```
27.   }
```

Codigo 11 - myBlueAdApp Class

Asimismo, desde cualquier widget puede consultarse el modo claro/oscuro de la UI. En el Codigo 12 se decide el color de fondo de un botón según el valor de dicho modo, pero es necesario repintar el botón entero (es decir, calcular sus layouts, solo cambia el color):

```
backgroundColor: Provider.of<ThemeModel>(context, listen: false).
mode==ThemeMode.dark ? Colors.white: Colors.white54
```

Codigo 12 - Ejemplo Provider ThemeModel

Otras acciones que surgen de la interacción app-usuario, como modificar los datos del perfil o la lista de favoritos, o cargar los BlueAds de la base de datos son gestionadas por la capa de negocio/lógica, ya que también intervienen los servicios del back-end, y se explican a continuación.

5.3.3 APLICACIÓN MÓVIL – SERVICIOS CLOUD

La configuración del proyecto Flutter en Firebase se explica en el Anexo II. Cuando se ejecuta la app (nivel main), es necesario inicializar de forma asíncrona los servicios de Firebase en el proyecto. Como se muestra en Codigo 13 Si el proceso de inicialización ha sido correcto, se construye la MaterialApp, la cual asienta las bases de la UI de la app.

```
1. //widget que inicializa FlutterFire de forma async para la
   conexion con Firebase
2. class InitializeApp extends StatefulWidget {
3.   @override
4.   _InitializeAppState createState() => _InitializeAppState();
5. }
6.
7. class _InitializeAppState extends State<InitializeApp> {
8.   // Set default `_initialized` and `_error` state to false
9.   bool _initialized = false;
10.    bool _error = false;
11.    String _msg;
12.
13.    // Define an async function to initialize FlutterFire
```

```
14.     void initializeFlutterFire() async {
15.       try {
16.         // Wait for Firebase to initialize and set initialized
    state to true
17.         await Firebase.initializeApp();
18.         setState(() {
19.           _initialized = true;
20.         });
21.       } catch (e) {
22.         // Set _error state to true if Firebase initialization
    fails
23.         setState(() {
24.           _error = true;
25.           //error trace
26.           _msg=e.toString();
27.         });
28.       }
29.     }
30.
31.     //initial state
32.     @override
33.     void initState() {
34.       //async function? -> error de conexion o ok
35.       initializeFlutterFire();
36.       super.initState();
37.     }
38.
39.     @override
40.     Widget build(BuildContext context) {
41.       // Show error message if initialization failed
42.       if (_error) {
43.         //recibe el mensaje de error
44.         return Error(_msg);
45.       }
46.
47.       // Show a loader until FlutterFire is initialized
48.       if (!_initialized) {
49.         return Loading();
50.       }
51.
52.       //ok initialized -> App
53.       return myBlueAdApp();
54.     }
```

Codigo 13 - InitializeApp Class

Es importante destacar que todos los métodos necesarios para la interacción app-servicios backend se desarrolla en Dart, en la capa de negocio, ya que es más fácil e inmediato, además

de que existe una documentación oficial para este propósito (FlutterFire). Otra opción habría sido agregar el SDK de Firebase Admin en un servidor local, creado con tecnologías como Node.js, Python, Java, etcétera.

Para desarrollar esta lógica, se han importado como dependencias en el pubspec.yaml los plugins FlutterFire, (firebase_auth, firebase_storage, cloud_firestore, firebase_dynamic_links), así como el Firebase Core, que permite la conexión del proyecto a múltiples aplicaciones.

- Autenticación

  El primer paso es habilitar en la consola de Firebase los servicios de acceso correspondientes (email, link, móvil, anónimo y proveedores 3rd party). En el caso de los proveedores Facebook y Twitter, se requiere crear un proyecto en cada plataforma con el perfil de desarrollador y vincular unos tokens de autenticación al proyecto de Firebase. Además, en el caso de acceso por móvil o Google, se requiere el SHA.

  Se crean métodos con Dart para cada tipo de acceso habilitado, cuya lógica se encuentra en la carpeta services. Se gestionan además los conflictos entre credenciales que puedan surgir, por ejemplo, cuando un usuario accede tanto via email como con un proveedor.

  Con la instancia de Firebase Auth, se puede acceder a la instancia de la clase User, correspondiente al usuario actual autenticado. De este objeto se pueden obtener los valores de atributos como: UID (token autenticación),email, fotoURL, número de teléfono, o nombre de usuario, los cuales según el tipo de acceso que se haya utilizado serán null o no. Por ejemplo, si un usuario se ha autenticado con su móvil, este atributo tendrá un valor, pero el email será nulo. Una vez autenticado el usuario, también se pueden actualizar dichos valores con determinados métodos, o “linkear” credenciales a otra cuenta.

Otra forma de obtener la instancia de User es mediante el getter de UserState (a través del Provider), como se ha explicado.

❖ Cloud Firestore

Los métodos en Dart para guardar u obtener datos de las colecciones users y retail_stores se encuentran en la capa de negocio. Se gestionan las rutas de acceso a las colecciones o documentos de forma estática mediante la clase FirestorePath.

```
1. class FirestorePath {
2.    //all users
3.     static String userscollection() => 'users';
4.     //one user-> uid auth.
5.      static String user(String userid) => 'users/$userid';
6.      //all retail stores
7.      static String retailstores()=> 'retail_stores';
8.       //one retail store
9.      static String blueads(Stringid)=>
10.      'retail_stores/$id/blueads';
11.     //one bluead
12.     static String bluead (String id, String blueid) =>
13.     'retail_stores/$id/blueads/$blueid';
14.     //bluead1ststore
15.     static String onebluead(String blueid)=>
16.     '/retail_stores/rwOEANtIjZVD0FRjnE6o/$blueid';
17.     //allblueads1stretail
18.     static String blueadscollection() =>
19.     '/retail_stores/rwOEANtIjZVD0FRjnE6o/blueads/';
20.     }
```

Codigo 14 - FirestorePath Class

Como se ve en Codigo 14 , el UID o token de acceso de aquellos usuarios autenticados con email se pasa como parámetro para utilizarlo como ID de un documento, acceder a él y obtener o modificar los datos de forma inmediata. Los usuarios anónimos o autenticados con el móvil no se guardan.

Por otro lado, se "fija" el ID del primer establecimiento (el único creado) para simplificar las querys, ya que para acceder a la subcolección blueads, hay que acceder

primero al documento de un retailstore. Todas estas consultas son asíncronas y se vuelven complejas si la jerarquía aumenta.

Se han desarrollado métodos de rol de administrador, para guardar, modificar o borrar BlueAds, asociar a un BlueAd una MAC Bluetooth escaneada, y también para gestionar la información de los usuarios. Las consultas en Cloud Firestore son asíncronas, por lo que es necesario trabajar con Streams y Futures.

Ya que en este proyecto se utiliza el modelo para manipular datos, y los documentos de Cloud Firestore tienen una estructura Map<String, Dynamic>, es necesario un proceso de mapeo para guardar u obtener datos correspondientes a instancias de la clase Usuario y de la clase BlueAd desde documentos de las colecciones. También se mapean listas de documentos a listas de objetos, ya que en determinadas consultas se devuelven varios documentos como resultado.

```
1. //constructor firebase -> obtener doc concreto de coleccion
2.   BlueAd.fromFirestore(DocumentSnapshot doc) :
3.         _fecharegistro=(doc.data()['fecharegistro'] as Timest
   amp).toDate(),
4.         _zona=doc.data()['zona'],
5.         _url=doc.data()['url'],
6.         _image=doc.data()['image'],
7.         _uid=doc.data()['uid'],
8.         _expiration= (doc.data()['expiration'] as Timestamp).
   toDate();
9.  //constructor firebase -> subir doc concreto a coleccion
10.
11.       Map <String, dynamic> toFirestore() =>
12.           {
13.             'fecharegistro': _fecharegistro,
14.             'zona' : _zona,
15.             'url' : _url,
16.             'expiration': _expiration,
17.             'uid' : _uid,
18.             'image' : _image
19.           };
20.
21.     //map stream de querysnapshots a List<BlueAd>
22.     List<BlueAd> toBlueAdList(QuerySnapshot query)
23.     {
```

```
24.       //lista forzada, porque map devuelve un iterable lista
25.         return query.docs.map((doc) =>
26.       BlueAd.fromFirestore(doc)).toList();
27.       }
28.
29.       //map stream de docsnapshots a BlueAd
30.       BlueAd toBlueAd(DocumentSnapshot doc)
31.       {
32.
33.         return BlueAd.fromFirestore(doc);
34.       }
35.
```

Codigo 15 - Mapeo Modelo-BBDD

Como se puede observar en Codigo 15 , a partir de una captura en tiempo real de un documento (DocumentSnapshot) se obtiene un BlueAd mediante el constructor nombrado BlueAd.fromFirestore. Asimismo, se obtiene una lista de BlueAds a partir de una captura en tiempo real de un conjunto de resultados (QuerySnapshot), mapeando cada uno de estos en un BlueAd con el constructor, y finalmente juntando todos los BlueAds en una lista.

Para la clase Usuario, se han creado los métodos equivalentes mapeando los atributos en cuestión.

❖ Cloud Storage

En la capa de negocio se ha desarrollado la lógica para subir imágenes (en forma de File) al bucket, tanto del perfil de usuario como de los BlueAds, así como para descargar las URLs de estas y guardarlas en los documentos de la BBDD como valor de los atributos photoURL(Usuario) o image (BlueAd), según corresponda. Para manejar las rutas de forma estática se ha utilizado la clase Storage Path.

```
1. //storage paths
2. class StoragePath {
3. static String profileimg(String userid) =>
4. 'users/profile_img/$userid';
5. static String beaconimg(String zona) =>
6.  'retail_stores/ads_img/$zona';
```

```
7. }
```

Codigo 16 - StoragePath Class

❖ Hosting

Para configurar el servicio de Hosting, se ha utilizado el Firebase CLI, el cual requiere npm (paquetes Node.js) para instalarlo. Una vez se inicializa el proyecto Firebase y se configura el servicio Hosting en la consola, se crea una carpeta con un index.html a mostrar por defecto y un 404.html por si ocurre un error en el despliegue. En este caso, se han diseñado páginas web sencillas con HTML5, JS, CSS Y Bootstrap4, con imágenes y videos para cada BlueAd. Estas páginas web son el contenido exclusivo que se muestra al usuario según donde esté.

Asimismo, para este proyecto se han creado dos targets o perfiles de hosting, con distintos dominios: usuario y administrador. La idea es que al usuario solo se le muestren las URLs pertenecientes a su dominio. Los targets se configuran en el archivo firebase.json.

```
1. {
2.   "hosting":
3.   [
4.     {
5.     "target": "user",
6.     "public": "hosting/user",
7.       "ignore": [
8.         "firebase.json",
9.         "**/.*",
10.             "**/node_modules/**"
11.          ],
12.          "appAssociation": "AUTO",
13.          "rewrites": [
14.            {
15.              "source": "/",
16.              "destination": "/myblueadstore.html",
17.              "dynamicLinks": true
18.            },
19.            {
20.              "source": "/welcome",
```

```
21.                 "destination": "/myblueadstore.html",
22.                 "dynamicLinks": true
23.               },
24.               {
25.                 "regex": "/jewelry(/.*)?",
26.                 "destination": "/jewelry/jewelry.html",
27.                 "dynamicLinks": true
28.               },
29.       (…)
30.             ]
31.         },
32.           {
33.             "target": "admin",
34.             "public": "hosting/admin",
35.             "ignore": [
36.               "firebase.json",
37.               "**/.*",
38.               "**/node_modules/**"
39.             ],
40.             "appAssociation": "AUTO",
41.             "rewrites": [ { "source": "/**", "dynamicLinks":
    true } ]
42.           }
43.           ]
44.       }
```

Codigo 17 - Archivo firebase.json

En el Codigo 17 utiliza la propiedad rewrite, para configurar Dynamic Links personalizados y mostrar el mismo contenido a aquellas URLs que cumplan un patrón determinado.

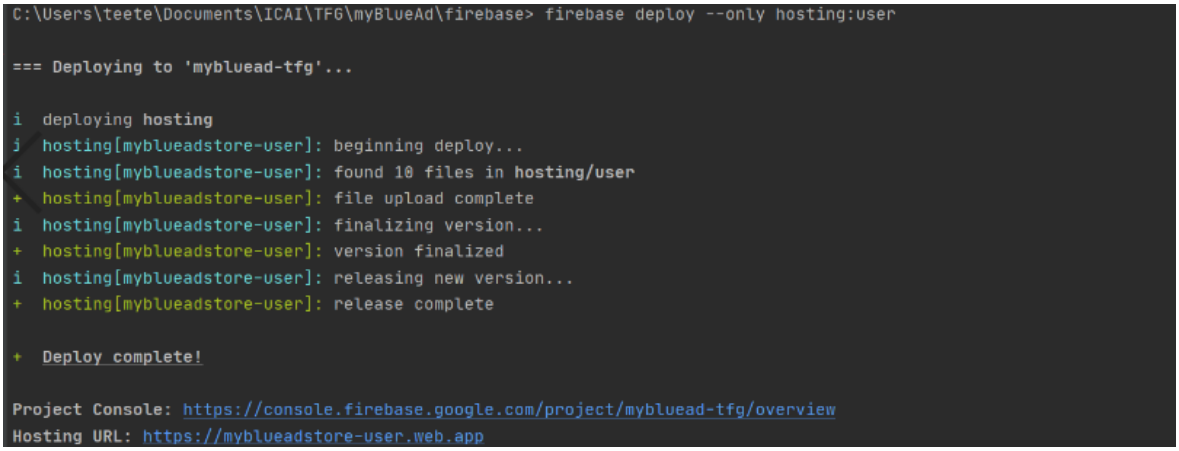


Ilustración 48 - Deploy Firebase Hosting CLI

5.3.4 APLICACIÓN MÓVIL – BEACONS

En este apartado se trata la interacción entre la app móvil y el conjunto de balizas. Cualquier usuario que haya accedido a la app, siempre que tenga la funcionalidad Bluetooth activada (y suponiendo que su dispositivo móvil tiene versión Bluetooth 4.0 o mayor) podrá comenzar a buscar o escanear blueads mediante el ya comentado proceso scanning.

Actualmente Flutter no soporta oficialmente la funcionalidad BLE y en consecuencia ningún protocolo beacon (a diferencia de Android con APIs compatibles con BLE partir de la versión 4.3). Por ello, es necesario utilizar algún paquete o plugin de un desarrollador independiente para implementar el scanning.

Se ha decidido utilizar el plugin multiplataforma FlutterBlue (versión 0.8.0), del desarrollador Paul Demarco. Este paquete, cuya última versión es de marzo de este mismo año, tiene un 96% de popularidad y puntuación notable (90/130). Es el más recomendado por la comunidad de Flutter para desarrollos que requieran BLE, por su fácil uso [50]. Se recomienda probar en dispositivos físicos.

Ilustración 49 - Logotipo Plugin FlutterBlue

La gran ventaja de este plugin, es que ofrece funcionalidad BLE tanto para iOS como para Android, por lo que es un complemento perfecto para una app desarrollada en Flutter. Sin embargo, este plugin trabaja a nivel BLE, no con protocolos beacon específicos. Por tanto, la comunicación app-baliza se lleva a cabo mediante BLE y no mediante un protocolo beacon. Ya que el SensorTag, además de ser compatible con iBeacon, utiliza Bluetooth 5.0, la comunicación BLE será satisfactoria, pero no se podrán obtener los valores UUID, Major o Minor en el proceso de advertising. Esto no es una desventaja, ya soluciona un problema de compatibilidad hardware encontrado durante el desarrollo del proyecto: el protocolo iBeacon de los SensorTags es cerrado y compatible con iOS, sin embargo se utilizan dispositivos Samsung (Android) para probar la aplicación vía debug.

La principal dificultad que se ha encontrado al utilizar este plugin es la falta de documentación en la API.

