Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Diseño y desarrollo de servicios basados en beacons para el campus universitario del Baix Llobregat

TITULACIÓN: Grado en Ingeniería Telemática

AUTOR: Marc Cabezas Alarcón

DIRECTOR: Rafael Vidal Ferré

FECHA: 27 de octubre del 2017

Títol: Disseny i desenvolupament de serveis basats en beacons pel Campus universitari del Baix Llobregat.

Autor: Marc Cabezas Alarcón

Director: Rafael Vidal Ferré

Data: 27 d'octubre del 2017

Resum

Els *beacons* són dispositius de petites dimensions i gran autonomia que es basen en la tecnologia *Bluetooth Low Energy* per a difondre, en un radi d'abast de fins a uns 100 metres, un identificador únic i una sèrie de paràmetres o dades. Aquella informació, si es rebuda per un *smartphone* compatible, permet oferir serveis de diferent índole a qualsevol tipus de públic.

L'objectiu d'aquest treball és comprovar la viabilitat d'integrar, en una aplicació mòbil, serveis basats en *beacons* útils per a la comunitat educativa del Campus del Baix Llobregat. També es pretén documentar la tecnologia en la que es basen aquests dispositius, tant el seu *hardware* com el seu *software*, per entendre millor les seves limitacions i les oportunitats que poden oferir en el nostre escenari d'aplicació i en d'altres.

Com a resultat d'aquest treball, s'han proposat una sèrie de serveis que s'han implementat en una aplicació disponible a *Google Play*. L'aplicació, anomenada *beaconsUPC*, s'ha desenvolupat amb el *framework* d'Ionic, el que ens ha permès programar-la amb tecnologies Web (*Typescript, Sass* y *HTML*) i, poder així, convertir-la en una *app* per a Android i iOS. A més, l'aplicació s'ha validat a partir del desplegament de *beacons* realitzat en alguns edificis del Campus.

La dificultat d'utilitzar *beacons* recau en saber oferir uns serveis que resultin d'utilitat per als clients. Si s'aconsegueix aquesta conjunció, s'haurà aconseguit el que moltes empreses desitgen: que els seus clients tinguin una aplicació mòbil instal·lada als seus *smartphones* amb la que disposar d'un canal de comunicació directe per a millorar la seva imatge de marca i, també, treure rendibilitat econòmica, per exemple, mitjançant l'enviament de publicitat.

Título: Diseño y desarrollo de servicios basados en beacons para el campus universitario del Baix Llobregat

Autor: Marc Cabezas Alarcón

Director: Rafael Vidal Ferré

Fecha: 27 de octubre del 2017

Resumen

Los *beacons* son dispositivos de pequeñas dimensiones y gran autonomía que se basan en la tecnología *Bluetooth Low Energy* para difundir, en un radio de alcance de hasta 100 metros, un identificador único y una serie de parámetros o datos. Esa información, si es recibida por un s*martphone* compatible, permite ofrecer servicios de distinta índole a cualquier tipo de público.

El objetivo de este trabajo es comprobar la viabilidad de integrar, en una aplicación móvil, servicios basados en *beacons* útiles para la comunidad educativa del Campus del Baix Llobregat. También se pretende documentar la tecnología en la que se basan estos dispositivos, tanto su *hardware* como su *software*, para entender mejor sus limitaciones y las oportunidades que pueden ofrecer en nuestro escenario de aplicación y en otros.

Como resultado de este trabajo, se han propuesto una serie de servicios que se han implementado en una aplicación disponible en *Google Play*. La aplicación, llamada *beaconsUPC*, se ha desarrollado con el *framework* Ionic, lo que nos ha permitido programarla con tecnologías Web (*Typescript, Sass* y *HTML*) y, poder así, convertirla en una *app* para Android y iOS. Además, la aplicación se ha validado a partir del despliegue de *beacons* realizado en algunos edificios del Campus.

La dificultad de utilizar *beacons* recae en saber ofrecer unos servicios que resulten de utilidad para los clientes. Si se consigue esa conjunción, se habrá conseguido lo que muchas empresas desean: que sus clientes tengan una aplicación móvil instalada en sus *smartphones* con la que disponer de un canal de comunicación directo para mejorar su imagen de marca y, también, sacar rentabilidad económica, por ejemplo, mediante el envío de publicidad.

Title: Design and development of services based on beacons for the Baix Llobregat university

Author: Marc Cabezas Alarcón

Director: Rafael Vidal Ferré

Date: 27th October 2017

Overview

Beacons are devices of small dimensions and great autonomy that are based on the Bluetooth Low Energy technology to diffuse, within a range of up to 100 meters, a unique identifier and a range of parameters or data. If it is received by a compatible smartphone, this information allows to offer different services to any type of public.

The objective of this paper is to verify the feasibility of integrating, in a mobile application, useful services based on beacons for the educational community of the Baix Llobregat Campus. Also, we pretend to document the technology on which these devices are based, in terms of hardware and software, in order to understand better their limitations and the opportunities they can offer in our application scenario among others.

As a result of this paper, a number of services have been proposed and implemented in an application available on Google Play. The application, called beaconsUPC, has been developed with the Ionic framework, which has allowed us to program it with Web technologies (Typescript, Sass and HTML) and, thanks to that, an Android and iOS app has been made. In addition, the application has been validated from the attachment of *beacons* in some buildings of the Campus.

The difficulty of using beacons lies in knowing what services are useful to customers. If that combination is achieved, many companies will have succeeded since, having a mobile application installed in customers' smartphones, allows such companies to have a direct communication. Consequently, they can improve their brand image and also obtain economic profitability, for example, by sending advertisings.

Agradecimientos

Pese a haber sido un trabajo duro en el que he tenido que sacar lo mejor de mí mismo y afrontar dificultades, he tenido la suerte de no recorrer el camino en soledad. Mucha gente ha aportado su granito de arena y, juntos, hemos llegado a la meta.

En primer lugar, me gustaría agradecer a familiares y amigos que han estado ayudándome a probar la aplicación, actualización tras actualización. Agradecer a mi padre y a mi amiga Irene la ayuda que me han brindado leyendo y corrigiendo toda la memoria, a Melany por haber pasado a mi lado los meses de verano encerrados en casa haciendo nuestros respectivos TFGs y a Emma por acompañarme y ayudarme a colocar los *beacons* en la facultad. También quiero dar las gracias a mi prima Yaiza por haberme diseñado el logo y los marcadores del mapa de mi aplicación.

Por otro lado, agradecer a mi tutor Rafael Vidal su plena confianza puesta en mí a la hora de apostar por mi idea de trabajo, por aceptar mis pautas y, también, por aportar siempre que lo he necesitado esa segunda opinión que me ha ayudado a resolver las dificultades.

Gracias a Imma Losada Martín, CEO de Namastech, por darme la oportunidad de trabajar en su empresa durante esos pocos pero intensos meses de verano. Gracias a ese trabajo he adquirido muchos conocimientos de programación que me han permitido desarrollar una aplicación muy por encima de mis expectativas iniciales. Gracias también, por publicarla en *Play Store*.

Gracias a David De Celis, *Bussiness Development* de Accent Systems, por cederme los 5 beacons iBKS de manera gratuita y desinteresada, gracias a ellos he podido abrir un mayor abanico de servicios a integrar en mi aplicación.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ACRÓNIMOS

API	*Application Programming Interface (Interfaz de programación de apps)*
BER	*Bit Error Rate (Tasa de error binario)*
BLE	*Bluetooth Low Energy (Bluetooth de baja energía)*
CLI	*Command Line Interface (Interfaz de línea de comandos)*
CPU	*Central Processing Unit (Unidad central de procesamiento)*
CRC	*Cyclick Redundancy Check (Verificación por redundancia cíclica)*
EID	*Ephemeral Identifier (Identificador efímero)*
GATT	*Generic Attribute Profile (Perfil genérico del atributo)*
GFSK	*Gaussian Frequency Shift Keying (modulación por desplazamiento de frecuencia gausiana)*
HTML	*HyperText Markup Language (Lenguaje de marcado de hipertexto)*
MFG ID	*Manufacturer's Identifier (Identificador de la empresa del fabricante)*
MIC	*Message Integrity Check (Comprobación de integridad de mensajes)*
IDE	*Integrated Development Environment (Entorno de desarrollo integrado)*
iOS	*iPhone Operative System (Sistema operativo de iPhone)*
ISM	*Industrial Scientific and Medical (Bandas de rediofrecuencia reservadas)*
NFC	*Near field communication (Comunicación de campo cercano)*
PAN	*Personal Area Network (Red de área personal)*
PDU	*Protocol Data Unit (Unidad de datos de protocolo)*
RAM	*Random Access Memory (Memoria de acceso aleatorio)*
RFU	*Reserved for Future Uses (Reservado para usos futuros)*
RSSI	*Received Signal Strength Indication (Indicador fuerza señal recibida)*
Sass	*Syntactically Awesome StyleSheets (Hojas de estilo sintácticamente impresionantes)*
SDK	*Software Development Kit (Kit de desarrollo de software)*
SIG	*Bluetooth Special Interest Group (Grupo especial interés en Bluetooth)*
URL	*Uniform Resource Locator (Localizador Uniforme de Recursos)*
UUID	*Universally Unique Identifier (Identificador Único Universal)*
TLM	*Telemetry (Telemetría)*
Wi-Fi	*Wireless Fidelity (Fidelidad inalámbrica)*
WWDC	*Worldwide Developers Conference (Conf. Mundial de Desarrolladores)*
W3C	*World Wide Web Consortium (Consorcio mundial de la web)*

INTRODUCCIÓN

Los *beacons* son dispositivos de pequeñas dimensiones y gran autonomía que se basan en la tecnología *Bluetooth Low Energy* para difundir, en un radio de alcance de hasta unos 70 metros, un identificador único y una serie de parámetros o datos. Esa información, si es recibida por un *smartphone* compatible, permite ofrecer servicios de distinta índole a cualquier tipo de público. Un ejemplo típico de servicios con *beacons* se da en el comercio, permitiendo hacer ofertas segmentadas y enfocadas al tipo de usuario que entra al establecimiento o la zona por las que se está moviendo.

En este trabajo se ha propuesto, utilizando estos dispositivos, diseñar y programar una aplicación móvil que ofreciera diferentes servicios de utilidad para la comunidad educativa del Campus del Baix Llobregat de la UPC. Otro objetivo previo ha sido el estudio y documentación de la tecnología utilizada por estos dispositivos, *hardware* y *software*, para entender de manera precisa su funcionamiento y determinar la viabilidad de apostar por ella, tanto para nuestro caso de uso como para otros.

En el primer capítulo de esta memoria, se definen los componentes físicos y el *firmware* de que se componen los *beacons,* la tecnología BLE en la que se fundamentan y los protocolos típicos que pueden implementar. También, se dan unas pinceladas de su funcionamiento y se enumeran las aplicaciones y usos más comunes para los que se utilizan estos dispositivos.