Para este proyecto, solamente es necesario escanear para descubrir las balizas entre todos los dispositivos BLE encontrados. Como ya se ha indicado, las balizas solamente anuncian su presencia, no requieren de ninguna conexión. Según la API del plugin, con una instancia de la clase FlutterBlue, se pueden descubrir dispositivos BLE cercanos mediante un proceso de escaneo (BluetoothDevice). De cada dispositivo encontrado (ScanResult), se puede obtener la información de anuncio (AdvertisementData) o su identificador (DeviceIdentifier (String id)). Este identificador no es más que la Bluetooth MAC, que como ya se ha explicado, en el caso de los SensorTag nunca cambiará. La clase AdvertisementData recoge la única información que realmente transmite una baliza. Según la API, en esta clase se obtienen propiedades como identificador del fabricante, la potencia de transmisión, si está disponible

o no, o un UUID del servicio (subcategoría GATT) de advertising. Este último valor no queda claro en la API si cambia o no.

```
1. //metodo map -> pasar una funcion que convierte los
   QuerySnapshots a List (o lo que sea)
2. Stream <List<BlueAd>> getBlueAds() {
3.   return db.collection(FirestorePath.blueadscollection()).order
   By('fecharegistro').snapshots().map(toBlueAdList);
4. }
```

Codigo 18 - getBlueAds

En el Codigo 18 se muestra el método para obtener la lista ordenada de blueads de la BBDD, la cual se actualiza en tiempo real.

```
1. // Provider, recuperamos los blueads de la base de datos en
   tiempo real
2.     final List<BlueAd> blueads = Provider.of<List<BlueAd>>(co
   ntext);
3.     //macs BBDD
4.     List <String> dbmacs=[];
5.     //macs escaneadas
6.     List <String> scanmacs=[];
7.     //macs bbdd distintas a null
8.     List<String> macsnotnull=[];
9.
10.     //mac coincidente
11.     String foundmac=null;
12.
13.     //rellenar lista macs base de datos y macs not null
14.         blueads.forEach((bluead) {
15.           dbmacs.add(bluead.uid);
16.           if (bluead.uid!=null)
17.             macsnotnull.add(bluead.uid);
18.         });
19.     StreamBuilder<List<ScanResult>>(
20.         stream:FlutterBlue.instance.scanResults,
21.         initialData: [],
22.         builder: (c, snapshot)
23.         {
24.
25.           //bucle para cada scanresult
26.           snapshot.data.forEach((r) {
27.     //coincidencia macs base de datos y Bluetooth MAC
   escaneada
28.             if (dbmacs.contains(r.device.id.id)) {
```

```
29.            //evitar que se repitan las macs coincidentes ya
   guardadas
30.                if (!scanmacs.contains(r.device.id.id))
31.                    scanmacs.add(r.device.id.id);
32.                 if (scanmacs.length>1) {
33.                                    //se han encontrado más de una
   coincidencia, se calcula la Bluetooth MAC con mayor rssi
34.                    foundmac = getmaxrssi(snapshot.data);
35.                  }
36.                  else {
37.                    //solo se encuentra una
38.                     foundmac = scanmacs[0];
39.                      }
40.                }
41.                });
42.               //mac escogida
43.           if (foundmac!=null) {
44.           //obtener BlueAd de la lista de BBDD uid=foundmac
45.             blueadshow = blueads.firstWhere((bluead) =>
46.              bluead.uid == foundmac, orElse: () => null);
47.             }
```

Codigo 19 - Scanning

En el Codigo 19 se explica el proceso scanning con Dart. Una vez se crean Listas inicializadas con las Bluetooth MACS, se construye un StreamBuilder que recibe un Stream de List<ScanResults>. Los dispositivos detectados se van añadiendo de forma asíncrona a la lista, hasta que se pare el scan. Si la captura en tiempo real del Stream tiene algún dato, en este caso, un ScanResult o dispositivo BLE detectado, se busca alguna coincidencia entre su Bluetooth MAC y la lista de MACs cargadas de la Base de Datos y si hay, se queda con dicha Bluetooth MAC.

Si hay varias coincidencias, esto significa, que se han encontrado varias balizas, la Bluetooth MAC elegida será aquella que corresponda a la baliza más próxima (max RSSI). Finalmente, entre todos los BlueAds de la BBDD, se muestra al usuario aquel con el uid coincidente a la Bluetooth MAC elegida.

```
1. //funcion que devuelva el id con el maximo rssi, el que esta
   más cerca.
2. String getmaxrssi(List <ScanResult> results) {
3.   List <int> _rssi = [];
4.   ScanResult def;
```

```
5.   for (ScanResult r in results) {
6.     _rssi.add(r.rssi);
7. //se queda con el maximo de la lista
8.     _rssi.reduce(max);
9.     if (r.rssi == _rssi.reduce(max))
10.  //es el scanresult con rssi max
11.         def = r;
12.    }
13.  //se devuelve mac dispositivo rssi max
14.    return def.device.id.id;
15.  }
```

Codigo 20 - getmaxRSSI

Actualmente hay 5 BlueAds en la base de datos y solo 2 SensorTags físicos, por lo que solo dos documentos tienen un uid con valor, siendo el resto nulos.

Cabe destacar, que para que el proceso de scanning se repita indefinidamente hasta que el usuario salga de la pantalla o haga click en la preview del BlueAd con la MAC "encontrada", se inicia el scan permitiendo duplicados. Esto significa que el StreamBuilder puede recibir de forma asíncrona el mismo ScanResult varias veces, mientras dure el scan.

Si no se permitiesen duplicados, el StreamBuilder solo recibiría ScanResults nuevos, lo que resultaría en que solo se detectaría cada SensorTag una vez, y el usuario solo recibiría el contenido del BlueAd asociado a dicha baliza esa vez. Para repetir el "recorrido" de contenido, tendría que volver a la pantalla principal y pulsar de nuevo en el botón de Scan, lo cual empeora la experiencia de usuario.

Capítulo 6. ANÁLISIS DE RESULTADOS

En este capítulo se muestran los resultados obtenidos tras el desarrollo del sistema, se comentarán los aspectos más relevantes y se hará un análisis de estos. Se dividen los resultados en las 3 grandes partes del desarrollo del proyecto: interfaz de usuario (UI), back-end en el servicio cloud y la comunicación BLE entre dispositivo y balizas.

6.1 INTERFAZ DE USUARIO

A continuación, se explicarán todas las pantallas de la aplicación según el orden de navegación del usuario. Para ello, se dividirán las pantallas en 3 bloques principales: Inicio, Home Usuario y Bluetooth.

6.1.1 INICIO

Este bloque engloba las páginas de inicio, acceso y registro, además de las secciones del menú lateral o drawer. La página de inicio será con la que arranque la aplicación siempre que se hayan inicializado correctamente los servicios de Firebase y no exista ninguna sesión abierta.

Si el usuario tiene conexión a Internet, podrá acceder a la pantalla de Registro o Acceso al pulsar en el botón en cuestión. Como ya se ha explicado, el usuario dispone de múltiples formas de acceso, además de la posibilidad de recuperar su contraseña. En caso de error de autenticación se avisa al usuario con un mensaje en forma de SnackBar. Además, todos los formularios poseen validación.

En acciones como acceso mediante link o recuperar contraseña, se avisa al usuario de que consulte su buzón de correo electrónico, ya que se la habrá enviado un correo con un link de redirección a la app (Dynamic Link) y completar la tarea pendiente. Con este sistema, el back-end se asegura de que email con el que el usuario intenta acceder es el suyo.

Ilustración 50 - Inicio App, Drawer Inicio y Secciones Drawer

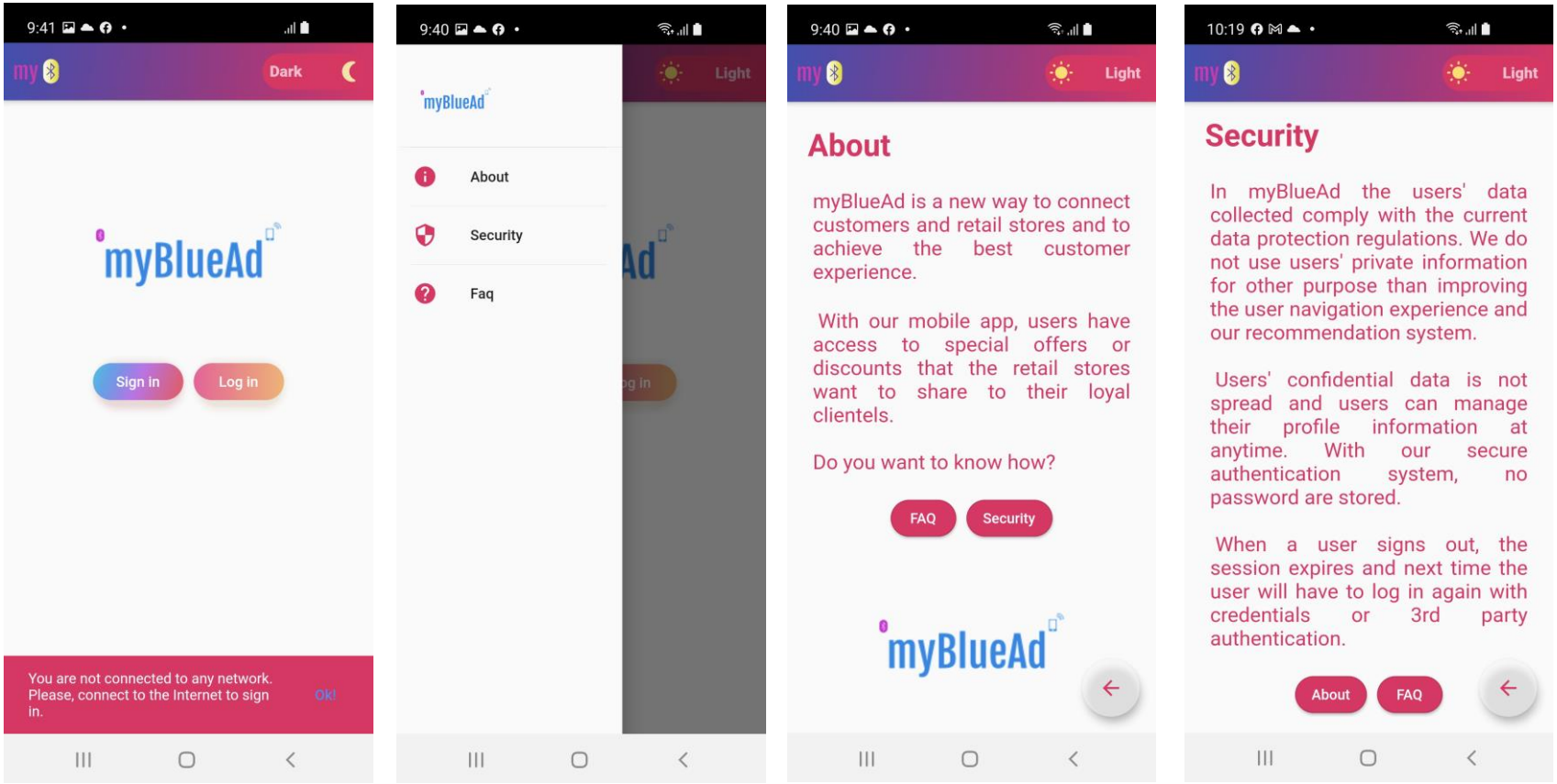
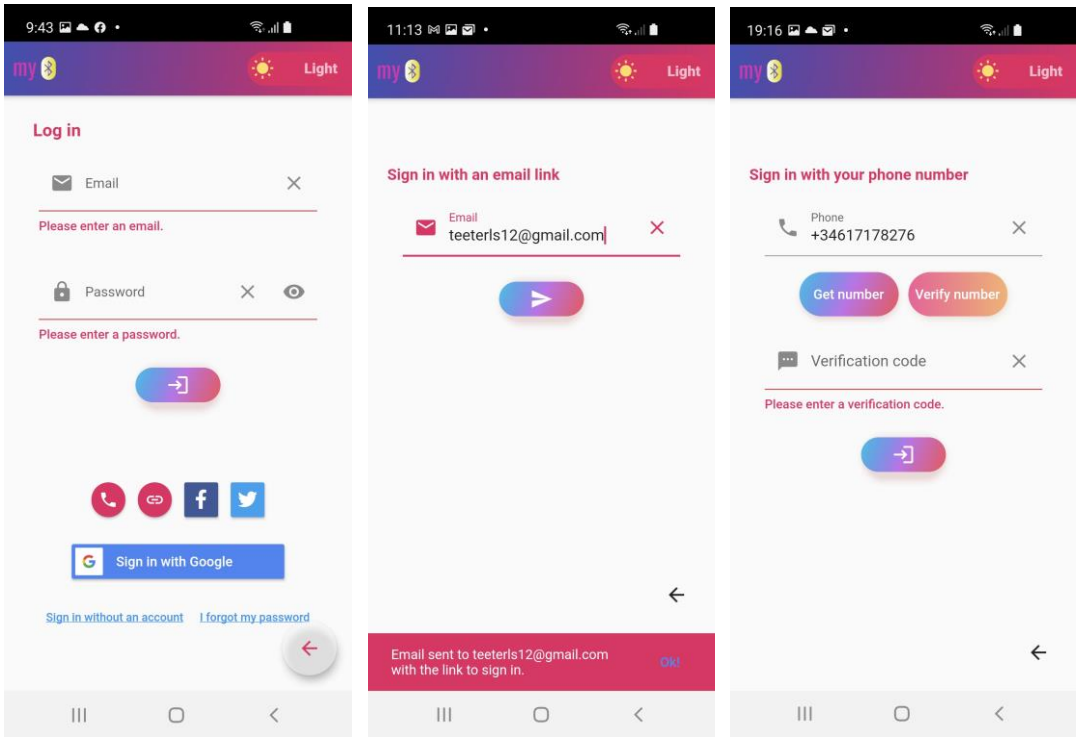
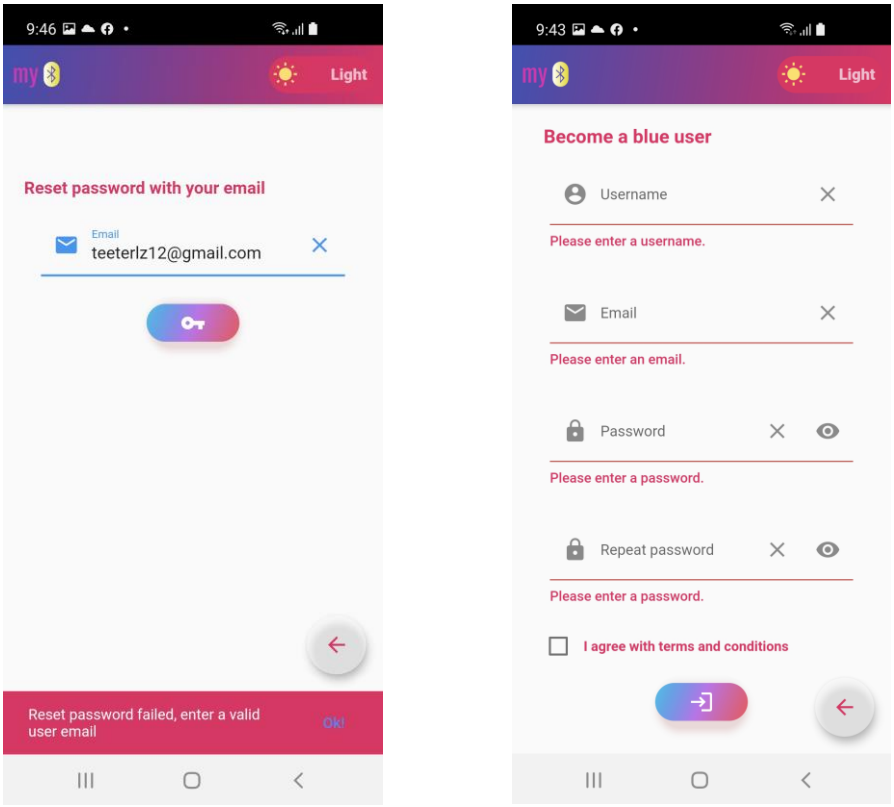