En el segundo capítulo, se presentan los dos tipos de *beacons* con los que hemos trabajado y sus características, diferencias y funcionalidades extra que aportan. Todas ellas se pueden consultar en el anexo 2 con capturas de pantalla de las aplicaciones propietarias de configuración de sus fabricantes.

En el tercer capítulo, se describe el proceso de desarrollo de la aplicación fruto de este trabajo, denominada *beaconsUPC*. Primero se razona el tipo de aplicación y el *framework* elegidos para su desarrollo. También se definen los servicios y funcionalidades que se han implementado en la aplicación y cómo se ha estructurado ésta. Para información relativa a los *plugins* utilizados o la estructuración de las bases de datos, ver el anexo 1. En el anexo 3, se incluyen guías de procesos que hemos tenido que realizar a lo largo del trabajo, como configurar el envío de notificaciones *push* o *nearby* desde las respectivas plataformas web, entre otras. El código de la aplicación puede consultarse en GitHub [1].

En el cuarto capítulo, se explica la configuración final que hemos aplicado a nuestros *beacons* y su despliegue en los edificios del Campus. En el anexo 4, se muestran algunas fotografías relativas a este despliegue.

Finalmente, se hace una pequeña reflexión sobre la consecución de los objetivos de este trabajo y la posible evolución de esta tecnología a corto plazo. Se comentan, asimismo, algunos aspectos a mejorar y desarrollar de nuestra aplicación a modo descriptivo de posibles actualizaciones futuras.

CAPÍTULO 1. BEACONS

Un *beacon*, baliza en castellano, es un dispositivo de pequeñas dimensiones que emite una señal de radio de corto alcance por medio de la tecnología Bluetooth. Esta señal, llamada señal de *advertising* o señal publicitaria, puede ser captada por dispositivos inteligentes hasta unos 100 metros de distancia.

La forma con la que estos *beacons* pueden interactuar con el usuario habitualmente es mediante una aplicación instalada en un Smartphone o tableta, aunque hay otros dispositivos inteligentes que también podrían detectarlos como por ejemplo los *smartwhatchs*.

Existen multitud de proveedores de *beacons* que implementan diferentes configuraciones en función de sus posibles usos finales pero, en general, todos se basan en unos mismos componentes y tecnologías. A continuación podemos ver algunos de los *beacons* que hay actualmente en el mercado:

Fig. 1.1 Algunos *beacons* en el mercado actualmente. Fuente Aislelabs [2]

En este capítulo veremos de qué partes se componen los *beacons* y en qué tipo de tecnología se basan. También analizaremos los protocolos más utilizados para configurar e interactuar con ellos y cuál es el funcionamiento típico que emplean. Posteriormente veremos algunas de las aplicaciones para las que están pensados los *beacons* y algunos ejemplos de implementaciones ya desplegadas y en funcionamiento en la actualidad.

1.1. Componentes

Todos los *beacons* tienen componentes de hardware y firmware bastante similares entre sí, independientemente del proveedor que lo fabrique. En este apartado analizaremos los componentes físicos principales que constituyen un *beacon* y las configuraciones que suelen incorporar para dotarlos de ciertas funcionalidades.

1.1.1. Hardware

El *hardware* de un *beacon* consiste básicamente en una batería y un *chipset* que monta un microcontrolador con un chip de radio BLE. Dependiendo del proveedor del *beacon* y las funcionalidades que se quieran obtener, se puede optar por añadir otros elementos al chipset como acelerómetros, sensores de temperatura e incluso chips NFC, entre otros.

1.1.1.1. Carcasa

Generalmente el recubrimiento del chipset y la batería de un *beacon* están hechos de plástico o silicona y suelen ser resistentes al agua y al polvo. Este tipo de carcasa protectora no debe interferir en las emisiones de radio del chip de BLE y suelen tener una abertura o mecanismo de acceso a la batería para poder reemplazarla una vez se haya agotado.

1.1.1.2. Chipset

En el chipset de un *beacon* encontramos el microcontrolador y el resto de sensores que haya decidido implementar el fabricante. El microcontrolador con chip de radio BLE está fabricado principalmente por dos grandes empresas y tienen las siguientes características:

Tabla 1.1. Fabricantes de microcontroladores para *beacons*.

	Modelo	Procesador	ROM y RAM	Sensores integrados	Versión bluetooth
Texas Instruments	CC2541 [3]	8-bit 8051 CPU Core	128kB/256kB flash + 8kB RAM		4.0 LE
Nordic Semiconductor	nRF51822 [4]	2-bit ARM® Cortex™ M0 CPU	128kB/256kB flash + 32kB/16kB RAM	Temperatura	4.0 LE

La gran mayoría de *beacons* que encontramos en el mercado cuentan con uno de estos dos modelos, aunque ya se han empezado a desarrollar otros con la compatibilidad de Bluetooth 5 [5].

1.1.1.3. Batería

La duración de la batería de un *beacon* puede oscilar de unos pocos meses a unos cuantos años dependiendo del tipo de batería que se utiliza y otros parámetros que veremos en el siguiente apartado.

Típicamente se suelen utilizar pilas de botón o pilas cilíndricas AA o AAA aunque algunos podrían incluso ir conectados a tomas de corriente o puertos USB. La mayoría de *beacons* utilizan pilas para ser más pequeños y poder colocarse en cualquier lugar.

1.1.2. Firmware

Cada proveedor dota a sus *beacons* de un *firmware* específico que aporta toda la lógica de bajo nivel para hacer funcionar su hardware y así controlar diferentes parámetros del dispositivo.

El *firmware* de un *beacon* permite controlar básicamente su potencia de transmisión y su intervalo de emisión de *advertisings*. Una correcta configuración de estos parámetros puede hacer alargar la duración de la batería.

Todos estos parámetros suelen ser editables por el administrador de los *beacons* a través de aplicaciones o programas desarrollados por los mismos fabricantes.

1.1.2.1. Potencia de transmisión

Los *beacons* transmiten su señal de *advertising* a una potencia fija pero configurable según los requerimientos de la aplicación a la que están destinados. La señal pierde intensidad a medida que ésta se aleja del *beacon* debido a la atenuación por la distancia y los otros posibles obstáculos que encuentre. También puede verse afectada por las interferencias del entorno u otras tecnologías que trabajen en la misma frecuencia de emisión.

1.1.2.2. Intervalo de emisión

Otro parámetro configurable por el usuario es el intervalo de emisión de la señal de *advertising*. Típicamente los *beacons* emiten en un intervalo de 100ms, es decir, 10 veces cada segundo. Los fabricantes permiten configurar estos intervalos desde cada 100ms a cada 10s, dejando al administrador la libertad de elegir el valor que más le convenga dentro de esa franja.

El intervalo de emisión elegido proporcionará una capacidad de respuesta del dispositivo receptor mayor o menor dependiendo de si el intervalo es más corto o más largo. Esa elección afectará directamente a la duración de las baterías haciendo que duren más cuanto más amplio sea el intervalo entre emisiones.

1.1.2.3. Duración de la batería

La duración de la batería de un *beacon* no es un parámetro configurable directamente por el usuario, sino que sería el resultado de la elección de la potencia de transmisión y el intervalo de emisión de *advertisings*.

A mayor potencia de transmisión y/o menor sea el intervalo de emisión, mayor será el consumo de batería del *beacon* y antes la tendremos que cambiar.

1.2. Tecnología Bluetooth Low Energy

Bluetooth es un protocolo de tecnología inalámbrica para intercambiar datos entre diferentes dispositivos a cortas distancias. Se utiliza en redes inalámbricas de área personal (PAN, *Personal Area Network*) utilizando un enlace de radiofrecuencia en la banda de 2,4GHz.

Este protocolo está definido por una asociación privada, sin ánimo de lucro, llamada SIG [6] (*Bluetooth Special Interest Group*) que básicamente engloba a todos los fabricantes que implementan su tecnología.

Recientemente ha salido una nueva especificación, Bluetooth v5 [5], con la que, entre otras cosas, se va a tener hasta cuatro veces más alcance, el doble de velocidad de transmisión de datos y un ancho de banda ocho veces mayor al que tenemos actualmente en la versión 4.2. Pese a todas esas posibles ventajas, aún no hay muchos fabricantes de *beacons* ni de *smartphones* que integren en sus dispositivos esa versión. Pasarán como mínimo un par de años para que esta versión esté implementada en la mayoría de los dispositivos.

En el año 2010, se completó la especificación de Bluetooth 4.0, que trajo consigo dos nuevos modos de Bluetooth: Bluetooth de alta velocidad y Bluetooth de bajo consumo (BLE, *Bluetooth Low Energy*). Este último es en el que se basan los *beacons* y, está diseñado para la trasmisión de pequeñas cantidades de datos con muy poco gasto de energía.

La topología de red que utiliza es de tipo estrella, es decir, los dispositivos centrales (llamados *master*, maestro en castellano) pueden conectarse a varios dispositivos periféricos (llamados *slaves*, esclavos en castellano), sin embargo, éstos solo pueden estar conectados a un único *master*.

La asociación SIG [6] define una pila de protocolos para la conexión y gestión de los dispositivos, dividida fundamentalmente en tres partes: controlador (*Controller*), anfitrión (*Host*) y aplicación (*Application*). El *controller* contiene la capa física y la capa de enlace del dispositivo físico, lo que permite transmitir y recibir señales de radio. El *Host* es la pila de *software* que administra cómo los dispositivos *bluetooth* se comunican entre sí. Esta parte del protocolo cuenta con una serie de elementos que cada sistema operativo o entorno puede adaptar en función de sus preferencias. Por último, la capa *Application* es la que diseña cada fabricante de servicios *bluetooth* para que interactúe con las capas inferiores.

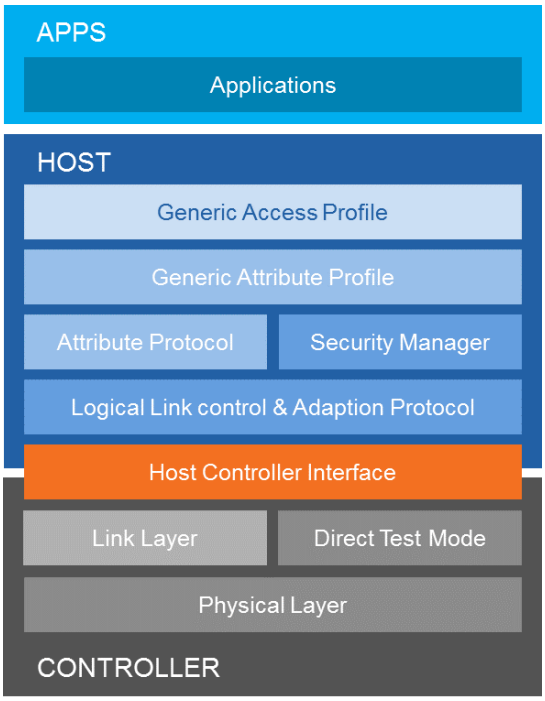


Fig. 1.2 Pila del protocolo *bluetooth*. Imagen oficial del SIG [7]

Para el caso que nos ocupa de los *beacons* sólo vamos a necesitar entender las capas del *controller*, puesto que estos dispositivos no establecen ninguna conexión directa con ningún dispositivo *master* o *slave* para intercambiar datos, simplemente emiten paquetes *advertising* en modo *broadcast*, como veremos más adelante.