Ilustración 51 - Métodos de acceso y registro





6.1.2 HOME USUARIO

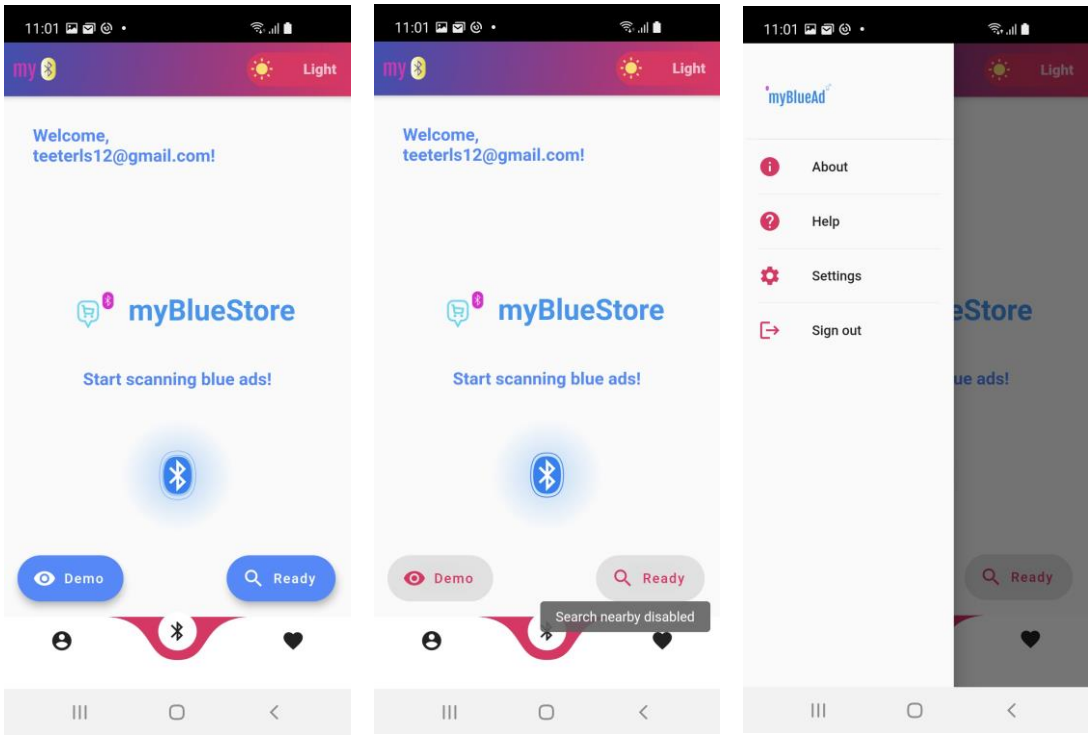
Este bloque abarca la sección home o principal del usuario, formada por varias pantallas, una barra de navegación inferior y un menú lateral. Las pestañas de navegación son distintas según la forma de acceso escogida por el usuario (email o anónimo/móvil):

- Usuario con email:
  - Perfil: El usuario puede ver sus datos personales y modificarlos en tiempo real.
  - Página principal: Página con dos botones principales: Demo y Scan, que se explican en el siguiente bloque.
  - Lista de favoritos: Si el usuario tiene algún favorito, se muestra una vista previa del BlueAd favorito, así como el tiempo que queda para que este contenido deje de estar disponible. Cuando llegue la fecha de expiración, se borra automáticamente de la lista. El usuario puede volver a visualizar el contenido favorito, así como borrar elementos de la lista.
  - Drawer (menú lateral): Sección donde el usuario puede configurar su cuenta (email, contraseña, borrar cuenta)

- Usuario anónimo o via móvil:
  - Registro: Pantalla de registro mediante email y contraseña, para darle la posibilidad a este tipo de usuario de crearse una nueva cuenta. Si se registra satisfactoriamente, automáticamente la aplicación le mostrará todas las funcionalidades del tipo de usuario con email.
  - Página principal: Página con un único botón: Scan.

En esta sección se detectan los cambios en el estado del Bluetooth en tiempo real en el dispositivo móvil para habilitar o deshabilitar ciertos botones que permiten la navegación al siguiente bloque.

Ilustración 52 - Página Principal Usuario con cambios Bluetooth, Drawer y Settings



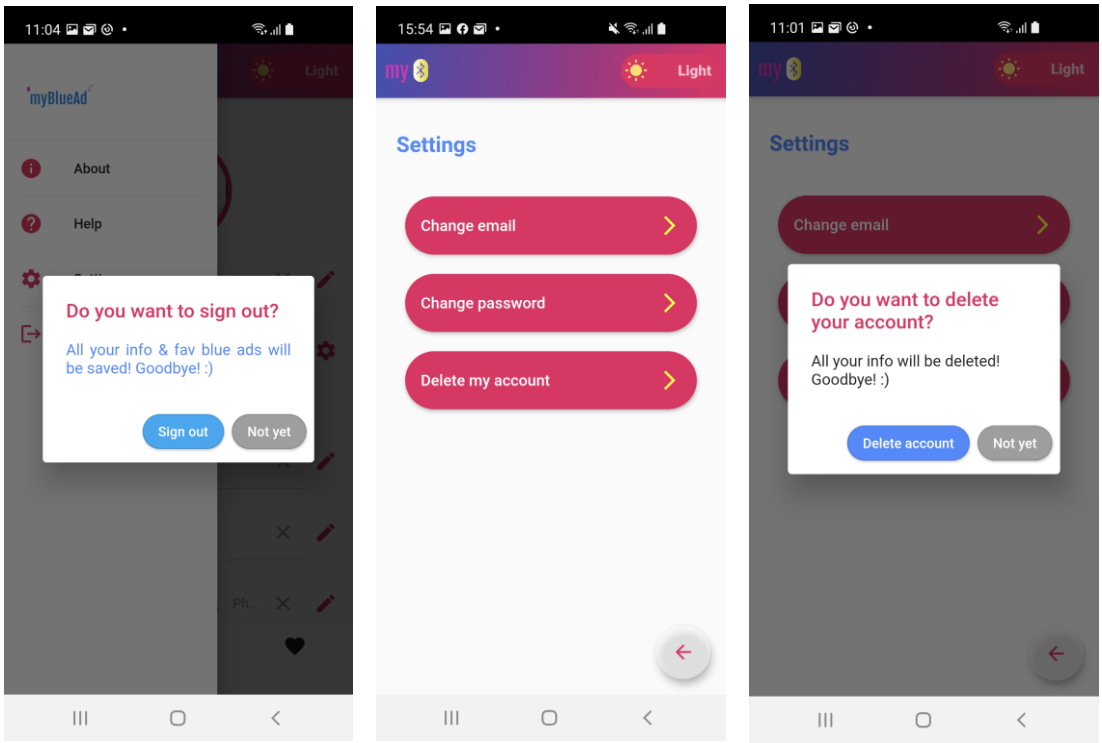
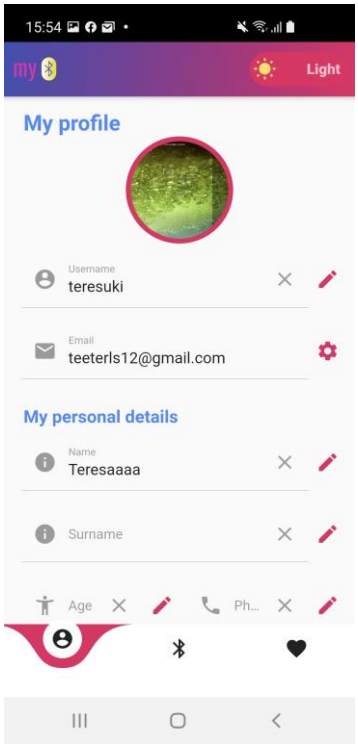
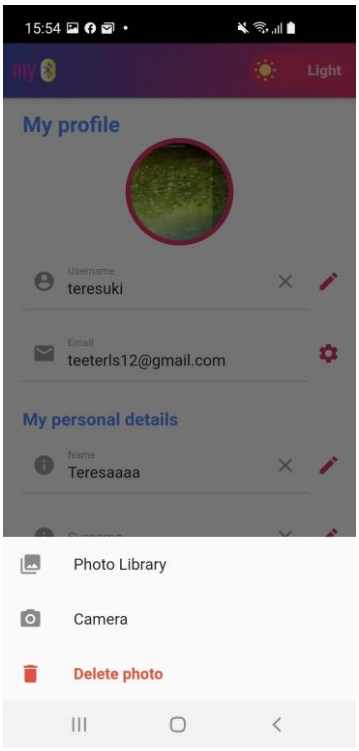
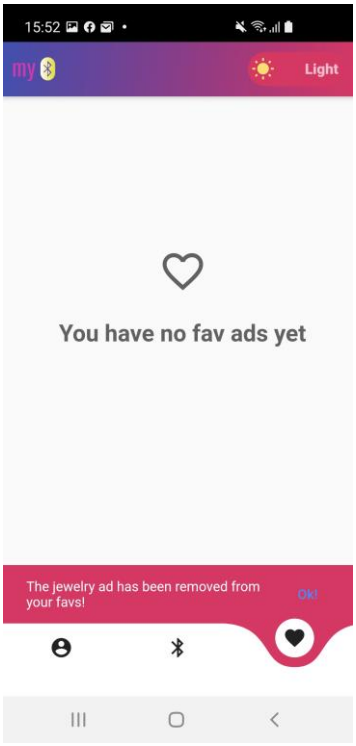
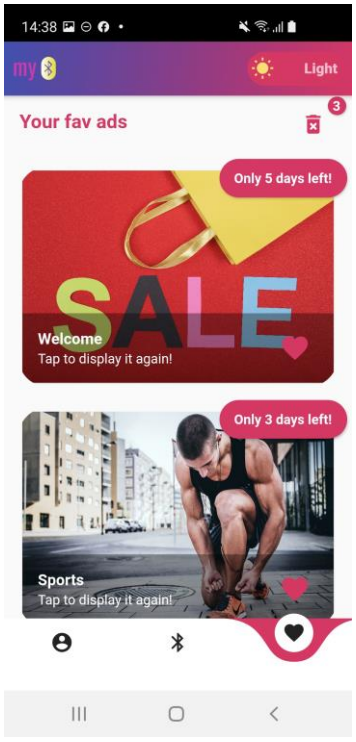


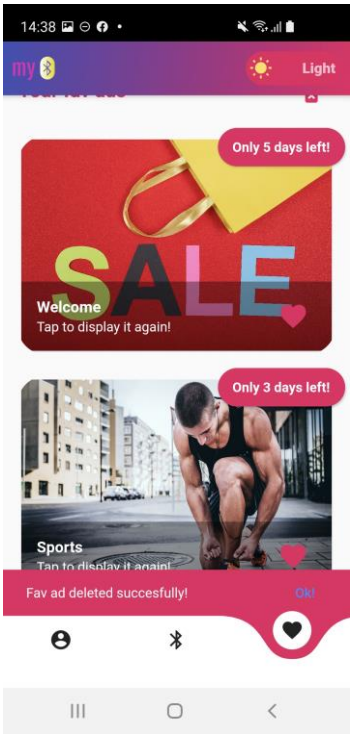
Ilustración 53 - Páginas Perfil y Favoritos Usuario











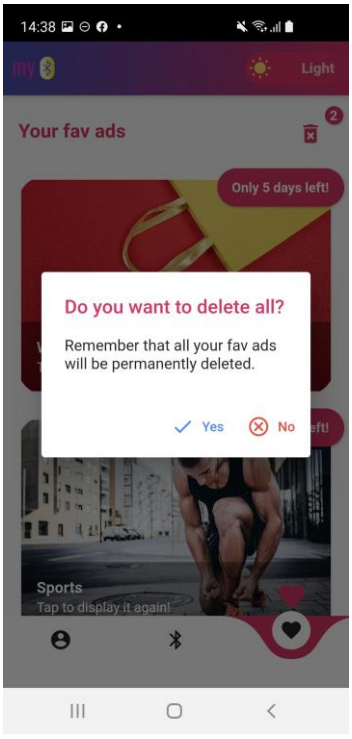
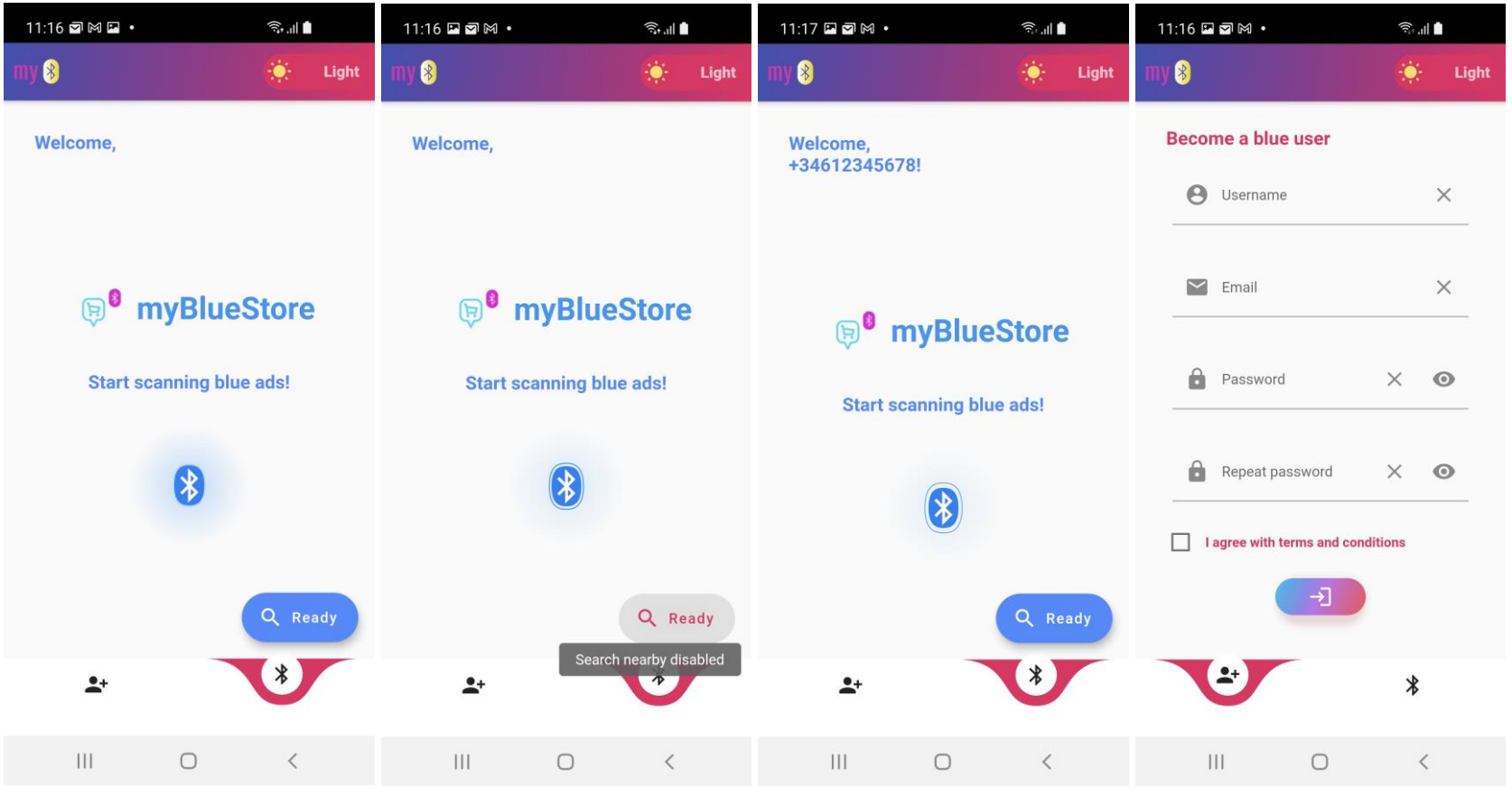


Ilustración 54 - Página Principal Usuario anonimo/móvil,cambios Bluetooth,Registro



6.1.3 BLUETOOTH

Este bloque incluye las páginas y funcionalidades de Demo y de Scan, además de la pantalla de visualización de contenido. Como ya se ha explicado, solamente los usuarios que han accedido con su email pueden visualizar la Demo.