Los *beacons* de la mayoría de fabricantes llevan configurado en su *firmware* todas las capas de la parte del *Host* para permitir que los administradores puedan conectarse a los *beacons* a través de una aplicación prioritaria o alguna genérica y así poder configurar los parámetros de éste.

Toda la información descrita en este apartado ha sido obtenida de la especificación principal de Bluetooth, concretamente la de la versión 4.2 [8]. La información relacionada con BLE se define en el Vol. 6 de dicha especificación y define en gran detalle todos los parámetros o funcionalidades de esta tecnología.

1.2.1. Capa física

La tecnología de BLE utiliza la banda de frecuencias ISM de 2,4GHz y una modulación GFSK a 1Mbps. Cuenta con un total de 40 canales de radiofrecuencia con una separación entre ellos de 2MHz. Tres de estos canales se utilizan únicamente para enviar mensajes de *advertising* (CH37, CH38, CH39) y el resto sirven para transmisión de datos (CH0 – CH36). En la siguiente imagen podemos ver mejor la distribución de estos canales en comparación con la de la tecnología Wi-Fi.

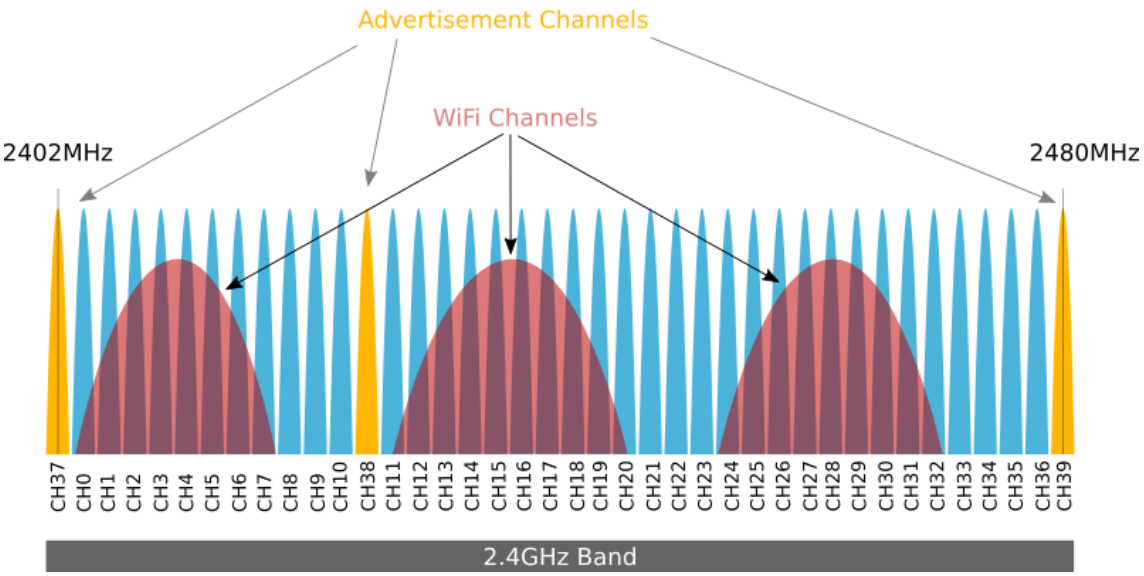


Fig. 1.3 Canales BLE y Wi-Fi en la banda de 2,4GHz. Imagen de Argenox [9]

Podemos observar como los tres canales publicitarios (*advertising*) están definidos en unas frecuencias concretas para evitar posibles interferencias con otras tecnologías que coexisten en la misma banda frecuencial como puede ser Wi-Fi (IEEE 802.11).

La sensibilidad de un receptor en BLE se define como el nivel de señal para el cual se alcanza una tasa de errores de bits (BER) de 10^{-3}. La especificación de BLE requiere una sensibilidad mayor o igual a -70dBm.

Un transmisor, en cambio, puede tener una potencia de salida comprendida entre 0,01mW (-20dBm) y 10mW (+10dBm). Esta potencia puede ser modificada para optimizar el consumo de batería o reducir interferencias con otros equipos cercanos.

La mayoría de fabricantes de *beacons* que hay actualmente en el mercado permiten establecer configuraciones de potencia de transmisión que varían de los -40dBm a los +4dBm. Esas potencias equivalen aproximadamente a distancias de transmisión que van de los pocos centímetros a decenas de metros.

1.2.2. Capa de enlace

Esta capa es la responsable de que un dispositivo pueda operar en alguno de los cinco estados que están definidos en esta tecnología. También, se encarga de definir la estructura de los datos en las tramas que se transmiten mediante BLE.

Un dispositivo no puede estar ejecutándose en más de uno de estos estados al mismo tiempo y hay definida una secuencia lógica concreta para cambiar de un estado a otro. Lo vemos a continuación:

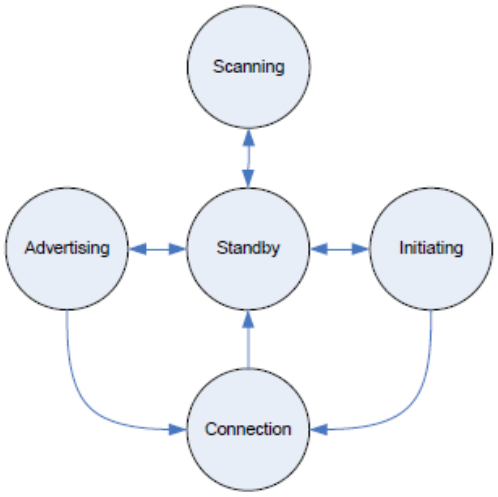


Fig. 1.4 Diagrama de estados de la capa de enlace de un dispositivo Bluetooth

- *Standby State* (Estado de reposo): La capa de enlace en este estado no transmite ni puede recibir paquetes.

- *Advertising State* (Estado publicitario): La capa de enlace estará transmitiendo paquetes de *advertising* a la vez que podrá estar escuchando y respondiendo a las respuestas generadas por estos paquetes. A un dispositivo que se está ejecutando en este modo se le define como *advertiser*.

- *Scanning State* (Estado de escaneo): En este estado el dispositivo estará escuchando por los canales de *advertising*. Un dispositivo en este estado se conoce como *scanner*.

- *Initiating State* (Estado de iniciación): En este estado el dispositivo estará escuchando por los canales de *advertising* paquetes de un dispositivo específico para iniciar una conexión con éste. Una vez recibe el *advertising* del dispositivo deseado, éste envía una serie de datos para establecer la conexión.

- *Connection State* (Estado de conexión): Un dispositivo puede entrar en este estado después de estar en el estado de iniciación o en el estado de *advertising*. En este estado existe una conexión directa entre *master* y *slave* que permite la transmisión de paquetes entre ellos.

Los *beacons* funcionan intercalando el estado de *Standby* con el de *Advertising*. Los dispositivos que recibirían los paquetes *advertising* de los beacon deben estar en modo *scanning* para detectarlos.

La capa de enlace tiene un único formato de trama tanto para las de *advertising* como para las de datos y cuenta con diferentes campos:

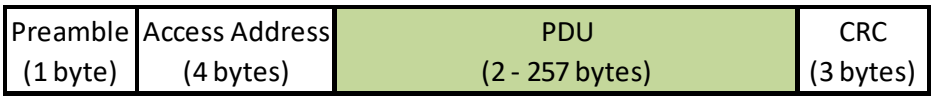


Fig. 1.5 Formato de paquete BLE genérico de la capa de enlace.

- *Preamble* (Preámbulo): Se usa en el receptor para la sincronización de frecuencia, control de ganancia, etc. Los paquetes de *advertising* tienen siempre los bits 10101010 como preámbulo.

- *Access Address* (Dirección de acceso): Valor aleatorio de 32 bits generado por el dispositivo cuando está en el estado de iniciación (*initiating state)* y sirve para identificar la conexión y mantenerla una vez se pasa al estado de conexión. Como para el resto de estados no se genera ese valor aleatorio puesto que no se establecerá ninguna conexión de datos, se utiliza una secuencia de bits predefinida: 10001110100010011011111011010110 (0x8E89BED6). Por lo tanto, los paquetes de *advertising* utilizarán ese valor siempre.

- *PDU* (Unidad de Datos de Protocolo): En este campo se alojarán todos los datos necesarios a transmitir según el tipo de paquete que sea. En un paquete de *advertising*, la PDU tiene una cabecera fija de 16 bits y un tamaño de carga útil (*payload*) variable de 2 a 37 bytes. Lo veremos en más detalle en el siguiente apartado.

- *CRC*: El *Cyclick Redundancy Check* es un código de detección de errores por redundancia cíclica que se calcula a partir de la PDU.

1.2.2.1. PDU de un paquete advertising

La PDU de un paquete *advertising* tiene una longitud máxima de 39 bytes, dos de los cuales corresponden a la cabecera y el resto a la carga útil (*payload*).

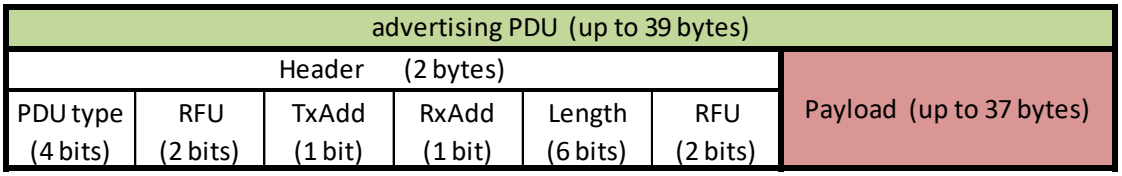


Fig. 1.6 Trama PDU de un paquete de *advertising*

- *PDU type*: En este campo se indica cuál de los siete tipos de PDU que existen para un paquete *advertising* se utiliza. A continuación, veremos algunos de estos tipos.

- *TxAdd*: Indica si la dirección del anunciante en el campo *AdvA* del *payload* es pública (*TxAdd* = 0) o aleatoria (*TxAdd* = 1).

- *RxAdd*: Indica si la dirección a la que va dirigido el paquete *advertising* es pública (*RxAdd* = 0) o aleatoria (*RxAdd* = 1).

- *Length*: Indica la longitud del *payload* de la PDU. El rango válido de longitud va de 6 a 37 bytes.