- Scan

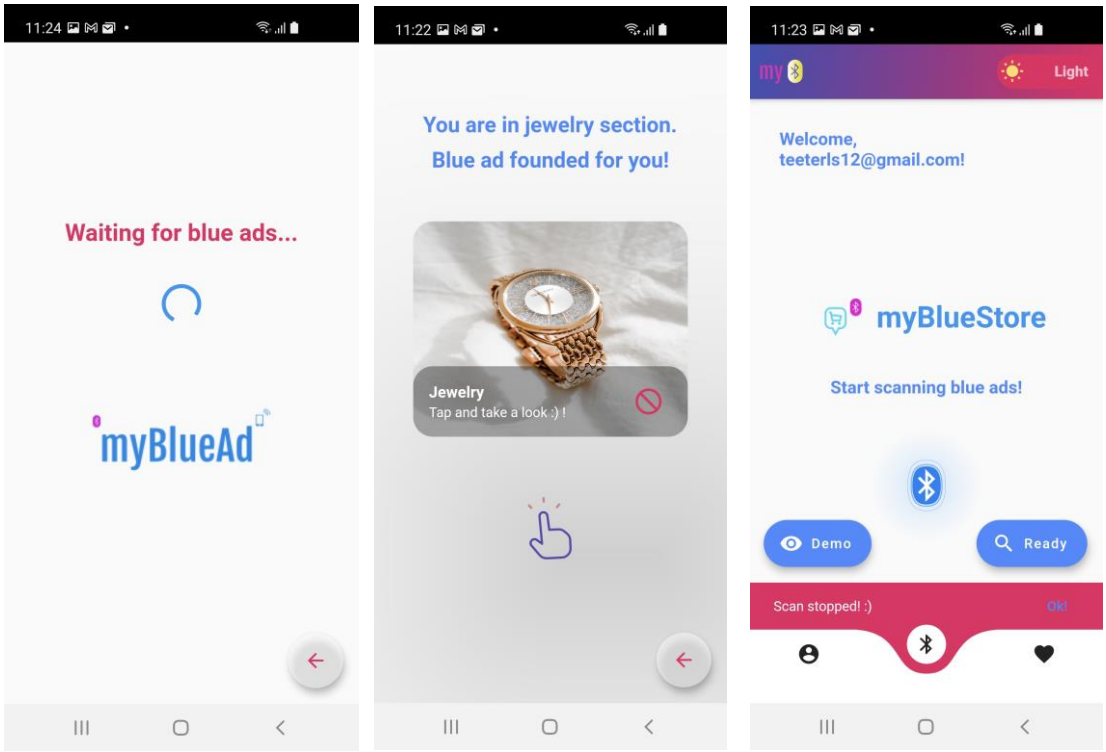
  Una vez el usuario le da al botón para empezar a escanear, se muestra una pantalla de espera hasta que se encuentra una previsualización del BlueAd en forma de ventana emergente o pop-up. Si el usuario desactiva Bluetooth, la pantalla de espera se mostrará indefinidamente. El usuario puede darle click para ver el contenido completo, en este caso el scan se paray la app le dirige a una pantalla donde se le muestra una página web dentro de un WebView. Si vuelve atrás, también el scan se para.

  Solo si el usuario ha accedido con email, se hace visible una barra inferior con botones para que esta puede guardar el contenido como favorito, o descartarlo.

- Demo

  Una vez arranca la demo, se muestra cada 20 segundos una previsualización (en pantalla completa) de un BlueAd distinto. Si hace click en la pantalla, se avisa al usuario que está viendo solo una demo, y no puede acceder al contenido exclusivo (url, que se muestra en el Scanning) El usuario puede parar la demo en cualquier momento, o repetirla una vez finalizada.
  Si la aplicación detecta que el Bluetooth esta desactivado durante la demo, esta se parará y mostrará al usuario una pantalla avisándole de que debe activar Bluetooth para continuar.

Ilustración 55 - Scan y Pantalla con BlueAd encontrado, Like para Usuarios con email



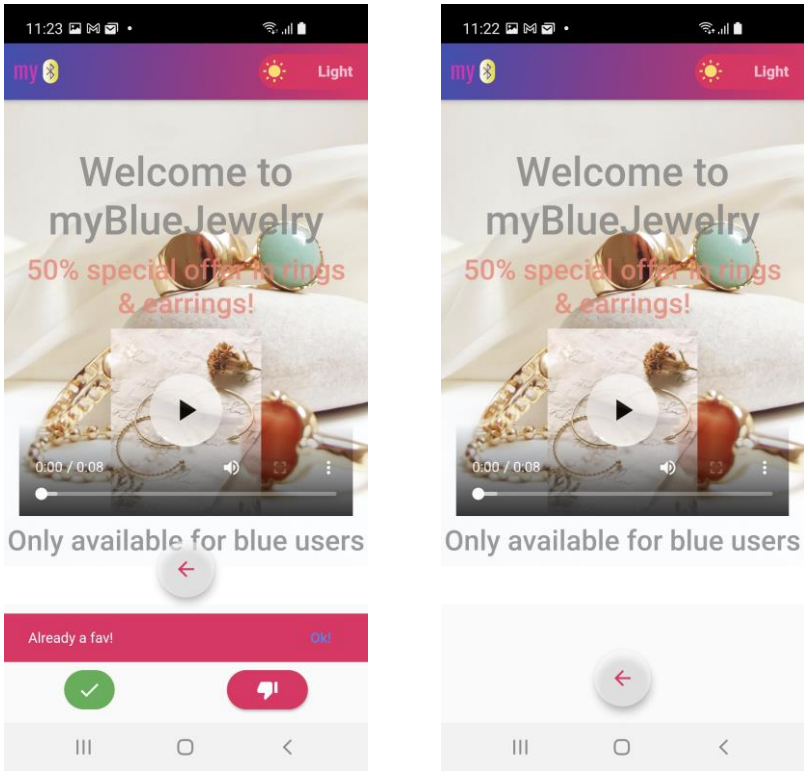
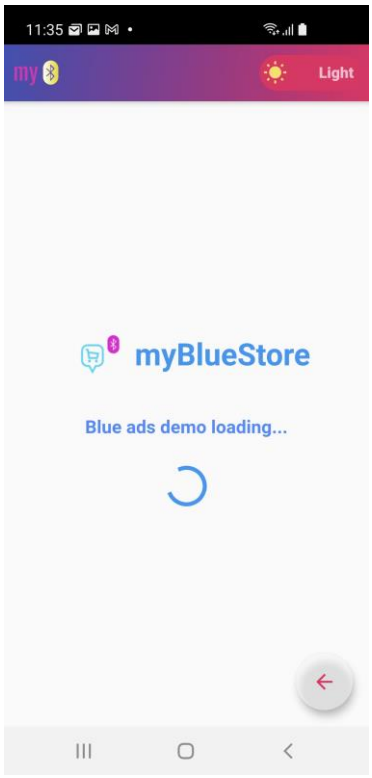
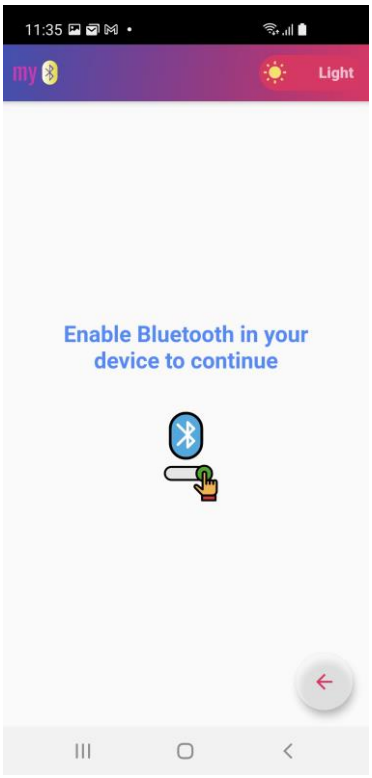
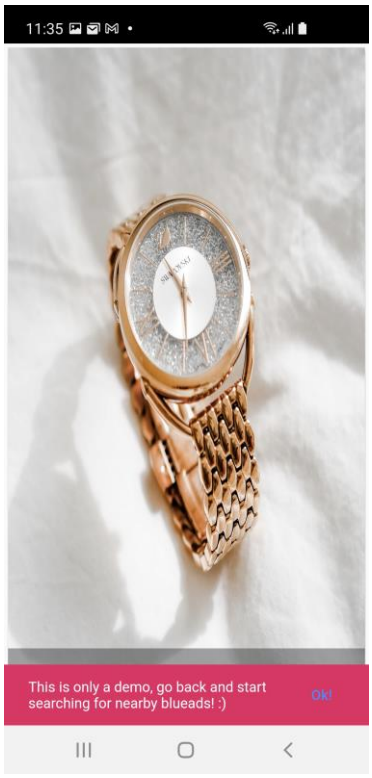
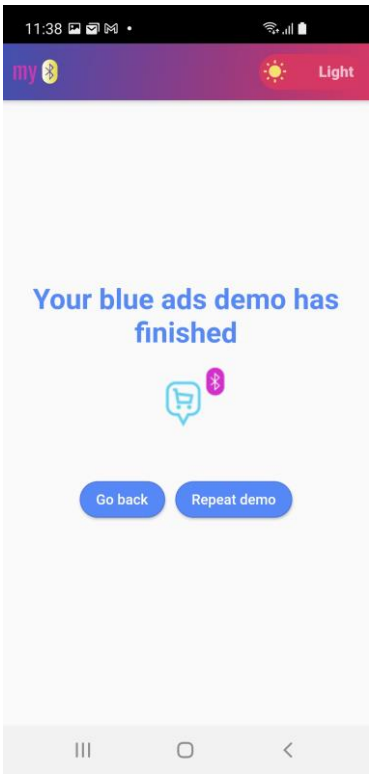


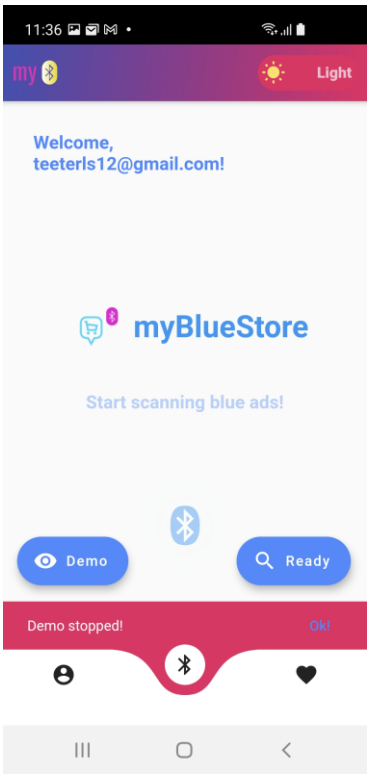
Ilustración 56 - Inicio Demo, Detección Bluetooth, Stop, Fin











6.2 BACK-END EN LA NUBE

A continuación, se muestra el correcto funcionamiento de los servicios de Firebase de forma visual.

❖ Autenticación

Identificador	Proveedores	Fecha de creación	Fecha de acceso ↓	UID de usuario
teeterls12@gmail.com		5 abr. 2021	25 jun. 2021	llNXSa2PB0fYUKAcbaT8jwoxG8Z2
(anónimo)		25 jun. 2021	25 jun. 2021	n06u7k6CqgZZZS5ynfrgleWTmdl1
(anónimo)		24 jun. 2021	24 jun. 2021	rfM9UG82ghM9y9xv9YtNEWjGJ5q2
+34612345678		2 abr. 2021	24 jun. 2021	ORlrzvlMB0enkOHoIRQrMHtL34h1
(anónimo)		24 jun. 2021	24 jun. 2021	mbHv7ssvaBQqhraS3kB8Gy62Igb2
pau@gmail.com		9 jun. 2021	24 jun. 2021	WtAXedndUYTcfk36KLxGHVKBCH...
luis12@gmail.com		19 jun. 2021	24 jun. 2021	fAqD0FVrnxS0qTllY6iLj2uCZff2
paca@gmail.com		11 jun. 2021	24 jun. 2021	nRDWfDgiFfR4tkOjfMWyPIiPLLi2
carmen@gmail.com		9 jun. 2021	24 jun. 2021	67MzxQ6PSJOO59GLA7T9RNr5Q...

Ilustración 57- Consola Firebase Auth

Se han realizado 120 autenticaciones satisfactorias. Como se puede observar en la imagen, un usuario se puede autenticar con distintos métodos, sin que haya conflicto de credenciales. Además, todos los métodos de acceso habilitados funcionan, y a cada usuario se le otorga un token UID único.

```
D/FirebaseAuth( 2029): Notifying id token listeners about user ( llNXSa2PB0fYUKAcbaT8jwoxG8Z2 ).
D/FirebaseAuth( 2029): Notifying auth state listeners about user ( llNXSa2PB0fYUKAcbaT8jwoxG8Z2 ).
```

Ilustración 58 - Ejemplo notificacion UID

❖ Cloud Firestore

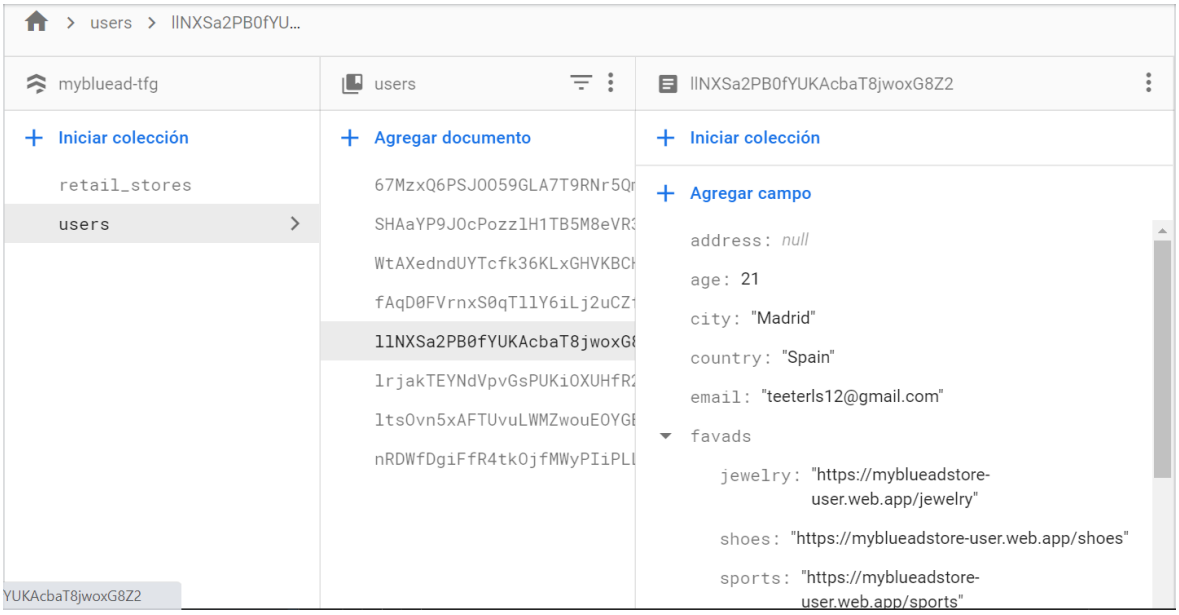


Ilustración 59 - Colección Users BBDD

En la colección users, se comprueba que solo aquellos usuarios que acceden con su email son guardados en esta, utilizando los tokens UID como identificador de documento.

Se comprueba cómo se modifican los datos de forma correcta. Por ejemplo, el atributo favads tiene estructura de Map<String, String>, y cada vez que un usuario guarde un favorito, aparecerá un nuevo elemento del mapa con forma “zona”:”url”. Además, la foto corresponde a una URL de una imagen del bucket de Cloud Storage.

En la subcolección blueads, se comprueba que los valores de las uid son los correctos (también mediante el debug del scan que se explica a continuación), así como las URL de las páginas web y la URL de las imágenes en el bucket.

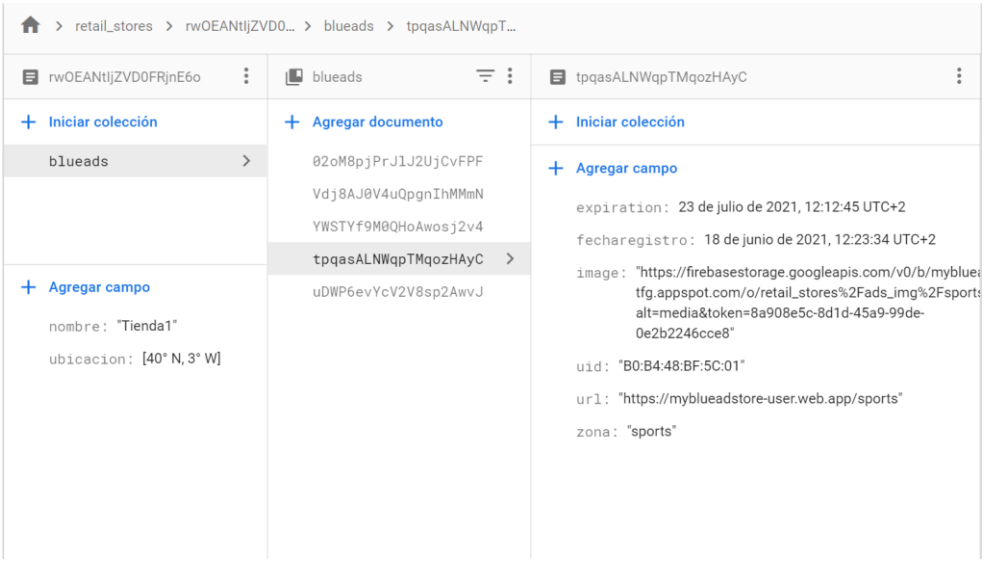


Ilustración 60 - Subcoleccion Blueads BBDD

❖ Cloud Storage

Se comprueba la correcta subida de las imágenes en las carpetas correspondientes. Como se puede observar en las figuras, se utilizan nombres únicos para las imágenes del bucket, con el fin de poder descargarlas fácilmente. En el caso de las imágenes de los usuarios, se utiliza el token UID, y para los blueads se utiliza la zona (que en este proyecto son valores únicos).

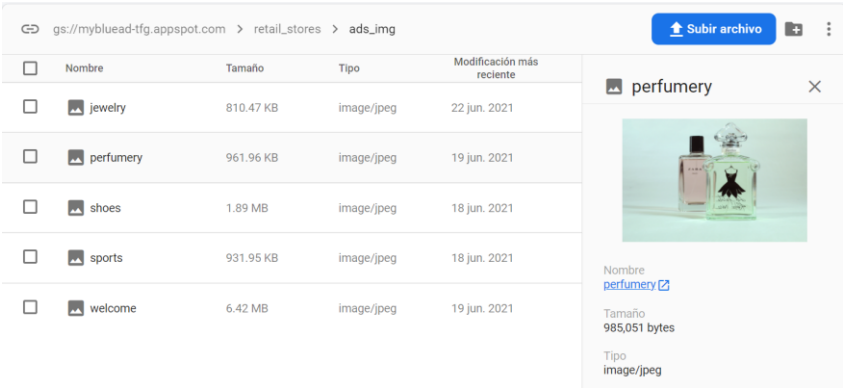


Ilustración 61 - Bucket RetailStores Images

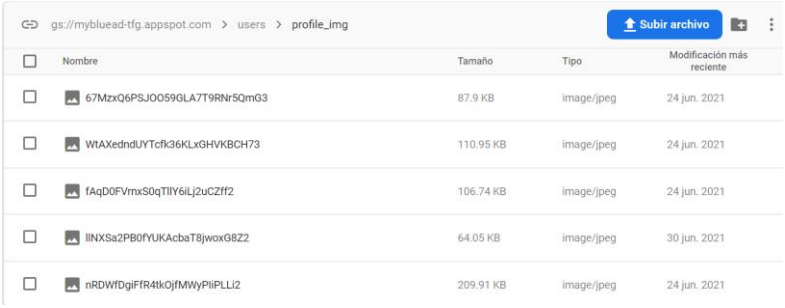


Ilustración 62 - Bucket Users Image

❖ Dynamic Links

Se comprueba el correcto funcionamiento de estos links: cuando el usuario hace click, se le redirige a la app.

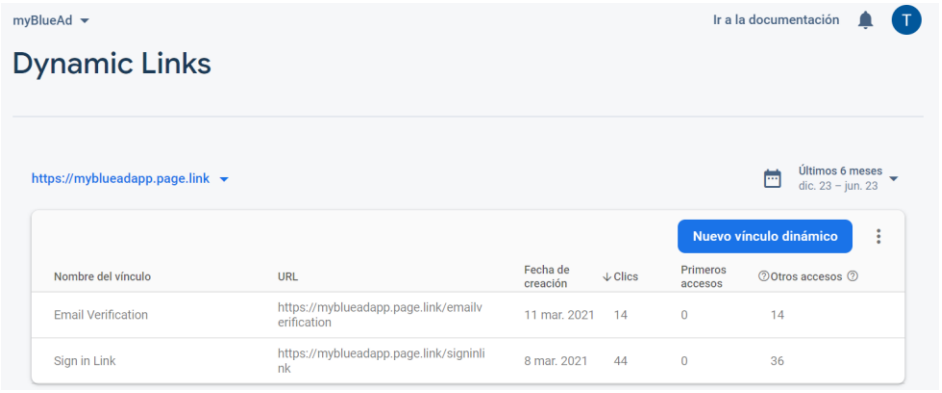


Ilustración 63 - Consola Dynamic Links

❖ Firebase Hosting

Se comprueba que todas las páginas web pertenecientes al dominio/target de usuario están disponibles en cualquier navegador, con su certificado SSL válido. También se pueden consultar el estado de las últimas actualizaciones en la consola de Firebase.

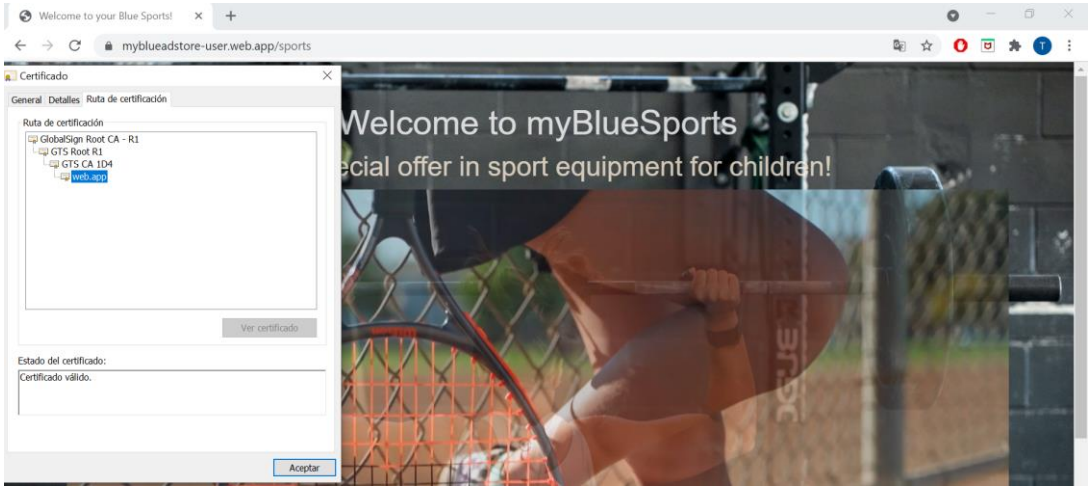


Ilustración 64 - Página web Certificado SSL Hosting

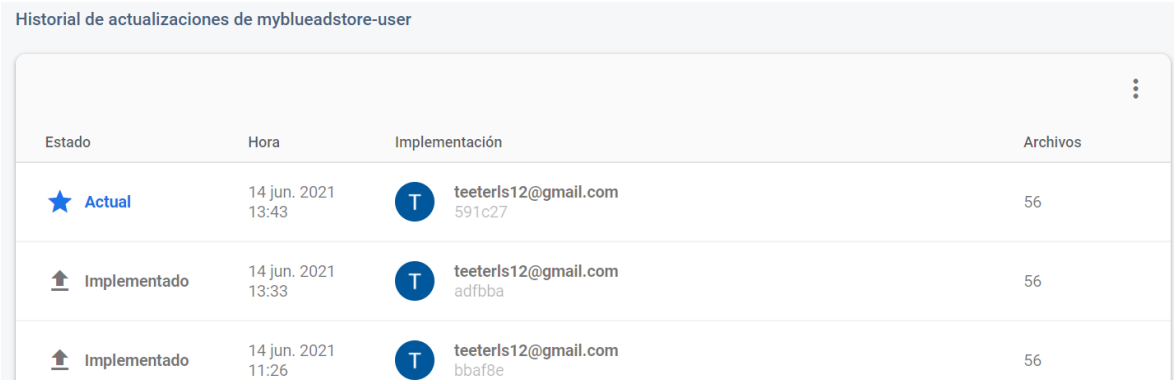


Ilustración 65 - Consola Firebase Hosting

6.3 COMUNICACIÓN BLE CON BALIZAS BLE

En este apartado se comprueba via debug el correcto funcionamiento del código en cuanto a la búsqueda tanto de la carga de MACS (o uids) en la base de datos, como de los dispositivos escaneados, buscar coincidencias y calcular los RSSI máximos.

Inicialmente se obtienen las Bluetooth MACs de los SensorTag mediante un scan que detecta el dispositivo más cercano (SensorTags lo más próximos al móvil) y se guardan como valor uid de 2 documentos (en este caso aleatorios) en la BBDD. Esta acción es de rol administrador.

```
I/flutter ( 2029): CC2650 SensorTag found!, rssi: -51, mac: B0:B4:48:BF:64:80, advertising data: AdvertisementData{localName: CC2650
 SensorTag, txPowerLevel: null, connectable: true, manufacturerData: {13: [3, 0, 0]}, serviceData: {}, serviceUuids:
 [0000aa80-0000-1000-8000-00805f9b34fb]}
```

Ilustración 66 - Ejemplo SensorTag escaneado y AdvertisementData

Como se puede observar, por defecto el fabricante no ha indicado el nivel de potencia transmitida. Este valor se puede cambiar con el Development Kit. Además de la MAC, ambién se recogen datos del fabricante, el UUID del servicio de advertising, el nombre del dispositivo y el valor RSSI.

Una vez se han guardado en la BBDD, el usuario ya puede comenzar a buscar BlueAds cercanos. Se sigue el proceso de scanning mediante el terminal:

```
I/flutter ( 2029): Scanning results:
I/flutter ( 2029):  found!, rssi: -77, mac: 52:53:AE:8D:AD:C4, advertising data: AdvertisementData{localName: , txPowerLevel: 8,
 connectable: true, manufacturerData: {76: [16, 7, 60, 27, 112, 70, 230, 203, 88]}, serviceData: {}, serviceUuids: []}
I/flutter ( 2029): This mac is not in database: 52:53:AE:8D:AD:C4
I/flutter ( 2029):  found!, rssi: -77, mac: FA:0D:AC:3E:1F:29, advertising data: AdvertisementData{localName: , txPowerLevel: null,
 connectable: true, manufacturerData: {101: [1, 201, 1]}, serviceData: {0000fdf7-0000-1000-8000-00805f9b34fb: [1, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3]}, serviceUuids: [0000fe78-0000-1000-8000-00805f9b34fb]}
I/flutter ( 2029): This mac is not in database: FA:0D:AC:3E:1F:29
I/flutter ( 2029): iPad de Teresa found!, rssi: -40, mac: 50:DE:06:3B:14:D3, advertising data: AdvertisementData{localName: ,
 txPowerLevel: 12, connectable: true, manufacturerData: {76: [16, 6, 12, 25, 84, 20, 225, 88]}, serviceData: {}, serviceUuids: []}
I/flutter ( 2029): This mac is not in database: 50:DE:06:3B:14:D3
I/flutter ( 2029): KD-49XF8096 found!, rssi: -91, mac: 5C:EA:1D:64:CD:72, advertising data: AdvertisementData{localName: ,
 txPowerLevel: null, connectable: false, manufacturerData: {301: [2, 0, 1, 16, 243, 80, 124, 246, 121, 234, 79, 58, 147, 39, 149, 41,
 61, 227, 229, 189, 56, 186, 129, 13, 149, 86]}, serviceData: {}, serviceUuids: []}
I/flutter ( 2029): This mac is not in database: 5C:EA:1D:64:CD:72
```

```
I/flutter ( 2029):  found!, rssi: -92, mac: 1C:51:11:50:33:4E, advertising data: AdvertisementData{localName: , txPowerLevel: null,
  connectable: false, manufacturerData: {19456: [2, 21, 38, 134, 243, 156, 186, 218, 70, 88, 133, 74, 166, 46, 126, 94, 139, 141, 0, 1,
  0, 0, 201]}, serviceData: {}, serviceUuids: []}
I/flutter ( 2029): This mac is not in database: 1C:51:11:50:33:4E
I/flutter ( 2029): CC2650 SensorTag found!, rssi: -29, mac: B0:B4:48:BF:5C:01, advertising data: AdvertisementData{localName: CC2650
  SensorTag, txPowerLevel: null, connectable: true, manufacturerData: {13: [3, 0, 0]}, serviceData: {}, serviceUuids:
  [0000aa80-0000-1000-8000-00805f9b34fb]}
I/flutter ( 2029): Mac in database with max rssi founded: B0:B4:48:BF:5C:01
I/flutter ( 2029): Total macs in database: [null, B0:B4:48:BF:5C:01, B0:B4:48:BF:64:80, null, null]
I/flutter ( 2029): Total macs in database founded: [B0:B4:48:BF:64:80, B0:B4:48:BF:5C:01]
I/flutter ( 2029): All macs in database founded!
```

Ilustración 67 - Resultados Scaning

Como se puede observar en la Ilustración 67, se va guardando en la lista correspondiente solo las Bluetooth MACs de los SensorTags (que coinciden con las de la BBDD) de forma asíncrona, según se van escaneando. Si hay más de una coincidencia, se calcula el RSSI máximo. También se comprueba si se han encontrado todas las balizas posibles.

```
I/flutter ( 2029): Show BlueAd with B0:B4:48:BF:5C:01 MAC
I/BluetoothAdapter( 2029): STATE_ON
I/BluetoothAdapter( 2029): STATE_ON
```

Ilustración 68 - BlueAd escaneado

Asimismo, se comprueba el correcto inicio y stop del scan, según la actividad del usuario. En este caso, el usuario para el scan ya que hace click en el preview del BlueAd mostrado.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Como ya se ha explicado, este proyecto engloba los primeros pasos para lograr el propósito final en el medio-largo plazo: desarrollar un sistema de marketing de proximidad con Balizas BLE completo, y poder aplicarlo al sector retail en particular.

Por tanto, el alcance de este proyecto es más reducido, y se resume en los objetivos (generales y específicos) explicados anteriormente (apartado 4.2). De estos objetivos iniciales, se han cubierto todos, aunque algunos con más profundidad que otros, ya que la planificación inicial y en consecuencia el tiempo dedicado al cumplimiento de cada objetivo ha sido flexible, adaptándose a cada dificultad y reto durante el desarrollo.

A continuación, se explican las conclusiones obtenidas con el alcance actual de este proyecto y los trabajos futuros más inmediatos.

7.1 CONCLUSIONES

En general, se han cumplido todos los objetivos iniciales de forma satisfactoria, ya que se ha desarrollado una aplicación multiplataforma con Flutter, atractiva, bonita y accesible, con las funcionalidades a esperar de una estrategia de marketing de proximidad (y más casos de uso añadidos), con una comunicación correcta entre app y balizas y un servicio back-end en la nube muy completo y escalable. Sin embargo, por falta de tiempo y de disponibilidad hardware no ha sido posible configurar el backend para una aplicación iOS.

Por otro lado, el objetivo que menos se ha cubierto es el testing. Si bien se han hecho tests de métodos a lo largo del desarrollo del código, no se han utilizado las herramientas de Flutter para el testing, como test de widgets o de integración de la app. Tampoco se han utilizado herramientas de Firebase para monitorizar el rendimiento de la aplicación.

El tipo de testing que más se ha utilizado ha sido el test de experiencia de usuario, ya que, en vez de utilizar emulador, se ha probado la aplicación via debug o depuración con

dispositivos Android, controlando los errores y los cambios realizados mediante el terminal integrado en el IDE.

Cabe destacar también que se ha probado el funcionamiento del sistema solo con un establecimiento, pero la estructura de la base de datos es perfectamente escalable. Asimismo, por falta de tiempo, no se ha desarrollado una plataforma para que los establecimientos puedan administrar sus contenidos. Dicha plataforma se explica a continuación como un trabajo futuro.

7.2 TRABAJOS FUTUROS

Este proyecto tiene una línea de continuidad clara para seguir avanzando en el desarrollo del sistema completo. Esta línea está muy enfocada en la escalabilidad y en la ampliación de las funciones del rol de administrador (establecimiento).

Las posibles mejoras detectadas en el apartado anterior son el punto de partida para seguir trabajando, entre las más importantes integrar herramientas de supervisión y lanzamiento de Firebase y configurar aplicación iOS.

Asimismo, en el medio-largo plazo será primordial planificar el modelo de negocio, ya que la aplicación móvil es gratis para el usuario, pero para que un establecimiento tenga acceso a este sistema y suba sus contenidos tendrá que pagar una cantidad (mensual o anual) a concretar. También es necesario contactar con proveedores de balizas con protocolo iBeacon.

Por otro lado, como ya se ha explicado, la estrategia de marketing de proximidad ha estado muy presente en las empresas durante los últimos años. Sin embargo, hoy el dato cobra más importancia que nunca. Cuando el proyecto escale, el volumen de datos aumentará (datos de usuarios, de establecimientos, de contenidos a mostrar, etcétera), y será necesario incorporar herramientas de Big Data para su gestión y análisis. Así mismo, se podrán identificar patrones entre los datos con técnicas de Machine Learning y realizar predicciones de forma automática.