- *RFU*: Campos reservados para usos futuros.

- *Payload*: Campo de longitud variable que contiene la carga de datos útil del paquete publicitario.

Hay definidos siete tipos de PDUs publicitarias que únicamente son transmitidas por los tres canales publicitarios disponibles para ese uso (CH37, CH38, CH39). Cuatro pueden ser utilizadas en el estado publicitario (ADV_*), otros cuatro en el estado de escaneo (SCAN_*) y uno en el estado de iniciación (CONNECT_REQ). El diagrama de estados lo hemos visto en la Fig. 1.4.

PDU Type $b_3b_2b_1b_0$	Packet Name
0000	ADV_IND
0001	ADV_DIRECT_IND
0010	ADV_NONCONN_IND
0011	SCAN_REQ
0100	SCAN_RSP
0101	CONNECT_REQ
0110	ADV_SCAN_IND
0111-1111	Reserved

Fig. 1.7 Tipos de PDUs de *advertising*

Puesto que los *beacons* únicamente pueden estar en el estado publicitario (de *advertising*), únicamente definiremos los cuatro eventos en los que pueden transmitir para saber cuál de ellos es el empleado por los *beacons* y así saber que estructura de paquete tienen definida.

- *ADV_IND*: Este tipo de evento se conoce como conectable y no dirigido (*conectable undirected*) y es el más común. Que sea conectable indica que un dispositivo central puede conectarse a uno periférico que esté enviando este tipo de *advertisings*. Que no sea dirigido hace referencia a que los paquetes se envían sin ningún destino definido, es decir, en modo *broadcast*.

Este tipo de paquetes lo utilizan los dispositivos periféricos (futuros *slaves*) para ayudar a dispositivos centrales (*Master*) a encontrarlos. Una vez encontrado, el dispositivo central puede proceder a iniciar el proceso de conexión.

- *ADV_DIRECT_IND*: Esta PDU es conectable y dirigida (*conectable directed*) y se utiliza cuando el *advertiser* quiere conectarse con un dispositivo concreto. Este tipo de paquete se utiliza para solicitar la conexión a otro dispositivo de manera directa.

- *ADV_NONCONN_IND*: Este tipo de evento es no conectable y no dirigido (*non-connectable* undirected) y se utiliza básicamente para emitir datos en modo *broadcast*, sin intención de establecer ningún tipo de conexión con los dispositivos que reciban los paquetes. Los dispositivos *scanner* que detecten estos paquetes únicamente pueden leer los datos del paquete.

- *ADV_SCAN_IND*: Este tipo de evento no dirigido pero escaneable (*scannable undirected*) se utiliza para emitir datos en modo *broadcast* sin permitir la conexión de ningún dispositivo. Sin embargo, dejan que los que los reciben puedan interactuar con ellos y solicitarles información adicional.

Los *beacons* utilizan el tipo de evento *ADV_NONCONN_IND* para sus transmisiones publicitarias y su *payload* está definido con la siguiente estructura de PDU:

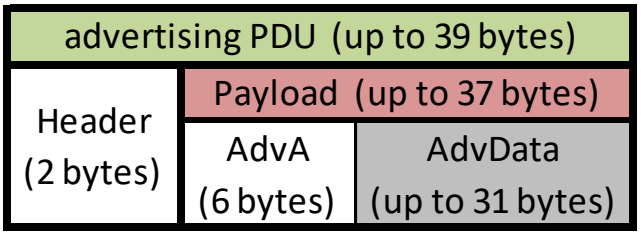


Fig. 1.8 PDU de un paquete *advertising* del tipo *ADV_NONCONN_IND*

- *AdvA*: Contiene la dirección del dispositivo público o aleatorio del anunciante, tal y como se indics en la *TxAdd*.

- *AdvData*: Este campo contiene los datos publicitarios del anfitrión. Existen diferentes tipos de protocolos para *beacons* que definen una estructura concreta de datos según su especificación, los tres más conocidos los veremos en profundidad en el apartado 1.3. de este capítulo.

1.2.2.2. Intervalo de Advertising

Cuando un dispositivo está en el estado de *advertising*, éste envía periódicamente paquetes de algunos de los tipos comentados en el subapartado anterior por los 3 canales disponibles (CH37, CH38, CH39). El tiempo que transcurre durante el envío de los paquetes por esos tres canales se define como evento publicitario.

El tiempo que pasa entre un evento publicitario y el siguiente está definido por un intervalo configurable entre 20ms y 10,24s con incrementos de 0,625ms, llamado intervalo de *advertising*, y un retardo aleatorio comprendido entre 0 y 10ms, que sirve para reducir colisiones entre dispositivos que están emitiendo en el mismo modo de *advertising*.

Cada envío del paquete de *advertising* por un canal tiene una duración máxima de 10ms, por lo que un evento publicitario no podrá durar más de 30ms.

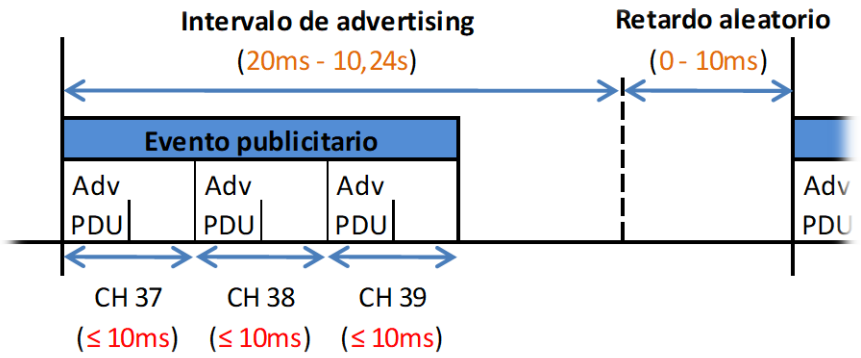


Fig. 1.9 Distribución de tiempos entre eventos publicitarios

1.3. Protocolos para beacons

Los *beacons* aprovechan el modo *advertising* de BLE para transmitir datos a dispositivos cercanos que estén dentro de su radio de alcance con eventos *ADV_NONCONN_IND* tal y como hemos comentado en el apartado anterior.

Este tipo de paquete tiene un campo de datos de 31 bytes (*AdvData*) que es utilizado por estos protocolos para contener la información útil que emite el *beacon* en función de las características del protocolo seleccionado. Estos datos luego los podemos recoger en un dispositivo que detecte estos paquetes y que tenga implementado el mismo protocolo para “descifrar” los datos y ejecutar acciones.

Estos protocolos aparecieron para tratar de dar soporte a las nuevas tendencias de mercado o a los nuevos hábitos de la sociedad, como que a día de hoy prácticamente todo el mundo lleva consigo a todas partes un *smartphone* con conectividad BLE.

A continuación, vamos a definir los protocolos más utilizados en la actualidad y su especificación para ver qué ventajas aporta utilizar uno u otro:

1.3.1. iBeacon

iBeacon es un protocolo creado por Apple, presentado en 2013 en la WWDC, y está soportado de forma nativa en los dispositivos iOS a partir de la versión 7 o superior. Fue el primer protocolo para *beacons* introducido en el mercado y actualmente es uno de los más utilizados por su simplicidad.

iBeacon utiliza BLE para transmitir un identificador único universal (UUID) y otros parámetros que son recogidos por una aplicación o sistema operativo compatible con el protocolo. Estos parámetros se pueden utilizar para identificar la posición física del dispositivo y/o lanzar acciones basadas en la localización como pueden ser las notificaciones *push*.

1.3.1.1. Especificación

El protocolo iBeacon utiliza 30 de los 31 bytes disponibles en el campo *advdata* del *payload* de un paquete *advertising* del tipo *ADV_NONCONN_IND*. En la trama hay campos con parámetros fijados por el fabricante (en gris en la Fig. 1.10) y otros que son configurables por el administrador del *beacon* (en azul en la Fig. 1.10). La estructura de datos y las funcionalidades de cada campo son las siguientes:

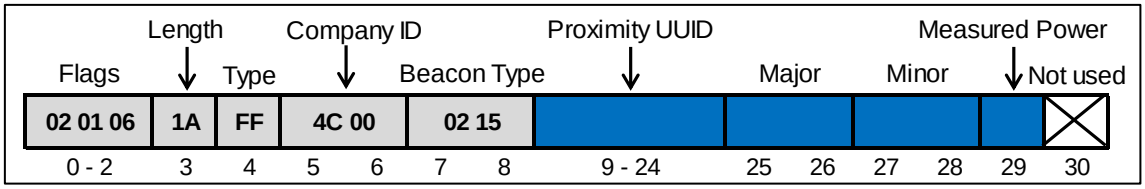


Fig. 1.10 *Payload* de un paquete *advertising* con el protocolo iBeacon

- *Flags* [3 bytes]: Se indica el tipo de elemento de datos de *advertising* según el número asignado por el GAP [10].Este parámetro nos indica que el UUID de los servicios proporcionados por este dispositivo se puede anunciar en formato de 128 bits (16 bytes).

- *Length* [1 byte]: Se indica la longitud de datos para el resto de campos especificados por el fabricante (Apple). 0x1A corresponde a una longitud de 26 bytes.

- *Type* [1 byte]: Este parámetro también está especificado en el GAP [10] y *0xFF* representa que hay una estructura de datos especificada por el fabricante: *Manufacturer Specific Data*.

- *Company ID* [2 bytes]: Código de identificación de empresa del fabricante del *beacon*. El código es asignado por el SIG [11] y en este caso 0x004C corresponde a Apple.

- *Beacon type* [2 bytes]: indica que es un *beacon* de proximidad usado por todos los iBeacons (0x02) y define la longitud restante de datos (0x15 = 21 bytes).

- *Proximity UUID* [16 bytes]: identificador único universal del *beacon*.

- *Major* [2 bytes]: identificador de un subconjunto de *beacons* dentro de un grupo grande.

- *Minor* [2 bytes]: identificador de un *beacon* concreto dentro del grupo grande definido en el *Major*.

- *Measured Power* [1 byte]: Valor de potencia medida a un metro del dispositivo. Este valor debe ser calibrado por cada dispositivo por el administrador o el fabricante del *beacon* y se utiliza para estimar el cálculo de la distancia.

1.3.2. AltBeacon

Radius Networks [12] anunció este protocolo de código abierto en julio de 2014 para competir con protocolo iBeacon de Apple. AltBeacon dota de algo más de flexibilidad, en los parámetros configurables, a los desarrolladores que lo implementan en sus *beacons*. A pesar de todo, es uno de los menos utilizados actualmente.

1.3.2.1. Especificación

AltBeacon tiene prácticamente la misma estructura de trama que un iBeacon pero libera el identificador de la compañía y del *beacon* y añade un parámetro extra para uso específico del fabricante.