Se pueden resumir los trabajos futuros en la siguiente lista:

- Configurar aplicación iOS en Firebase a partir del proyecto Flutter, para así lograr no solo un front-end multiplataforma, si no también un back-end.
- Integrar a ambas apps móviles servicios de Firebase de lanzamiento, supervisión y analíticas para monitorear el funcionamiento de la app, detectar errores y obtener informes de rendimiento, entre otros.
- Configurar la persistencia de los datos en Cloud Firestore para tener acceso a ellos offline.
- Integrar herramientas de Machine Learning mediante el kit de Aprendizaje Automático y extensiones de Firebase, algunas basadas en Inteligencia Artificial, como traducción de textos o detección de rostro.
- Integrar algoritmos y técnicas de procesado de los valores del RSSI.
- Configurar reglas Firebase en los servicios utilizados para restringuir accesos.
- Probar las aplicaciones con un número de balizas mayor, entre 5 y 10.
- Subir las respectivas apps a Play Store (Android) y App Store (iOS), creando un perfil de desarrollador y completando el proceso a seguir en ambas plataformas.
- Desarrollar una plataforma web para los establecimientos minoristas "partners", utilizando el dominio de administrador ya creado. En esta, cada establecimiento podría monitorizar la actividad y posición de los usuarios, además del funcionamiento y la cobertura BLE de las balizas mediante heatmaps. También podría dar de baja o alta a usuarios, además de añadir o eliminar contenidos asociados a determinadas balizas.

  Esta plataforma se configurará como aplicación web en Firebase y se dará uso de herramientas de analítica y supervisión. Se gestionarán todos los datos a partir de la estructura actual de la BBDD Cloud Firestore, la cual es perfectamente escalable para que se vayan añadiendo nuevos establecimientos.
- También se contempla desarrollar otra app móvil multiplataforma para que los establecimientos puedan escanear o "registrar" sus balizas fácilmente, y que los

empleados puedan trackear la posición y actividad de los clientes desde un móvil, entre otras funcionalidades.

Capítulo 8. BIBLIOGRAFÍA

[1] Daugherty, P. La era post-digital ya está aquí. ¿Estás preparado para lo que viene? *Technology Vision 2019, Accenture.* Recuperado de https://www.accenture.com/es-es/insights/technology/technology-trends-2019 el 21/1/21.

[2] Pasamón. F. El futuro del sector retail. *Deloitte Articles.* Recuperado de https://www2.deloitte.com/es/es/pages/consumer-business/articles/El-futuro-del-sector-Retail.html el 5/5/21.

[3] Refojos, M. El retail ya piensa en el 2021: ¿cuáles son los desafíos más urgentes? *El Periódico.com.* Recuperado de https://www.elperiodico.com/es/activos/empresas/20201118/retail-comercio-desafios-urgentes-8209155 el 10/7/21.

[4] Olivero, E. Informe: Hábitos de Consumo Mobile en España y en el Mundo en 2018, *Pickaso.* Recuperado de https://pickaso.com/2018/informe-consumo-mobile-2018 el 15/12/20.

[5] Olivero, E. 2017: El móvil se posiciona como el primer dispositivo a nivel mundial con mayor tiempo de uso, *Pickaso.* Recuperado de https://pickaso.com/2017/estudio-tiempo-uso-dispositivos-moviles el 15/12/20.

[6] Min Shum, Y. Situación Global Mobile 2020. Recuperado de https://yiminshum.com/mobile-movil-app-2020/ el 15/12/20.

[7] Escacena, J. Flutter visto con gafas de programador web. *Paradigma Digital.* Recuperado de https://www.paradigmadigital.com/dev/flutter-visto-con-gafas-programador-web/ el 17/11/20

[8] Rodrigo, G. ¿Apps híbridas o apps nativas? Un breve análisis comparativos de tecnologías móviles, *Irontec.* Recuperado de https://blog.irontec.com/apps-hibridas-vs-apps-nativas-un-breve-analisis-comparativo-de-tecnologias-moviles/ el 18/11/20 .

[9] Ruchir, C. ¿Cuáles son las ventajas del desarrollo móvil multiplataforma en la movilidad empresarial?, *Cisin.* Recuperado de https://www.cisin.com/coffee-break/es/Enterprise/what-are-the-advantages-of-cross-platform-mobile-development-in-enterprise-mobility.html el 20/5/21.

[10] Espeso, P. Las 3 tecnologías clave para el Internet de las Cosas, *Xataka.* Recuperado de https://www.xataka.com/internet-of-things/las-3-tecnologias-clave-para-el-internet-de-las-cosas el 21/1/21.

[11] Martín, M. Servicios Cloud: ¿Qué es IaaS, SaaS u PaaS?, *Profile.* Recuperado de https://profile.es/blog/servicios-cloud-que-es-iaas-saas-y-paas/ el 6/7/21.

[12] Cloud Computing: Qué es y cuáles son sus ventajas en la industria, *Edimar Electrónica.* Recuperado de https://edimar.com/cloud-computing-que-es-ventajas/ el 6/7/21.

[13] ¿Qué es Bluetooth? Toda la información sobre el estándar inalámbrico, *ionos KnowHow.* Recuperado de https://www.ionos.es/digitalguide/servidores/know-how/que-es-bluetooth/ el 6/7/21.

[14] Jiménez, J. ¿Qué gano si me compro dispositivos con Bluetooth 5.2?, *Redes Zone.* Recuperado de https://www.redeszone.net/reportajes/tecnologias/bluetooth-5-2-caracteristicas-mejoras-cambios/ el 7/7/21.

[15] Afaneh, M. Bluetooth Addresses & Privacy in Bluetooth Low Energy, *NovelBits.* Recuperado de https://www.novelbits.io/bluetooth-address-privacy-ble/ el 5/7/21.

[16] BLECTF, Capture The Flag en formato hardware basado en Bluetooth Low Energy BLE + Write-Up, *Hacker de cabecera.* Recuperado de

https://www.hackerdecabecera.com/2019/12/blectf-capture-flag-en-formato-hardware.html el 5/6/21.

[17] Pastorino, C. Cómo funciona Bluetooth Low Energy: el protocolo estrella de IoT. *WeliveSecurity,* Recuperado de https://www.welivesecurity.com/la-es/2020/03/17/como-funciona-bluetooth-low-energy/ el 21/1/21.

[18] Woolley, M. Bluetooth Technology protecting your privacy, *Bluetooth SIG.* Recuperado de https://www.bluetooth.com/blog/bluetooth-technology-protecting-your-privacy/ el 3/5/21.

[19] Dart de Google: Una introducción al lenguaje Dart, *Ionos.* Recuperado de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/lenguaje-de-programacion-dart-de-google/ el 2/7/21.

[20] Alejandro, M. Dart 2: Introducción, *Medium.* Recuperado de https://medium.com/@flutterman/dart-2-introducci%C3%B3n-ce97fd7fbca0 el 2/7/21.

[21] Diví. V. ¿Qué es el lenguaje de programación Dart?, *inLab.* Recuperado de https://inlab.fib.upc.edu/es/blog/que-es-el-lenguaje-de-programacion-dart el 2/7/21.

[22] Documentación oficial de Flutter. Recuperado de https://flutter.dev/docs/development/platform-integration/platform-channels el 2/7/21.

[23] Repositorio oficial en Github SDK de Flutter. Recuperado de https://github.com/flutter/flutter el 15/10/20.

[24] ¿Qué canal utilizo en Flutter SDK?, *it-swarm.* Recuperado de https://www.it-swarm-es.com/es/flutter/que-canal-utilizo-en-fluttersdk/805823963/ el 3/6/21.

[25] Mroczkowska, A. Skuza, B, Wlodarczyk D. Flutter vs Reactive Native. What to choose in 2021? Recuperado de https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2021#f2 el 15/6/21.

[26] Danielzczyk, J. How does Flutter work? A basic guide, *Iteo.* Recuperado de https://iteo.com/blog/post/how-does-flutter-work-a-basic-guide el 24/5/21.

[27] Backend para una aplicación Flutter, *Back4App.* Recuperado de https://blog.back4app.com/es/backend-para-aplicacion-flutter/ el 24/5/21.

[28] Katz, D. Empezando con Cloud Firestore para IOS, *Code.tuts.* Recuperado de https://code.tutsplus.com/es/tutorials/getting-started-with-cloud-firestore-for-ios--cms-30910 el 24/5/21.

[29] Toporov, E. Intellij IDEA is the base for Android Studio, the new IDE for Android Developers, *JetBrains Blog.* Recuperado de https://blog.jetbrains.com/blog/2013/05/15/intellij-idea-is-the-base-for-android-studio-the-new-ide-for-android-developers/ el 24/5/21.

[30] Ligero, R. 8 cosas que deberías saber sobre los beacons, *Accent Systems.* Recuperado de https://accent-systems.com/es/blog/8-cosas-que-deberias-saber-sobre-los-beacons/ el 17/1/21.

[31] González, D. Beacons en Android, *Ahora Somos Izertis.* Recuperado de https://ahorasomos.izertis.com/solidgear/beacons-en-android/ el 20/3/21.

[32] Arregui, A (2016). *Beacons BLE (Bluetooth Low Energy) en el sector turístico, control de afluencia y servicios de valor añadido.*(Trabajo Fin de Master). Universidad Oberta de Catalunya, España. Recuperado de http://openaccess.uoc.edu/webapps/o2/bitstream/10609/53332/8/arkaitzarregiTFM0616memoria.pdf el 27/4/21.

[33] Beacon Technology Market, *Research Dive.* Recuperado de https://www.researchdive.com/170/beacon-technology-market el 5/7/21.

[34] New BLE Beacons Coverage, *Abi Research.* Recuperado de https://www.abiresearch.com/pages/ble-beacons/ el 5/7/21.

[35] Documentación oficial CC2650 Texas Instrument. Recuperado de https://www.ti.com/product/CC2650 el 16/6/21.

[36] Documentación oficial CR2032 Panasonic Lithium-Coin. Recuperado de https://www.panasonic-batteries.com/en/specialty/lithium-coin/coin-lithium-cr2032 el 16/6/21.

[37] Documentación oficial CC2650STK Texas Instrument. Recuperado de https://www.ti.com/tool/CC2650STK el 16/6/21.

[38] Documentación oficial de Flutter. Recuperado https://flutter.dev/docs/development/packages-and-plugins/using-packages,

[39] Página oficial de Git. Recuperado de https://git-scm.com/ el 10/6/21.

[40] ¿Qué es y para qué sirve el marketing digital? *ClickAge Marketing Digital.* Recuperado de https://clickage.es/contenidos/que-es-y-para-que-sirve-el-marketing-de-proximidad/ el 18/12/20

[41] Wi-Fitracking: la nueva moda en retail y centros comerciales. *Neuromobile.* Recuperado de https://neuromobile.es/Wi-Fitracking-la-nueva-moda-en-retail-y-centros-comerciales/ el 20/12/20.

[42] BLE vs Wi-Fi: Which is better for IoT Product development? *Cabot Solutions.* Recuperado de https://www.cabotsolutions.com/ble-vs-wi-fi-which-is-better-for-iot-product-development el 20/12/20.

[43] Página oficial de BeaconStac. Recuperado de https://www.beaconstac.com/ el 2/7/21.

[44] Página oficial de BTrazing. Recuperado de https://btrazing.com/ el 2/7/21.

[45] Fujitsu muestra cuales serán las 10 tendencias para el sector retail a lo largo de 2021, *Fujitsu.* Recuperado de https://www.fujitsu.com/es/about/resources/news/press-releases/2021/01-03-2021-fujitsu-muestra-cuales-seran.html el 18/6/21.

[46] Rodríguez, C. Situación y retos 2020 Mercado de gran consumo España., *Nielsen.* Recuperado de https://www.ioncomunicacion.es/wp-content/uploads/Tendencias-consumidor-2020.pdf el 15/5/21.

[47] Soluciones de Enterprise Working: Una nueva forma de trabajar. *Axity*. Recuperado de https://www.axity.com/es/cisco/cisco-una-nueva-forma-de-trabajar/ el 17/1/21.

[48] Buscador de salarios de empleos Indeed. Recuperado de https://es.indeed.com/career/ el 4/6/21.

[49] Lanza, P. Patrón Provider en Flutter, *Dev.* Recuperado de https://dev.to/cat_yena/patron-provider-en-flutter-4hf1 el 30/5/21.

[50] Plugin FlutterBlue. Recuperado de https://pub.dev/packages/flutter_blue el 5/3/21.

[51] Página oficial de los ODS de la ONU. Recuperado de https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ el 2/6/21.

[52] Impacto de los dispositivos IoT en la sostenibilidad, *Envira IoT*. Recuperado de https://enviraiot.es/dispositivos-iot-sostenibilidad-impacto/ el 2/7/21.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

En este anexo se tratará la relación del proyecto con los Objetivos de Desarrollo Sostenible (ODS). Estos objetivos fueron aprobados por la Organización de Naciones Unidas (ONU) en septiembre de 2015, y se encuentran recogidos en la Agenda 2030 para el desarrollo sostenible. [51] Con estos objetivos se pretenden afrontar los retos y desafíos actuales (pobreza, cambio climático, igualdad, educación, etcétera), garantizando las necesidades del presente, pero sin comprometer a las futuras generaciones. El cambio empieza en las acciones del día a día.

Ilustración 69 - 5 áreas de los ODS (Agenda Euskadi 2030)

Hoy nos encontramos en la década de la acción (2021), en la que se busca "acelerar las soluciones sostenibles dirigidas a los principales desafíos del mundo". Asimismo, el impacto negativo de la pandemia en el ámbito social y económico ha acelerado todavía más la puesta en práctica de estos ODS, convirtiéndose así la Agenda 2030 en primordial para los

gobiernos. Además, la ONU ha elaborado marcos para la respuesta inmediata ante el impacto de la pandemia, todos ellos alineados con los ODS y la Agenda 2030.

Los ODS abarcan 3 esferas generales: ambiental, social y económica. Más concretamente, estos se pueden dividir en 5 áreas de prioridad: prosperidad, planeta, personas, paz y cooperación, como se muestra en la Ilustración 69.

Aunque a simple vista sea difícil relacionar un proyecto de desarrollo software con alguno de estos ODS, si se profundiza en las metas que persiguen cada uno de estos, y la motivación, justificación y aplicación de los resultados del presente trabajo, se puede concluir que hay alineación entre varios ODS y el proyecto, siendo la relación entre ellos de distinto nivel de importancia, como se recoge en la siguiente tabla:

ODS	ÁREA PRIORITARIA	ROL EN EL PROYECTO
8 – Trabajo decente y crecimiento económico	Prosperidad	Primario
9 – Industria, innovación e infraestructura	Prosperidad	Primario
11 – Ciudades y comunidades sostenibles	Prosperidad	Secundario
12 – Producción y consumo responsables	Planeta	Terciario

Tabla 4 - Relación proyecto con ODS

- **ODS 8 – TRABAJO DECENTE Y CRECIMIENTO ECONÓMICO**

  Como ya se ha explicado, el impacto socio- económico de la pandemia ha agravado todavía más la obsolescencia tecnológica existente en los establecimientos locales y minoristas, y muchos de estos han tenido que cesar su actividad debido a falta de ingresos.

La principal motivación de desarrollar este proyecto ha sido ofrecer un servicio económico y fácil de implementar para acelerar la transformación tecnológica del sector retail y hacer frente a las carencias existentes que denunciaban los clientes y/o consumidores durante su experiencia de compra y también de post-compra.

Implementar este sistema de marketing de proximidad supondrá un valor añadido en el costumer journey, que se resumirá en fidelización de la clientela actual, un aumento de clientes potenciales, mayor alcance y ventas, y en definitiva, un aumento de beneficios económicos y oferta de puestos de trabajo.

- **ODS 9 – INDUSTRIA, INNOVACIÓN E INFRAESTRUCTURA**

Si bien el marketing de proximidad ya existe desde la última década, las tecnologías utilizadas para implementar esta estrategia son innovadoras y dan respuesta a la transformación tecnológica sostenible que se persigue en este objetivo.

El uso de Cloud Computing, con sus múltiples ventajas, se resume en un uso sostenible de la infraestructura, en la que no es necesario desplegar una física propia, si no contratar servicios de proveedores. La tecnología cloud supone un progreso para la actividad empresarial en cuanto consumo de recursos como nunca visto antes, especialmente en el área de desarrollo software.

Por otro lado, la tecnología beacon es una opción ideal para las industrias sostenibles, ya que es económica, sus conexiones suponen un bajo consumo de energía (BLE), y tiene múltiples aplicaciones. Asimismo, se puede integrar esta tecnología dentro de una solución cloud, en la que se monitoricen las comunicaciones y se gestionen la información intercambiada entre dispositivos.

- **ODS 11 – CIUDADES Y COMUNIDADES SOSTENIBLES**

El avance en desarrollo sostenible gracias a la tecnología IoT + Cloud es una realidad.

Las smart cities, además de inteligentes, son seguras y sostenibles. Se utilizan estas tecnologías para dar respuesta a los desafíos de la urbanización, con soluciones de control remoto y monitoreo para dar uso eficiente de los recursos disponibles (agua o energía, como en el control del alumbrado o de las estaciones de carga de los coches eléctricos). La gestión de la información obtenida de todos los dispositivos conectados se lleva a cabo en la nube, que como ya se ha explicado, es la infraestructura estrella en cuanto uso eficiente de recursos. [52]

Como se puede observar en la siguiente ilustración, los proyectos IoT se relacionan con varios ODS, siendo el 11 uno de los más alineados con esta tecnología.

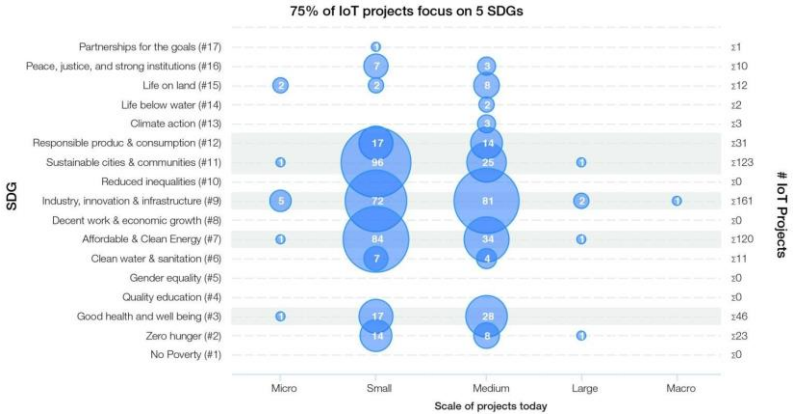


Ilustración 70 - ODS Proyectos IoT (Puentesdigitales.com)

En el caso de este proyecto, se utilizan beacons, que además de económicos, son tecnologías con bajo consumo de energía y un gran potencial, con múltiples aplicaciones, y fácil gestión en la nube. Un despliegue de beacons y recopilar la información obtenida en la nube, es una opción sostenible con un coste bajo de energía.

- **ODS 12- PRODUCCIÓN Y CONSUMO RESPONSABLES**

  El cumplimiento de objetivo es más a medio-largo plazo en comparación losanteriores de acuerdo con el estado actual del proyecto.

Para que una estrategia de marketing de proximidad tenga éxito, atrayendo a los clientes, es clave entender cómo se comportan y qué necesitan. Actualmente se está observando un cambio en el comportamiento en el consumidor, que compra o consume de forma más concienciada [46] , dando más importancia a otros valores distinto al precio o la calidad, como puede ser el impacto del medio ambiente o la responsabilidad social corporativa de la marca o empresa.

Este cambio necesariamente debe reflejarse en el contenido de las campañas de marketing. En este proyecto se ha creado un contenido estándar mediante páginas web, que sirve como ejemplo para mostrar el funcionamiento de la aplicación. Sin embargo, con el sistema desarrollado se puede administrar cualquier tipo de contenido, como promocionar productos ecológicos o lanzar encuestas personalizadas para recibir feedback de los clientes. Así, los establecimientos se aseguran de que están vendiendo lo que los clientes quieren, reduciendo el stock sobrante y finalmente transformar la cadena de producción a una más responsable.

ANEXO II: GUÍA DE INSTALACIÓN Y CONFIGURACIÓN

En este anexo se explica cómo instalar y configurar las tecnologías software utilizadas en el proyecto: Flutter, Firebase y los IDEs

I. FLUTTER

Para instalar flutter, hay que ir a la página web https://flutter.dev/docs/get-started/install. El SDK de flutter está disponible para cualquier SO (Windows, macOS, Linux, Chrome OS), si bien hay ciertos requerimientos en cuanto a sistema operativo o espacio de disco. Nos centramos en Windows 10 Home.

Se pulsa get FLUTTER SDK con la última versión estable (viene incluido el Dart SDK) para obtener la Flutter Console. Se descarga un zip que se debe extraer en un directorio que no necesite permisos de administrador para ser ejecutado.
También se puede clonar el repositorio público de Flutter SDK alojado en Github.En este caso la ruta es C:/src/flutter. Se puede buscar donde se encuentra el SDK de flutter con el comando where flutter.

```
C:\Users\teete> where flutter
C:\src\flutter\bin\flutter
C:\src\flutter\bin\flutter.bat
```

Ilustración 71 - Comando Flutter Where

Para utilizar la consola de Windows hay que actualizar la variable de entorno PATH con la carpeta bin del SDK

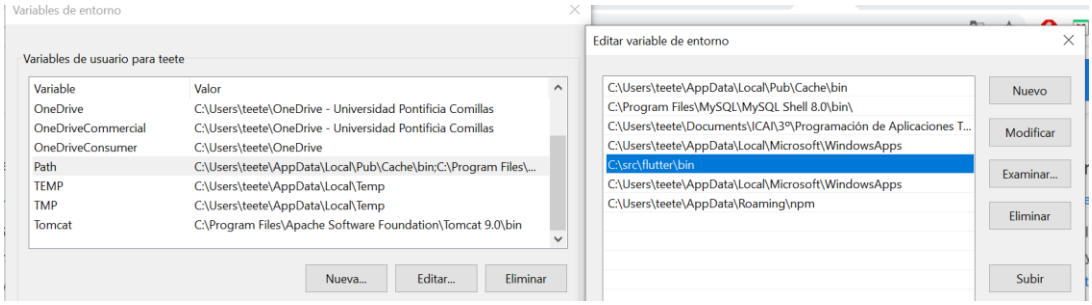


Ilustración 72 - Variable PATH

Para controlar el estado del setup (completo o incompleto, además de actualizar las dependencias), se utiliza el comando flutter doctor. Si al ejecutar una app de Flutter hay algún tipo de error, también es recomendable consultar la salida de este comando.

```
C:\Users\teete> flutter doctor
Running "flutter pub get" in flutter_tools...                    12,3s
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.2.1, on Microsoft Windows [VersiÃ³n 10.0.19041.985], locale es-AR)
[√] Android toolchain - develop for Android devices (Android SDK version 30.0.2)
[√] Chrome - develop for the web
[√] Android Studio (version 4.1.0)
[√] IntelliJ IDEA Ultimate Edition (version 2020.2)
[√] Connected device (3 available)

• No issues found!
```

Ilustración 73 - Comando Flutter Doctor

También una vez finalizado el setup, se puede comprobar en que canal (o rama del proyecto Flutter en Github) se está trabajando. Inicialmente, al descargar el SDK el canal por defecto es el estable.

```
C:\Users\teete> flutter channel
Flutter channels:
  master
  dev
  beta
* stable
```

Ilustración 74 - Comando Flutter Channel

Para cambiar de canal, por ejemplo al canal developer se utiliza fluttelr channel **canal**

```
C:\Users\teete> flutter channel dev
Switching to flutter channel 'dev'...
git: From https://github.com/flutter/flutter
git:  + f9c825981c...fa5883b78e dev                 -> origin/dev  (forced update)
git:  * [new branch]            flutter-2.3-candidate.19 -> origin/flutter-2.3-candidate.19
git:  * [new branch]            flutter-2.3-candidate.20 -> origin/flutter-2.3-candidate.20
git:  * [new branch]            flutter-2.3-candidate.21 -> origin/flutter-2.3-candidate.21
git:  * [new branch]            flutter-2.3-candidate.22 -> origin/flutter-2.3-candidate.22
git:  * [new branch]            flutter-2.3-candidate.23 -> origin/flutter-2.3-candidate.23
git:  * [new branch]            flutter-2.3-candidate.24 -> origin/flutter-2.3-candidate.24
git:  * [new branch]            kwalrath-patch-1     -> origin/kwalrath-patch-1
git:    58ac7b85d9..39546426f9  master               -> origin/master
git:  * [new branch]            revert-81813-ExpansionPanelList_elevation -> origin/revert-81813-ExpansionPanelList_elev
ation
git:  * [new branch]            revert-82570-patch-1 -> origin/revert-82570-patch-1
git:  * [new branch]            revert-84342-flutter-engine-flutter-autoroll-a31192cf-636b-427f-821f-ace70b4be488-162332
1347 -> origin/revert-84342-flutter-engine-flutter-autoroll-a31192cf-636b-427f-821f-ace70b4be488-1623321347
git:  * [new tag]               2.3.0-16.0.pre       -> 2.3.0-16.0.pre
git: Branch 'dev' set up to track remote branch 'dev' from 'origin'.
git: Switched to a new branch 'dev'
Successfully switched to flutter channel 'dev'.
To ensure that you're on the latest build from this channel, run 'flutter upgrade'
```

Ilustración 75 - Comando Futter Change Channel

Se recomienda actualizar las versiones de los SDk de Flutter y Dart, demás de la última versión de las dependencias con flutter upgrade.

```
C:\Users\teete>flutter upgrade
Checking Dart SDK version...
Downloading Dart SDK from Flutter engine 2f067fc4c57d53e565532f4e188c947fe48cd57d...
Expanding downloaded archive...
Building flutter tool...
Running pub upgrade...
Flutter is already up to date on channel dev
Flutter 2.3.0-16.0.pre • channel dev • https://github.com/flutter/flutter.git
Framework • revision fa5883b78e (3 weeks ago) • 2021-05-21 13:04:03 -0700
Engine • revision 2f067fc4c5
Tools • Dart 2.14.0 (build 2.14.0-136.0.dev)
```

Ilustración 76 - Comando Flutter Upgrade

Para habilitar flutter web se utiliza este comando.

```
C:\Users\teete> flutter config --enable-web
Setting "enable-web" value to "true".

You may need to restart any open editors for them to read new settings.
```

Ilustración 77 - Comando Enable Web

Se pueden ver los dispositivos habilitados para Flutter con flutter devices. En este caso, web está habilitado por lo que aparecen los navegadores web, además del dispositivo físico con SO Android.

```
C:\Users\teete> flutter devices
3 connected devices:

SM G970F (mobile) • RF8MB128QMF • android-arm64  • Android 11 (API 30)
Chrome (web)      • chrome      • web-javascript • Google Chrome 91.0.4472.101
Edge (web)        • edge        • web-javascript • Microsoft Edge 91.0.864.41
```

Ilustración 78 - Comando Flutter Devices

Creamos el proyecto de Flutter en el directorio que queramos con flutter create **nameproject**:

```
C:\Users\teete> flutter create proyectoflutter
Creating project proyectoflutter...
  proyectoflutter\.gitignore (created)
  proyectoflutter\.idea\libraries\Dart_SDK.xml (created)
  proyectoflutter\.idea\libraries\KotlinJavaRuntime.xml (created)
  proyectoflutter\.idea\modules.xml (created)
  proyectoflutter\.idea\runConfigurations\main_dart.xml (created)
  proyectoflutter\.idea\workspace.xml (created)
  proyectoflutter\.metadata (created)
  proyectoflutter\analysis_options.yaml (created)
  proyectoflutter\android\app\build.gradle (created)
  proyectoflutter\android\app\src\main\kotlin\com\example\proyectoflutter\MainActivity.kt (created)
  proyectoflutter\android\build.gradle (created)
  proyectoflutter\android\proyectoflutter_android.iml (created)
  proyectoflutter\android\.gitignore (created)
  proyectoflutter\android\app\src\debug\AndroidManifest.xml (created)
  proyectoflutter\android\app\src\main\AndroidManifest.xml (created)
  proyectoflutter\android\app\src\main\res\drawable\launch_background.xml (created)
  proyectoflutter\android\app\src\main\res\drawable-v21\launch_background.xml (created)
  proyectoflutter\android\app\src\main\res\mipmap-hdpi\ic_launcher.png (created)
  proyectoflutter\android\app\src\main\res\mipmap-mdpi\ic_launcher.png (created)
  proyectoflutter\android\app\src\main\res\mipmap-xhdpi\ic_launcher.png (created)
  proyectoflutter\android\app\src\main\res\mipmap-xxhdpi\ic_launcher.png (created)
  proyectoflutter\android\app\src\main\res\mipmap-xxxhdpi\ic_launcher.png (created)
  proyectoflutter\android\app\src\main\res\values\styles.xml (created)
  proyectoflutter\android\app\src\main\res\values-night\styles.xml (created)
  proyectoflutter\android\app\src\profile\AndroidManifest.xml (created)
  proyectoflutter\android\gradle\wrapper\gradle-wrapper.properties (created)
  proyectoflutter\android\gradle.properties (created)
```

Ilustración 79 - Comando Flutter Create

Para añadir un paquete al proyecto como dependencia con el cmd se utiliza flutter pub add **paquete.**

También se pueden importar paquetes manualmente desde Pub, de la siguiente forma:

1. Se añade el paquete con la última versión en la sección de dependencias del archivo pubspec.yaml. Por defecto viene añadido el paquete cupertino icons (iconos para lograr un look&feel de iOS)
2. Se actualizan las dependencias con flutter pub get
3. Se importa el paquete en el código de la forma “import:package/nombrepaquete”.

```
C:\Users\teete\Documents\ICAI\TFG\myBlueAd> flutter pub add location
  A new version of Flutter is available!

  To update to the latest version, run "flutter upgrade".


Resolving dependencies...
  async 2.6.1 (2.7.0 available)
  badges 1.2.0 (2.0.1 available)
  charcode 1.2.0 (1.3.1 available)
  cloud_firestore 1.0.7 (2.2.2 available)
  cloud_firestore_platform_interface 4.0.3 (5.1.2 available)
  cloud_firestore_web 1.0.7 (2.1.2 available)
  file 6.1.1 (6.1.2 available)
  firebase_analytics 7.1.1 (8.1.2 available)
  firebase_analytics_platform_interface 1.1.0 (2.0.1 available)
  firebase_analytics_web 0.2.0+1 (0.3.0+1 available)
  firebase_dynamic_links 0.8.0 (2.0.6 available)
  flutter_signin_button 1.1.0 (2.0.0 available)
  flutter_switch 0.2.2 (0.3.1 available)
  font_awesome_flutter 8.12.0 (9.1.0 available)
  geolocator 7.0.3 (7.1.0 available)
  geolocator_platform_interface 2.1.0 (2.1.1 available)
  gradient_widgets 0.5.2 (0.6.0 available)
  image_picker 0.7.5+4 (0.8.0+3 available)
```

Ilustración 80 - Comando Flutter Upgrade

```
name: myBlueAd
description: myBlueAd cross-platform mobile app

# The following line prevents the package from being accidentally published to
# pub.dev using `pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number used as CFBundleVersion.
# Read more about iOS versioning at
# https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html
version: 1.0.0+1

environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.1
  webview_flutter: ^2.0.4
  cloud_firestore: ^1.0.0
  firebase_core: ^1.0.0
  firebase_auth: ^1.0.0
  google_sign_in: ^5.0.0
  http: ^0.13.0
  flutter_signin_button: ^1.1.0
  url_launcher: ^6.0.2
  sms_autofill: ^1.2.7
  uni_links: ^0.4.0
  firebase_dynamic_links: ^0.8.0
  provider: ^4.3.3
  gradient_widgets: ^0.5.2
  flutter_switch: ^0.2.0
  flutter_login_facebook: ^1.0.1
```

Ilustración 81 - Ejemplo Pubspec.yaml

Para lanzar la aplicación, se utililiza el comando flutter run desde el directorio raíz del proyecto. Este ejecuta el archivo donde se encuentre el método main(), por

defecto en lib/main.dart. En el caso de este proyecto, se ejecuta modo debug con un dispositivo Android físico.

```
C:\Users\teete\Documents\ICAI\TFG\myBlueAd> flutter run
Launching lib\main.dart on SM G970F in debug mode...
Running Gradle task 'assembleDebug'...                             63,4s
√  Built build\app\outputs\flutter-apk\app-debug.apk.
Installing build\app\outputs\flutter-apk\app.apk...                 5,5s
I/FlutterBluePlugin(22676): setup
D/FlutterLocationService(22676): Creating service.
D/FlutterLocationService(22676): Binding to location service.
Syncing files to device SM G970F...                                357ms

Flutter run key commands.
r Hot reload.
R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).

Running with unsound null safety
For more information see https://dart.dev/null-safety/unsound-null-safety

An Observatory debugger and profiler on SM G970F is available at: http://127.0.0.1:50951/PckUiUAm_-o=/
The Flutter DevTools debugger and profiler on SM G970F is available at:
http://127.0.0.1:9101?uri=http%3A%2F%2F127.0.0.1%3A50951%2FPckUiUAm_-o%3D%2F
I/SurfaceView(22676): onWindowVisibilityChanged(4) false io.flutter.embedding.android.FlutterSurfaceView{7f218a9
```

Ilustración 82 - Comando Flutter Run

Ya que utilizar el CMD para desarrollar un proyecto es una tarea inviable, Flutter recomienda utilizar los IDEs Android Studio o Intellij IDEA. Ambos son compatibles con plugins de Flutter y Dart. Al instalar estos plugins, se puede crear un proyecto de Flutter con la interfaz de estos IDEs en vez de con la consola.