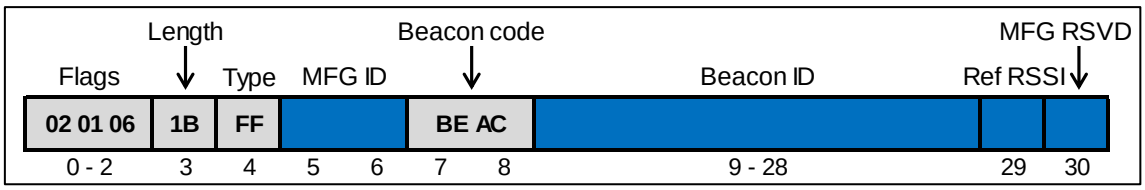


Fig. 1.11 *Payload* de un paquete *advertising* con el protocolo AltBeacon

- *Flags* [3 bytes]: Se indica el tipo de elemento de datos de *advertising* según el número asignado por el GAP [10].

- *Length* [1 byte]: Se indica la longitud de datos para el resto de campos de la trama. 0x1B corresponde a una longitud de 27 bytes.

- *Type* [1 byte]: Este parámetro también está especificado en el GAP [10] y *0xFF* representa que hay una estructura de datos especificada por el fabricante: *Manufacturer Specific Data*.

- *MFG ID* [2 bytes]: Código de identificación de empresa del fabricante del *beacon*. El código es asignado por el SIG [11].

- *Beacon code* [2 bytes]: código *advertising* para AltBeacon. Código obligatorio para indicar a los receptores que los campos que vienen a continuación son los del protocolo AltBeacon.

- *Beacon ID* [20 bytes]: Identificador de 20 bytes único para el *beacon*. Para fines de interoperabilidad, los primeros 16 bytes del identificador deben ser exclusivos de la unidad organizativa del anunciante. Los 4 bytes restantes pueden subdividirse según convenga. Si se dividen en 2 y 2 equivaldría al valor *Major* y *Minor* del protocolo iBeacon.

- *Reference RSSI* [1 byte]: Valor que representa la intensidad media de la señal recibida a 1m del *beacon* anunciante.

- *MFG RSVD* [1 byte]: Reservado para su uso por el fabricante o administrador por si quiere implementar características especiales. La interpretación de este valor debe ser definida por el fabricante y debe ser evaluada basándose en el valor MFG ID. Algunos de los usos para los que se destina es para enviar información del estado de la batería del *beacon*.

1.3.3. Eddystone

Protocolo de código abierto para *beacons* desarrollado por Google que fue presentado en julio del año 2015. Google, con esta tecnología, pretende fomentar el Internet de las cosas (IoT) y, para ello, ha desarrollado cuatro tipos de tramas dentro del mismo protocolo que tienen funcionalidades diferentes.

Google permite interactuar con *beacons* mediante notificaciones *Nearby* [13] gracias los servicios de Google y su API o directamente con la web física (*Physical Web* [14]) sin necesidad de usar esos servicios. Éste último podría interactuar con un iPhone tal y como se define en el anexo (A3.2).

1.3.3.1. Especificación

Este protocolo tiene una estructura de trama común en todas sus variantes y una carga de hasta 19 bytes para los datos específicos de cada uno de sus cuatro tipos de formatos definidos.

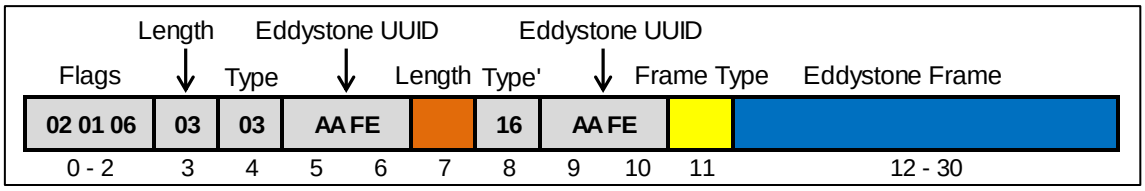


Fig. 1.12 *Payload* genérico de un paquete *adv.* con el protocolo Eddystone

- *Flags* [3 bytes]: Se indica el tipo de elemento de datos de *advertising* según el número asignado por el GAP [10].

- *Length* [1 byte]: 0x03 corresponde a una longitud de 3 bytes. Los campos de longitud siempre se refieren a la longitud hasta el próximo parámetro de longitud que se encuentre en la trama.

- *Type* [1 byte]: Este parámetro también está especificado en el GAP [10] y *0x03* representa que se incluirá una clase de servicio de identificadores únicos de 16 bytes en vez de los de 128 bytes que se usan en los protocolos iBeacon o AltBeacon.

- *Eddystone UUID* [2 bytes + 2bytes]: Identificador único de 16 bits asignado al miembro de Bluetooth SIG Google Inc. (0xFEAA) [15]. En este caso, se utiliza para identificar las tramas Eddystone. Este parámetro se emplea dos veces en la trama.

- *Length* [1 byte]: este parámetro depende del tipo de trama Eddystone que se está utilizando, lo veremos más adelante en las definiciones de estos formatos.

- *Type'* [1 byte]: Este parámetro también está especificado en el GAP [10] y *0x16* representa que se incluirá un dato de servicio de 16 bits: *Service Data - 16-bit UUID.*

- *Frame Type* [1 byte]: Especifica el tipo concreto de trama Eddystone codificada. Por el momento, existen 4 tipos de tramas que veremos en los siguientes apartados. El resto de posibles valores específicos que se pueden asignar a este campo están reservados para nuevos tipos de trama que puedan desarrollar en un futuro.

- *Eddystone Frame* [19 bytes]: Parámetros específicos de cada tipo de trama Eddystone: 0x00 para Eddystone-UID, 0x10 para Eddystone-URL, 0x20 para Eddystone-TLM y 0x30 para Eddystone-EID.

A continuación, veremos los cuatro tipos de trama definidos en este protocolo:

1.3.3.2. Eddystone–UID

Esta trama transmite un ID único de 16 bytes compuesto por un espacio de nombres de 10 bytes y una instancia de 6 bytes. El ID del *beacon* puede ser útil para asignar un dispositivo a un registro en un almacenamiento externo. La parte de espacio de nombres del ID puede usarse para agrupar un conjunto particular de *beacons*, mientras que la instancia identifica dispositivos individuales en el grupo. La división del ID en el espacio de nombres y los componentes de instancia también se pueden utilizar para optimizar las estrategias de exploración de BLE filtrando, por ejemplo, sólo en el espacio de nombres.

Este tipo de trama vendría a ser, salvando las diferencias, la equivalente a la trama del protocolo iBeacon.

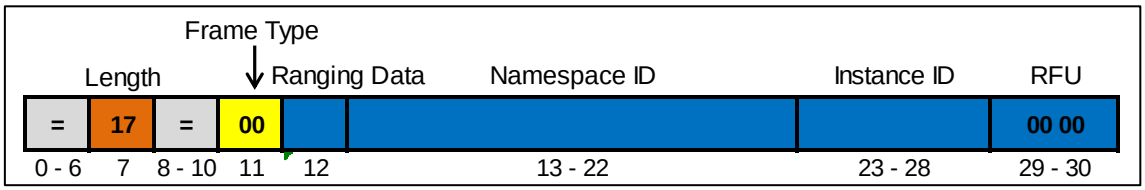


Fig. 1.13 *Payload* de un paquete *adv.* con el protocolo Eddystone-UID

- *Length* [1 byte]: Longitud restante para completar los 31 bytes de los datos de *advertising* (AdvData). 0x17 = 23 bytes.

- *Frame Type* [1 byte]: codificación para el tipo de trama Eddystone-UID.

- *Ranging Data* [1 byte]: Potencia de transmisión en dBm emitida por el *beacon* a 0 metros, a diferencia de otras especificaciones que lo miden a 1 metro. El valor es un entero de 8 bits y oscila entre -100dBm y +20dBm con una resolución de 1dBm. Se recomienda a los desarrolladores que para determinar este valor hagan la medida a 1 metro de distancia y a continuación añadan 41dBm, que es la pérdida de señal que se produce a 1 metro de distancia.

- *Namespace ID* [10 bytes]: Espacio de nombres auto-asignado. Este espacio de nombres está diseñado para asegurar la unicidad de ID entre múltiples implementadores de Eddystone y puede usarse para filtrar el escaneo en los dispositivos de escaneo.

- *Instance ID* [6 bytes]: Identificador único dentro del Identificador de espacio de nombres (Namespace ID). Este ID puede ser generado mediante cualquier método aleatorio, secuencial, jerárquico, etc.

- *RFU* [2 bytes]: Campo reservado para usos futuros.

1.3.3.3. Eddystone–URL

Esta trama permite difundir una URL utilizando un formato de codificación comprimido para aprovechar al máximo los pocos bytes útiles libres. Cualquier dispositivo que recibiera el paquete podría acceder a esa URL.

Eddystone-URL forma la columna vertebral de la Web física [14] e incorpora todos los aprendizajes del formato UriBeacon [16] del cual evolucionó.

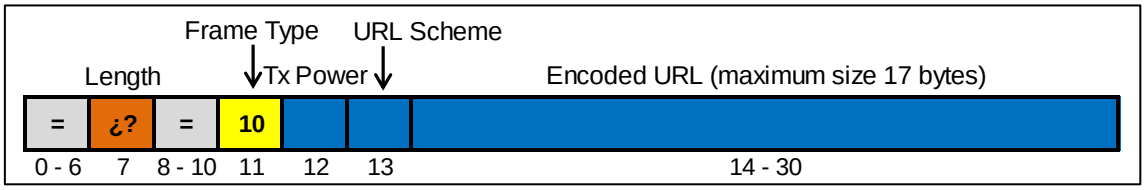


Fig. 1.14 *Payload* de un paquete *adv.* con el protocolo Eddystone-URL

- *Length* [1 byte]: La longitud de este campo depende de la longitud de la URL que se introduce en el último campo por lo que como máximo tendrá un valor de 0x17 (23 bytes).

- *Frame Type* [1 byte]: codificación para el tipo de trama Eddystone-URL.

- *Tx Power* [1 byte]: Potencia de transmisión calibrada a 0 metros.

- *URL Schema* [1 byte]: Define el esquema de identificador, un prefijo opcional y como se codifica el resto de la URL.

- *Encoded URL* [1 a 17 bytes]: La codificación consiste en una secuencia de caracteres. Los códigos de caracteres excluidos de la codificación de URL se utilizan cómo códigos de expansión de texto y se pueden encontrar en su GitHub [17].

Puesto que las URL difícilmente ocupan menos de 17 bytes, aunque se use el formato de compresión, muy probablemente deberemos utilizar algún acortador de URLs como el que tiene Google [18]. En la hoja de anexos se puede encontrar un documento detallado de cómo se configuró un *beacon* con el formato Eddystone-URL para que emitiera una URL en un evento que se hizo en la universidad.