Todos estos IDEs tienen terminal integrado, por lo que todos los comandos de Flutter Console se pueden realizar dentro de los mismos. Tiene vital importancia flutter pub get para actualizar las dependencias del proyecto (paquetes de terceros, assets), las cuales se deben listar en el archivo pubspec.yaml alojado dentro del proyecto.

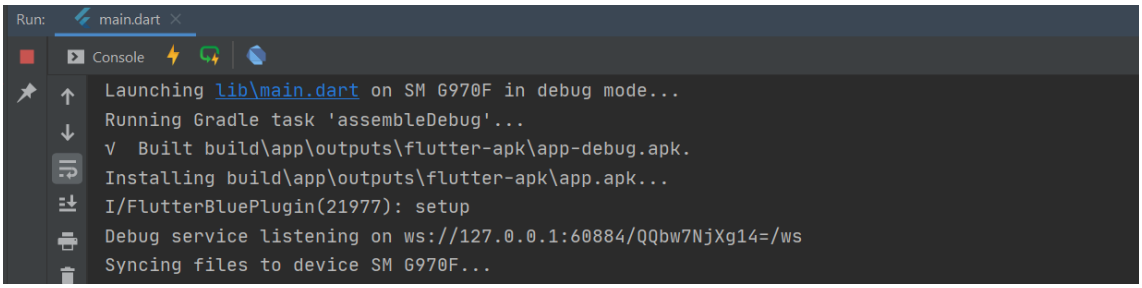


Ilustración 83 - Ejemplo Futter Run Debug

II. FIREBASE

Para utilizar la consola de desarrollador de Firebase, es necesario iniciar sesión con una cuenta Gmail (Google). Una vez se accede, la primera pantalla que se muestra en la consola es una lista de proyectos Firebase existentes, y la opción de agregar un nuevo proyecto.

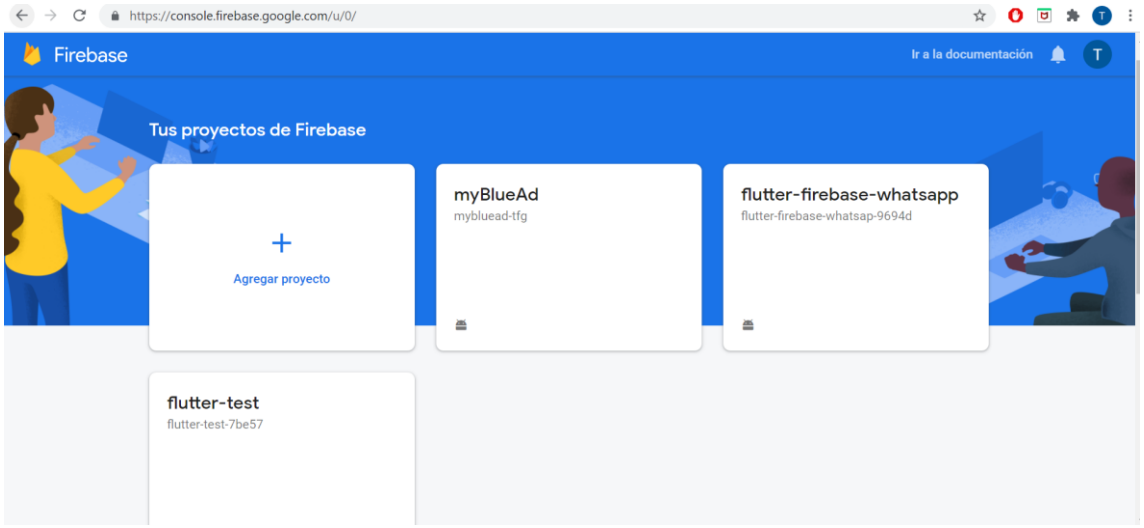


Ilustración 84 - Consola Firebase Proyectos

Como se ha explicado en la memoria, Firebase permite despegar aplicaciones tanto para Android, iOS, Web y Unity, con un límite de 30 apps de cualquiera de estos tipos por proyecto, según la documentación oficial de Firebase.

En este proyecto por cuestiones de tiempo y compatibilidad entre los SO de los dispositivos físicos utilizados se ha desplegado únicamente una aplicación Android en un proyecto Firebase a partir de la app desarrollada con el framework Flutter. Sin embargo, ya que Flutter es un framework multiplataforma, con el mismo código y las configuraciones de Firebase para iOS pertinentes, se podría agregar una app para iOS en este proyecto.

A continuación se explica cómo crear un proyecto y configurar una app Android para el mismo.

Cuando se pulsa en agregar proyecto, es necesario ponerle un nombre al mismo.

También se puede habilitar Google Analytics en este momento, herramienta muy útil para monitorear el rendimiento de la app, detectar fallas, realizar seguimiento del usuario e informes detallados, entre otras funciones.

En la consola se muestran todos los servicios y extensiones que Firebase ofrece a los desarrolladores, divididos por categorías.

Como se ha explicado en la memoria, Firebase actualmente tiene dos planes de facturación, el Spark (gratis) y el Blaze (pago por uso), y desde la consola se puede modificar cuando se quiera.

Una vez creado el proyecto, ya se puede agregar cualquiera de las apps disponibles en Firebase. Nos centramos en la configuración de una app Android:

Paso 1: Registrar la app

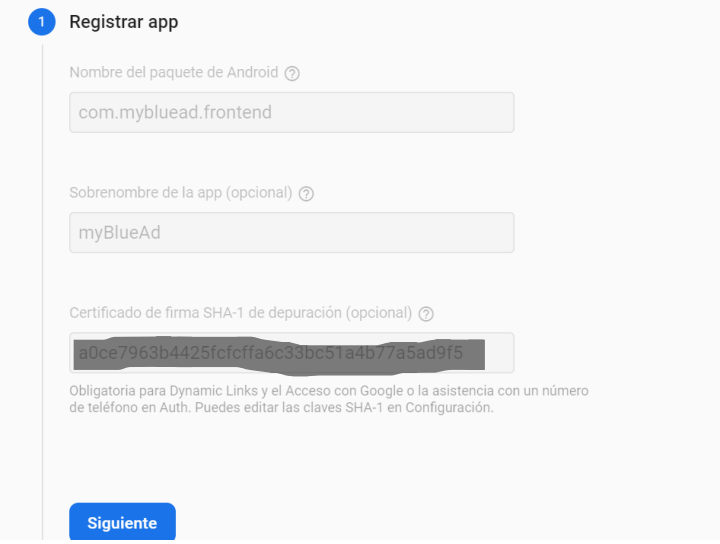


Ilustración 85 - Registrar App Android Firebase

El paquete de Android se encuentra en el archivo AndroidManifest.xml, en la ruta Android/app/src/main/AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.mybluead.frontend">
    <application
            android:label="frontend"
            android:icon="@mipmap/ic_launcher">
        <activity
```

Ilustración 86 - Paquete Android

La firma SHA-1 o SHA-256 es obligatoria para acceder con Google, la autenticación vía número de teléfono o con Dynamic links. Se pueden editar en cualquier momento en configuración.

Para obtener la firma SHA-1 o SHA-256 de la aplicación Android, desde el terminal hay que introducir el siguiente comando:

```
keytool -list -v -alias androiddebugkey -keystore
%USERPROFILE%\.android\debug.keystore
```

Codigo 21 - Obtener SHA

La contraseña es Android. A continuación, aparecerán ambos SHAs, que se han firmado con el algoritmo RSA 2048-bit.

Paso 2: Descargar archivo de configuración

Google-services.json es un archivo donde se encuentra información necesaria para que el proyecto tenga acceso a los servicios de Google, y en consecuencia a los de Firebase. Sin este archivo, el proyecto de Firebase no "encuentra" la app Android que se quiere añadir al proyecto. En este archivo se encuentra la información del proyecto, claves de autenticación del cliente, claves de acceso a la API y a los servicios. Es un archivo privado de cada proyecto con información sensible, que en el caso de alojar el proyecto en un lugar público como un repositorio público Github, debe añadirse al archivo gitignore y así no subirse al repositorio.

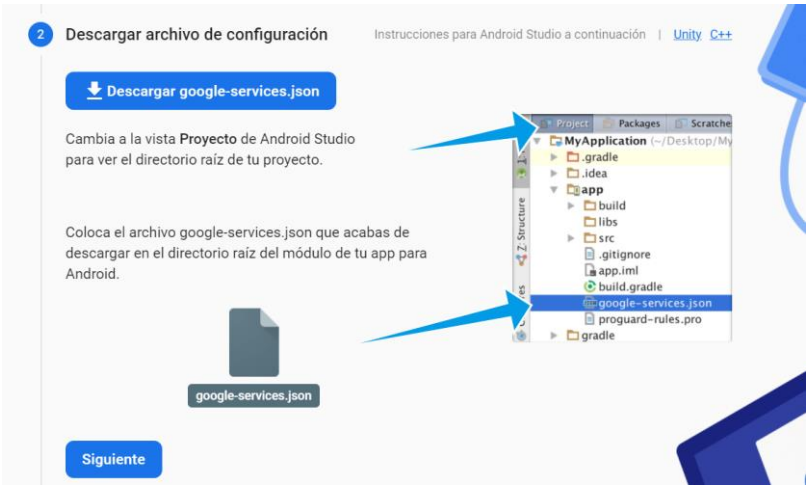


Ilustración 87 - Descarga archivo configuración

La seguridad de los proyectos alojados en alguno de los servicios web en la nube dentro de Google Cloud Platform (en este caso Firebase) es tal que si la plataforma detecta que las credenciales han sido publicadas en algún sitio web público (como el caso de Github mencionado), envia un aviso al correo electrónico de asistencia de la configuración con los pasos a seguir para que el administrador proteja sus cuentas y el proyecto:

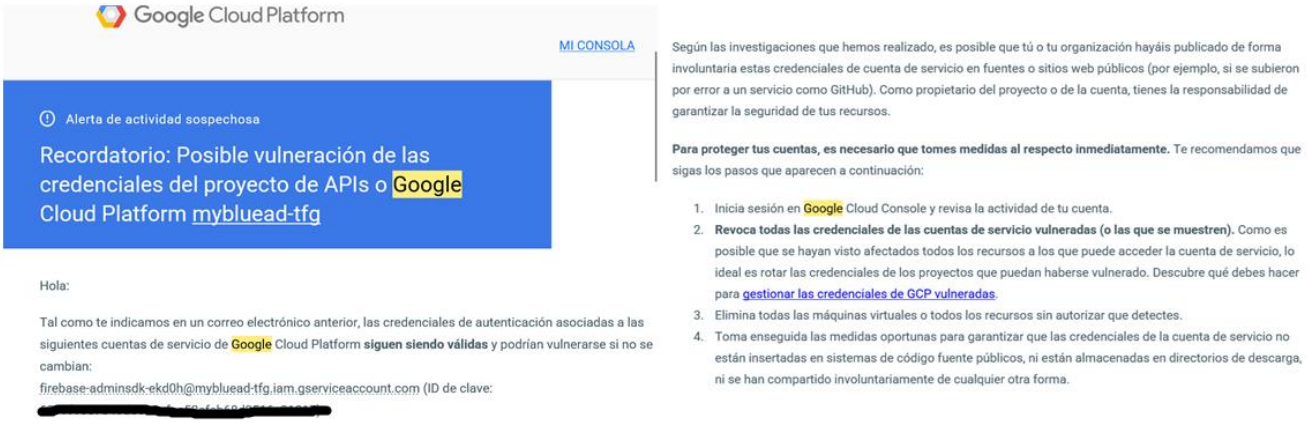


Ilustración 88 - Aviso CGP

Paso 3: Agregar SDK Firebase

En el caso de nuestro proyecto, utilizamos el lenguaje Kotlin para desarrollo en Android. Se habilitan los servicios de Google para cargar el archivo de configuración en el archivo build.gradle,

Ilustración 89 - Agregar SDK Firebase

III. ANDROID STUDIO

Este IDE es gratuito y se puede descargar la versión del ejecutable desde https://developer.android.com/studio.

IV. INTELLIJ IDEA

Intellij IDEA se ofrece en versión gratuita (Community) y comercial (Ultimate), con distintas prestaciones.

Para instalar cualquier versión de Intellij IDEA, se puede instalar de forma manual, con la versión compatible con el sistema operativo correspondiente, o mediante la

herramienta Toolbox, https://www.jetbrains.com/help/idea/installation-guide.html#toolbox con la que una vez se crea una cuenta JetBrains se pueden consultar los productos instalados o comprados.

Una vez instalado el IDE, es necesario descargar los plugins de Flutter y Dart del Marketplace para crear un proyecto Flutter, así como añadir el Path donde se encuentra el SDK.

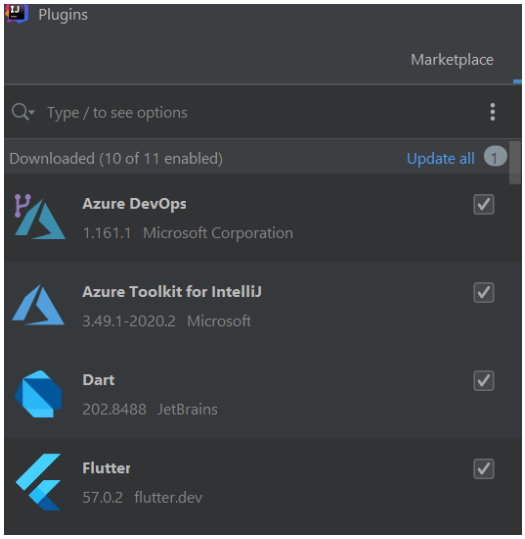


Ilustración 90 - Plugins

Si se tiene alguna licencia, como es en este caso, es necesario vincularla al IDE. Para ello, hay que ir Help > Register y rellenar el formulario.

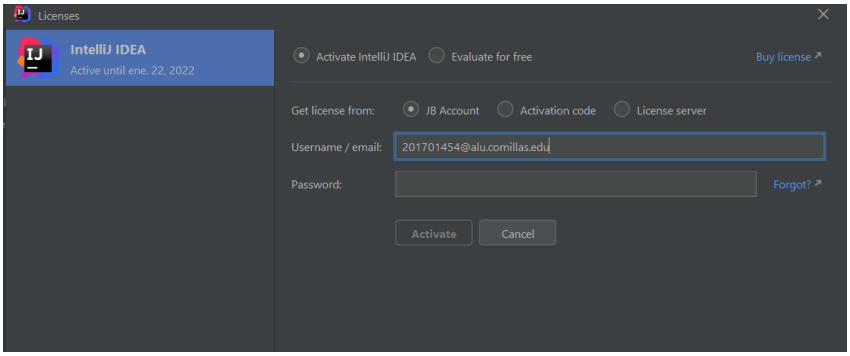


Ilustración 91 - Añadir Licencia

Se pueden importar proyectos de repositorios de Github, y subirlos desde el IDE habilitando alguna herramienta de control de versiones desde Settings -> Version Control -> Añadir Directorio. Será necesario habilitar un token de acceso desde dicha herramienta.

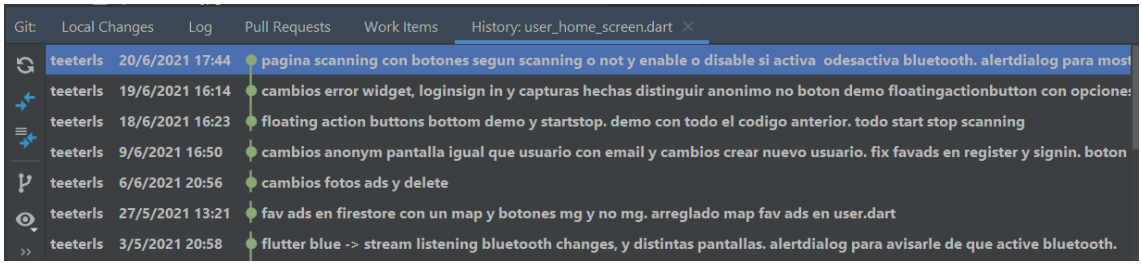


Ilustración 92 - Ejemplo Git en IDE

V. SUBLIME TEXT

Para instalar este editor de texto, únicamente hay que ir a la página https://www.sublimetext.com/ y descargar la versión del ejecutable