1.3.3.4. Eddystone–TLM

Los *beacons* compatibles pueden transmitir datos sobre sus propias operaciones, funcionamiento y estado a los administradores de éstos. También pueden utilizarse para transmitir parámetros relacionados con los sensores extra que integren esos *beacons*.

Como la trama de telemetría Eddystone no cuenta con un identificador de *beacon*, éste debe enviarse intercalado con una trama Eddystone-URL o Eddystone-UID, que sí que lo tienen. Se puede decidir la frecuencia de intercalado de las tramas TLM en función de los requerimientos de la aplicación.

Las tramas de telemetría pueden ser difundidas en claro o cifradas con una clave de identidad. Veamos los dos tipos:

La trama transmitida en claro debe ir asociada a una trama Eddystone-URL o Eddystone-UID y la estructura de datos que conforma la trama que se transmite es la siguiente:

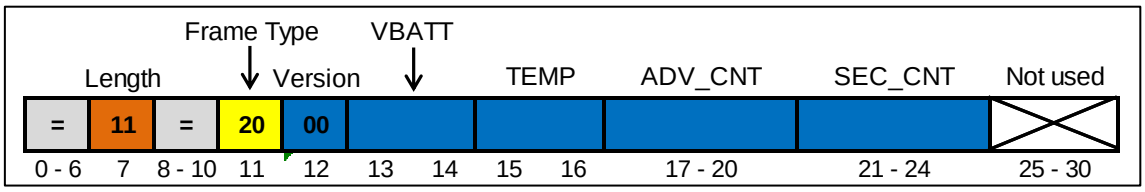


Fig. 1.15 *Payload* de un paq.*adv.* con el protocolo Eddystone-TLM sin encriptar

- *Length* [1 byte]: 17 bytes (0x11). Los 5 últimos bytes no se utilizan.
- *Frame Type* [1 byte]: Codificación para el tipo de trama Eddystone-TLM.
- *Versión* [1 byte]: Versión de trama de telemetría sin encriptar.
- *VBATT* [2 bytes]: Carga actual de la batería en milivoltios, expresada como 1mV por bit. Si es un *beacon* alimentado por otro tipo de fuente como USB o está directamente conectado a la corriente, el valor debe ponerse a 0.
- *TEMP* [2 bytes]: Temperatura detectada por el *beacon*. Si no se admite el valor, se debe establecer el campo en 0x8000 (-128ºC).
- *ADV_CNT* [4 bytes]: Contador de las tramas *advertising* de todos los tipos enviadas por el *beacon* desde el encendido o reinicio.
- *SEC_CNT* [4 bytes]: Contador que representa el tiempo transcurrido desde que el *beacon* se enciende o reinicia.

Si un *beacon* Eddystone se ha configurado con Eddystone-EID, cuando se le pida que emita su telemetría, utilizará este tipo de trama cifrada para evitar transmitir información que pueda hacer que el dispositivo sea identificable de forma exclusiva e impedir que se pueda leer la información que se transmite. La estructura de datos de la trama es la siguiente:

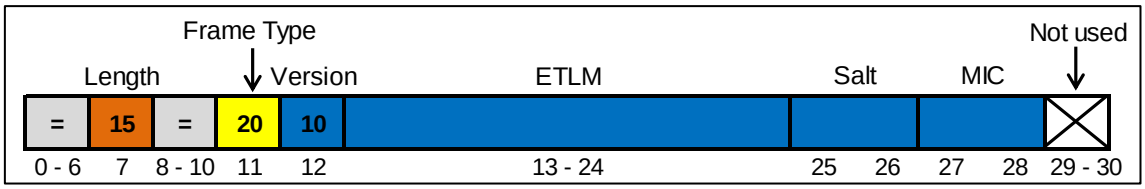


Fig. 1.16 *Payload* de un paq. *adv.* con el protocolo Eddystone-TLM encriptado

- *Length* [1 byte]: 21 bytes (0x15). Los 2 últimos bytes no se utilizan.
- *Frame Type* [1 byte]: Codificación para el tipo de trama Eddystone-TLM.
- *Versión* [1 byte]: Versión de trama de telemetría encriptada.
- *ETLM* [12 bytes]: datos de telemetría encriptados. Estos 12 bytes corresponderían a los campos VBATT, TEMP, ADV_CNT y SEC_CNT de la versión de telemetría sin encriptar.
- *SALT* [2 bytes]: 16 bits aleatorios utilizados para cifrar la trama. Para encriptar el campo de datos, se utiliza el algoritmo AES-EAX que proporciona autenticación y privacidad.
- *MIC* [2 bytes]: Campo de comprobación de la integridad del mensaje.

1.3.3.5. Eddystone–EID

Este tipo de trama transmite un identificador efímero cifrado que cambia periódicamente a una velocidad determinada durante el registro inicial con un servicio web.

El identificador efímero puede ser resuelto remotamente por el servicio con el que fue registrado, pero a la vista de otros observadores, parece estar cambiando aleatoriamente. Esta trama está diseñada para su uso en servicios en los que se requiere seguridad y privacidad. Además, se evita el uso de estos *beacons* por terceros.

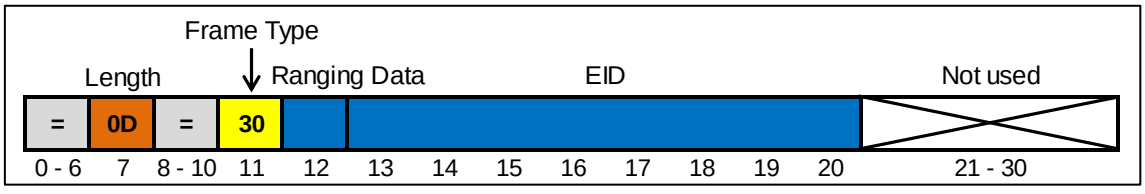


Fig. 1.17 *Payload* de un paquete *adv.* con el protocolo Eddystone-EID

- *Length* [1 byte]: 13 bytes (0x0D). Los 10 últimos bytes no se utilizan.

- *Frame Type* [1 byte]: Codificación para el tipo de trama Eddystone-EID.

- *Ranging Data* [1 byte]: Potencia de transmisión calibrada a 0 metros.

- *EID* [8 bytes]: Identificador efímero generado criptográficamente según su especificación [19].

Dado que el propósito de este tipo de trama es ofrecer una mejor privacidad, no se recomienda intercalar estos mensajes *advertising* con ningún otro tipo de trama exclusivo del mismo *beacon*. Además, el *beacon* debe ir rotando su dirección *broadcast* (BD_ADDR) y el valor EID para evitar el rastreo más allá del período de rotación.

1.3.3.6. Servicio de configuración

Eddystone proporciona un servicio de configuración GATT [20] que permite la interoperabilidad entre los fabricantes de *hardware* y los desarrolladores de aplicaciones. Si los *beacons* cumplen con la especificación completa del protocolo, éstos se pueden registrar en este servicio de Google que permite reconfigurar los datos de difusión y muchos parámetros básicos utilizando la aplicación de Google Beacon Tools [21]. Además, estar registrado en este servicio permite, entre otras cosas, programar notificaciones *nearby* [22], algo muy interesante que veremos en capítulos posteriores.

1.4. Funcionamiento

Con todo lo que hemos visto hasta ahora, sabemos cómo es físicamente un *beacon*, qué tecnología utiliza para emitir sus paquetes de *advertising* y con qué protocolos podemos configurar esos paquetes para enviar cierta información. Estas señales que emite el *beacon* de manera *broadcast*, necesitan de algún sistema inteligente que las detecte, recopile y, lo más importante, realice las acciones pertinentes en función del servicio que hemos implementado.

En la gran mayoría de servicios basados en *beacons*, se diseña una aplicación móvil que, instalada en un *smartphone* compatible con BLE, nos permite detectar estos paquetes. A partir de la recopilación de estos paquetes, se suele establecer una conexión a alguna infraestructura en la nube que nos diga qué acciones realizar cuando se detecta alguno de los *beacons* desplegados en una región.

Lo más común y sencillo es, una vez recibido un paquete, conectarse a una base de datos y descargar la información asociada a ese *beacon* para ofrecer el servicio para el que fue implementado.

Las aplicaciones detectan e interactúan con *beacons* de dos maneras: monitoreo (*monitoring*) y alcance (*ranging*).

1.4.1. Monitoreo

La aplicación realiza acciones al entrar o salir del rango de una región, es decir, permite a una aplicación saber cuándo entra o sale de la zona de cobertura de los *beacons* configurados con esa región.

El monitoreo únicamente reconoce eventos de entrada o salida de la región y no proporciona información sobre qué *beacon* concreto desencadenó el evento. Tampoco ofrece estimaciones de proximidad y la sensibilidad de detección no es muy alta, es decir, detectar la entrada o salida de una región se puede demorar hasta algunos minutos dependiendo de en qué modo de ahorro de energía se encuentre el *smartphone* en ese momento. Los eventos de salida de la región se suelen retrasar 30 segundos para evitar notificaciones de salida de la región erróneas.

El sistema operativo iOS incorpora un *framework* desde la versión 7.0 que integra toda su API nativa de localización, lo que permite tener ejecutándose el servicio de monitoreo de la aplicación aunque ésta esté abierta ni en *background* en el dispositivo del usuario. Apple permite monitorizar hasta 20 regiones a la vez.

El sistema operativo Android no integra un sistema de monitoreo de regiones de *beacons* que permita detectar la entrada a una región definida por éstos, aunque, si el usuario tiene la aplicación cerrada cuenta con la opción de crear geovallas (*geofences*) [23].

Una geovalla es un área circular con el centro en unas coordenadas concretas (latitud y longitud) y un radio lo más óptimo posible para contener una localización geográfica equivalente a la zona de cobertura de nuestros *beacons*. Esta área funcionaría del mismo modo que en el caso de iOS, un *smartphone* con Android que tuviera esa geovalla definida, realizaría acciones en función de si entramos o salimos de esa zona. Se pueden definir hasta 100 geovallas a la vez.

1.4.2. Alcance

La aplicación ejecuta acciones basadas en la detección de un *beacon* concreto o de su proximidad respecto al usuario. La detección por alcance nos permite recopilar una lista de los *beacons* detectados con todos sus parámetros encapsulados en el *payload* de los paquetes de *advertising* que emiten. A partir de esa información que recopilamos, nuestra aplicación actuará en consecuencia con los servicios que hayamos implementado.

La estimación de la distancia entre usuario y *beacon* se ve alterada con facilidad, ya sea por el movimiento de personas, o por paredes y objetos que puedan crear interferencias en el entorno. Se recomienda la colocación de *beacons* en entornos abiertos y de visión directa con la zona de estancia de los usuarios a los que queremos dar servicio para que las aproximaciones sean lo más precisas posibles.

1.5. Aplicaciones y usos

Implementar servicios con *beacons* es relativamente sencillo en cuanto a la configuración del entorno se refiere, puesto que sólo hay que adquirirlos y configurarlos con un protocolo según nuestras necesidades o nuestra idea de servicio a ofrecer al usuario.

Los servicios que se pueden llegar a implementar con *beacons* únicamente tienen como límite la imaginación de cada uno. Hay que saber encontrar la manera de integrarlos en cada situación para obtener un servicio útil para el usuario.

Podemos encontrar muchos de los casos de éxito de implementaciones o servicios con *beacons* en la siguiente recopilación de *UsingBeacons* [24]. A continuación, veremos los usos más comunes para los que se emplean estos dispositivos.

1.5.1. Publicidad y geomarketing

Los *beacons* pueden precisar la situación de un cliente en interiores, lo que permite a comercios o grandes superficies dar información relevante o mostrar ofertas en función de en qué sección o zona se encuentran, o en función del tipo de cliente o consumidor que sea.

La gran cantidad de información que pueden llegar a recopilar de sus clientes, puede ayudar a los comerciantes a redistribuir los productos de la tienda en función de las zonas que sean más transitadas, crear campañas publicitarias a medida, etc.

Este tipo de tecnología no está muy extendida en España, aunque se han realizado algunas pruebas piloto en algunas cadenas de supermercados como Condis [25].

En Estados Unidos, en cambio, hay muchos más ejemplos de despliegue de *beacons*. Alguno de los más representativos es el de los supermercados Macy's que integraron esta tecnología en 2013 e hicieron una especie de juegos para que los clientes fueran a sus tiendas y las fueran recorriendo en busca de recompensas durante la semana del *Black Friday* [26], aportándoles un aumento de ventas muy significativo respecto a años anteriores. Desde entonces tienen esta tecnología en su aplicación y proporcionan al usuario alertas en tiempo real y notificaciones de eventos o promociones entre otras funciones [27].

1.5.2. Localización en interiores

Como ya hemos comentado anteriormente, los *beacons* nos permiten estimar la ubicación de los usuarios cercanos a ellos. Esta información nos permitiría situar al usuario en un mapa e ir notificándole información útil para él.

Una de las implementaciones más frecuentes de este tipo de servicio se encuentra en aeropuertos, estaciones de metro, grandes superficies…

Aena tiene implantada esta tecnología en el aeropuerto de Barcelona-El Prat desde hace algunos años y permite suministrar información directa a los pasajeros sobre sus vuelos, tiempos de paso en los filtros, ofertas comerciales, etc [28]. Toda la tecnología para trabajar con esos *beacons*, que utilizan el protocolo iBeacon visto en el apartado 1.3.1. , está integrada en su aplicación para *smartphones* [29].

1.5.3. Otros servicios e implementaciones

Museos, hospitales, universidades, centros públicos… Cualquier tipo de ubicación es válida para desplegar servicios con esta tecnología siempre que se encuentre la manera de dar un servicio útil para los usuarios.

En museos normalmente se coloca un *beacon* en cada obra para mostrar al usuario información relevante sobre ésta de manera automática al acercarse a ella. El museo Guggenheim de Nueva York fue uno de los primeros en implementar esta tecnología y facilita a sus usuarios la recopilación de información en tiempo real de las obras que están visualizando [30].

También se pueden integrar *beacons* en dispositivos que adquieren los usuarios para permitir localizarles en caso de pérdida o para dotarles de información extra. Algunos de los casos más interesantes los podemos ver en dos proyectos que ha desarrollado el fabricante Accent Systems para la marca de coches Volkswagen y el fabricante de maletas Samsonite:

- *Volkswagen Connect* [31]: Consiste en una llave de coche que integra en el llavero un *beacon* que interactúa con la aplicación del usuario permitiéndole encontrar las llaves en caso de pérdida.

- *Samsonite Track&Go* [32]: Las maletas que incorporan estos *beacons* permiten localizar de manera segura el equipaje perdido, puesto que utilizan el protocolo Eddystone-EID visto en el apartado 1.3.3.5.

Los fabricantes que integran este tipo de servicios a productos ya existentes, consiguen tener un gran número de usuarios con su aplicación instalada en su *smartphone*, lo que permite tener un canal de comunicación abierto con sus clientes.

Ese canal de comunicaciones abierto permite, mediante notificaciones *push*, comunicar al cliente posibles novedades, ofertas, actualizaciones, etc. Dicha notificación puede incentivar a un gran número de clientes a adquirir algún nuevo producto o mejorar alguno de los servicios contratados, aumentando el beneficio de la empresa.

CAPÍTULO 2. PLATAFORMA UTILIZADA

Para la realización de este proyecto, contamos con un kit de desarrollo que incorpora 3 *beacons* de proximidad BLE de la marca Estimote [33] valorado en unos 60€. También contamos con un kit de la marca Accent Systems [34] con un precio en el mercado de 75€. Este último incluye 5 *beacons* del modelo iBKS 105 [35].

Ambos fabricantes afirman que sus dispositivos alcanzan una distancia de transmisión de hasta 70 metros y que sus baterías suelen tener una vida media de unos dos años con la configuración de fábrica. Los dos son compatibles con los protocolos iBeacon [36] y Eddystone [37] que hemos definido en el apartado 1.3. de esta memoria.

Fig. 2.1 Kit *beacons* Estimote [33] (izq.) y kit Accent Systems (derecha) [34]

En los siguientes subapartados veremos detalladamente el *hardware* y *software* de estos modelos de *beacons* y algunas de sus funcionalidades.

2.1. Hardware y firmware de nuestros beacons

Tal y como vimos en el apartado 1.1.1. , los *beacons* cuentan con una carcasa que contiene una pila y un chip en su interior.

Fig. 2.2 *beacon* Estimote (izq.), *beacon* Accent Systems (derecha)

2.1.1. Carcasa y ubicaciones

Los *beacons* Estimote tienen un recubrimiento de silicona totalmente sellado que protege frente al agua y al polvo los componentes internos, lo que permite que sean ubicados en zonas exteriores o con condiciones ambientales más adversas. La extracción de la batería, en consecuencia, no es sencilla y requiere el remplazo de la parte adhesiva inferior por otra de nueva.

La carcasa de los *beacons* Accent Systems son dos pequeñas tapas de plástico rígido que se enroscan entre sí albergando en su interior el chip y la batería. Este mecanismo de cierre no sella completamente la estructura, lo que comporta que su uso sea siempre en interiores para evitar que entre humedad en el interior. Como aspecto positivo destacamos lo fácil que es efectuar la sustitución de la batería.

Ambos dispositivos cuentan en la parte inferior con un potente adhesivo que los deja enganchados permanentemente en cualquier superficie lisa.

2.1.2. Chipset y componentes extra

Nuestros dos tipos de *beacons* cuentan con el mismo microcontrolador, el nRF51882 [38] del fabricante Nordic Semiconductor's. Este microcontrolador viene acompañado de un procesador de 64Mhz ARM Cortex-M4F, 512kB de memoria flash y 64kB de memoria RAM.

Los dos modelos son compatibles con el estándar Bluetooth LE a 2.4GHz [39] y pueden emitir paquetes con el protocolo iBeacon [36] o Eddystone [37].

Los *beacons* Estimote integran también un chip NFC y un sensor de movimiento que puede utilizarse, entre otras cosas, para desactivar las emisiones del *beacon* al ponerlo boca-abajo.

2.1.3. Batería y duración

Nuestros modelos de *beacons* utilizan la misma pila de 1000mAh (modelo CR2477). Podemos comprar 10 de estas pilas con Amazon Prime a un precio de 35€ [40], lo que equivale a 3,50€ la unidad.

La estimación de duración de la batería es muy cambiante en función de a qué potencia y a qué velocidad de transmisión estén configurados los *beacons*, tal y como vimos en el apartado 1.1.2.3.

Pada determinar esa variabilidad de la batería de nuestros *beacons* se ha cogido un *beacon* Estimote con la batería al 100% y se le han ido configurando diferentes valores de potencia de transmisión e intervalo de emisión, obteniendo en cada caso una estimación de duración de batería dada por el mismo fabricante. Los resultados obtenidos en la duración de la batería son los siguientes:

Tabla 2.1. Duración, en meses, de la batería de un *beacon* Estimote en función de la potencia a la que se transmiten los paquetes de *advertising* (fila superior) y a la frecuencia a la que lo hacen (columna izquierda).

	-40dBm (0,25m)	-20dBm (3,5m)	-16dBm (7m)	-12dBm (15m)	-8dBm (30m)	-4dBm (40m)	0dBm (50m)	+4dBm (70m)
100ms	13	12	12	12	11	11	10	8
500ms	39	37	37	37	36	35	33	28
1s	51	50	50	49	49	48	46	41
5s	68	68	68	68	68	67	66	64
10s	71	71	71	71	71	71	70	69

Tras estos resultados, podemos apreciar que, dentro de un mismo intervalo de emisión, la duración de la batería puede variar entre 2 y 5 meses dependiendo de la potencia que elijamos. En cambio, si miramos la variación entre intervalos de emisión, la diferencia puede llegar a ser de más de 12 meses.

Con todo esto, podemos concluir que será muy importante elegir el valor de emisión de *advertisings* más grande posible para nuestra aplicación, puesto que ese parámetro será el mayor condicionante para la vida útil de la batería. La potencia de emisión no será un valor tan importante para la duración de la batería, por lo que podemos sobredimensionarla o dejarla a máxima potencia.

2.2. Software y funcionalidades de nuestros beacons

La gran mayoría de fabricantes cuentan con una aplicación que permite conectarse a sus *beacons* para configurar sus parámetros básicos o activar algunas funcionalidades extra. A continuación, veremos qué nos ofrecen nuestros fabricantes de *beacons* y sus ventajas e inconvenientes.

2.2.1. Aplicaciones de configuración

Tanto el fabricante Estimote como Accent Systems tienen publicados en Play Store (Android) [41] [42] y en App Store (iOS) [43] [44] sus propias aplicaciones de configuración para sus *beacons*:

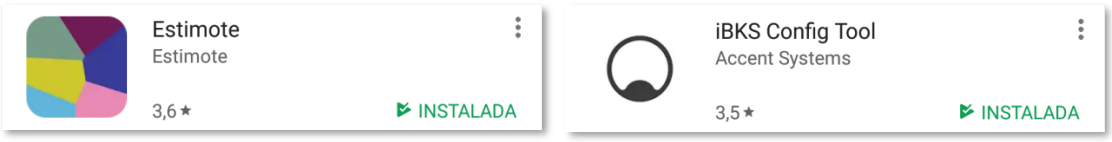


Fig. 2.3 Aplicaciones para configurar nuestros *beacons*

Estas aplicaciones nos permiten conectarnos a nuestros *beacons* para, de una manera cómoda y visual, configurarlos con alguno de los protocolos compatibles que integran, especificando parámetros tan básicos como la potencia de transmisión o el intervalo de emisión de paquetes.

Ambos fabricantes cuentan con muy buena documentación y extensos tutoriales para sacar el máximo partido a todas sus funcionalidades.

Se pueden ver capturas de sus aplicaciones en el Anexo 2.

2.2.1.1. Entorno de configuración y desarrollo Estimote

Estimote dispone de un panel web de configuración [45] con el que configurar remotamente los parámetros de nuestros *beacons*, editar los protocolos, habilitar opciones de seguridad, etc.

Su plataforma dispone de una API en la nube a la que poder acceder para ver las configuraciones de nuestros *beacons* o datos analíticos que hayamos configurado.

También cuenta con su propio SDK de desarrollo pero únicamente está pensado para entornos de programación relacionados con iOS (Objective C y Swift) y Android (Java). Dispone de plantillas programadas en esos tres lenguajes listas para descargar y ejecutar.

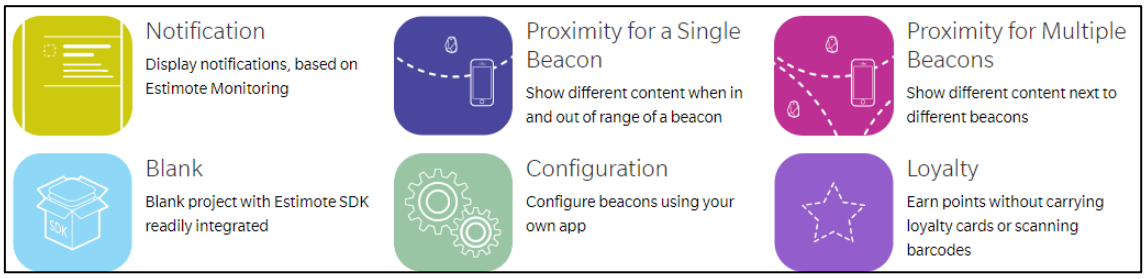


Fig. 2.4 Algunas de las plantillas para descargar en la nube de Estimote

En los primeros días en posesión de los *beacons* Estimote se hicieron unas pequeñas pruebas descargándose las tres primeras plantillas en formato java y ejecutándolas en Android Studio. Con muy poca configuración funcionaron en un dispositivo Android.

Si la idea es programar aplicaciones nativas de Android y/o iOS, estas plantillas son una buena base para juguetear con *beacons*. Además, Estimote tiene una gran comunidad de usuarios [46] que comparte ideas, proyectos y ayuda a resolver dudas de usuarios al más puro estilo StackOverflow [47].

2.2.2. Funciones extra y limitaciones

Como ya hemos comentado anteriormente, cada fabricante puede dotar de funcionalidades extra a sus *beacons*, ya sea implementando sensores al *firmware* o bien añadiendo características de *software*. Al mismo tiempo, algún fabricante puede decidir limitar algunos aspectos o ampliar otros. Todo esto, lo vemos a continuación.

2.2.2.1. Beacons Estimote

Una de las funcionalidades que implementa el fabricante viene dada gracias al sensor de movimiento que integra, el cual permite que al poner el *beacon* boca-abajo, éste deje de emitir paquetes de *advertising*. Esta funcionalidad es ideal para cuando se dispone de muchos *beacons* y se quieren configurar o cuando se está validando alguna implementación y no quieres que el envío de paquetes del resto afecte.

Recordemos que también integran un chip NFC. Este chip se puede configurar con el enlace a nuestra aplicación Android o iOS y, al acercar un dispositivo compatible con NFC, automáticamente se abre la aplicación o el enlace de la tienda de aplicaciones para descargarla.

Cabe comentar que con estos *beacons* únicamente se puede tener configurado un protocolo simultáneamente, es decir, un mismo *beacon* Estimote no puede estar emitiendo paquetes iBeacon y paquetes Eddystone-URL, por ejemplo. En el anexo (A2.1) se pueden ver algunas de éstas opciones de configuración.

2.2.2.2. Beacons Accent Systems

Lo que más destaca de la configuración de estos dispositivos es, sin duda, que uno puede convertirse en muchos a la vez. Un *beacon* de Accent Systems puede tener configurado a la vez hasta 2 paquetes diferentes con el protocolo iBeacon y hasta 4 con cualquiera de los 4 tipos del protocolo Eddystone. Esto nos permite, por ejemplo, crear diferentes áreas de alcance con un mismo *beacon* o emitir paquetes iBeacon a la vez que compartimos una URL con Eddystone.

Sin embargo, poder configurar tantos protocolos a la vez con un mismo *beacon* perjudica gravemente la duración de su batería, pero también han pensado en ello y le han añadido una funcionalidad que permite configurar las horas de encendido del *beacon*. Por ejemplo, podemos definir que el *beacon* únicamente entre en funcionamiento en el horario comercial, de 10:00h a 22:00h. Esta configuración se puede consultar en el anexo (A2.2).

Si se configura un mensaje iBeacon, el fabricante permite utilizar el byte sobrante del *payload* (iBeacon utiliza 30 de los 31 bytes disponibles) para enviar información sobre el estado de la batería del *beacon*.

Quizá el único inconveniente que les hemos encontrado, es que no tenemos una manera cómoda de parar momentáneamente su emisión a la hora de hacer pruebas, a no ser que le quitemos la pila.

CAPÍTULO 3. DISEÑO DE LOS SERVICIOS

Una vez vistos los productos con los que vamos a trabajar, es hora de ver cómo hemos creado una aplicación móvil que interactúa con ellos y proporciona servicios a alumnos, profesores y visitantes del campus [48].

En este capítulo veremos el tipo de aplicación que hemos decidido implementar y el por qué. Además, definiremos el entorno de trabajo que hemos empleado para el desarrollo de nuestra aplicación y, también veremos, qué estructura hemos seguido y qué proveedores externos hemos utilizado para hacerla funcionar. Por último, definiremos los servicios y/o funcionalidades que hemos acabado implementando en ella.

Todo el código de nuestra aplicación se puede consultar en GitHub [1].

3.1. Tipo de aplicación

La mayoría de servicios con *beacons* se diseñan para interactuar con los usuarios mediante una aplicación accesible desde sus *smartphones*. Se suelen definir tres tipos de aplicaciones móviles:

- Aplicaciones nativas: Son aplicaciones que se desarrollan de manera específica para un sistema operativo en concreto, utilizando para su programación el SDK que proporciona el fabricante de ese SO. Suelen ser más caras de desarrollar pero permiten exprimir al máximo todas las características del *hardware* de los dispositivos que las usan.

- Aplicaciones web: Son aplicaciones desarrolladas con lenguajes de programación muy conocidos como HTML, Javascript y CSS. Se pueden programar independientemente del sistema operativo en el que se vaya a utilizar puesto que se ejecutan en cualquier navegador web de cualquier sistema operativo a través de una URL. Muchas páginas web tradicionales están adaptadas a sistemas operativos móviles redistribuyendo la estructura y estilo según si se accede desde un navegador web de un *smartphone* o un navegador de ordenador.

- Aplicaciones híbridas: Aplicaciones resultantes de combinar las dos anteriores. Este tipo de aplicación se desarrolla con los lenguajes de programación de las aplicaciones web y, al finalizar la aplicación, éstas se transforman en aplicaciones nativas mediante un proceso de agrupación y transformación de código, listas para publicarse en las tiendas de aplicaciones de los sistemas operativos móviles.

A continuación, podemos ver una tabla comparativa entre ellas:

Tabla 3.1. Ventajas e inconvenientes de los principales tipos de aplicaciones que se pueden desarrollar.

	Aplicación Nativa	Aplicación Web	Aplicación Híbrida
VENTAJAS	• Acceso completo al dispositivo • Mejor experiencia de usuario • Envío de notificaciones • Publicación en *stores* nativas	• Código reutilizable para múltiples SO • Desarrollo sencillo y económico • Sin necesidad de aprobación para publicarla • Siempre actualizada	• Código reutilizable para múltiples SOs • Desarrollo sencillo y económico • Acceso a *hardware* del dispositivo • Envío de notificaciones • Publicación en *stores* nativas
INCONVENIENTES	• Código no reutilizable para otros SO • Desarrollo más caro y complejo • Necesidad de aprobación para publicarse • Actualizaciones más lentas	• Requiere conexión a Internet • Acceso limitado a elementos y características del *hardware* • Peor experiencia de usuario • No están en una *store* • Envío de notificaciones	• Necesidad de aprobación para publicarse • Actualizaciones más lentas • Retraso en obtención de últimas mejoras de los SO por no usar SDKs nativos

Se ha decidido desarrollar una aplicación híbrida para este proyecto por la mayor facilidad de programación que tiene y a la posibilidad de, con ese mismo código, poder generar aplicaciones nativas publicables en las tiendas de aplicaciones de Android e iOS.

3.1.1. Privacidad

Nuestra aplicación va a requerir que el usuario nos permita el acceso a la ubicación de su dispositivo. También, requiere que éste se registre facilitando datos personales como su correo electrónico, fecha de nacimiento, nombre y apellidos.

Este tipo de datos van a ser usados únicamente para mejorar los servicios que se le quieren ofrecer y no van a ser vendidos a empresas externas bajo ningún concepto. Esta aplicación está diseñada con fines demostrativos y no incorpora publicidad ni compras integradas.

Los *beacons* por sí solos no pueden estimar dónde se encuentra situado un usuario así que, si el usuario deshabilita la ubicación y el *bluetooth* de su dispositivo, ya no podrá ser rastreado.

3.2. Entorno de trabajo

Existen diferentes entornos de trabajo (*frameworks*) pensados para desarrollar aplicaciones híbridas pero uno de los más conocidos, sin duda, es Ionic [49]. Este *framework* se ha hecho muy popular por ofrecer unos diseños de interfaz de usuario al más puro estilo de una aplicación nativa y que se integran en el código de una manera muy sencilla.

Ionic se nutre de diferentes tecnologías y *frameworks* de código abierto para conseguir una hegemonía y sencillez muy útil para el desarrollador. Además, Ionic cuenta con una línea de comandos [50] (CLI) que simplifica mucho la creación de diferentes tipos de componentes dentro de nuestra aplicación y la ejecución de los procesos para transformar nuestro código web en una aplicación lista para publicar en Play Store o App Store.

Fig. 3.1 Estructura del entorno de trabajo Ionic

3.2.1. Lenguajes de programación

Ionic utiliza diferentes tecnologías web para construir su código:

- *TypeScript* [51]: Lenguaje de programación desarrollado por Microsoft que evoluciona de JavaScript [52] añadiendo tipado estático y objetos basados en clases, entre otros.

- *HTML5*: Última versión de este estándar a cargo de la W3C [53] que incorpora nuevos elementos, atributos y comportamientos permitiendo a los sitios web y aplicaciones ser más diversas y tener mayor alcance.

- *Sass* [54]: Lenguaje de hoja de estilos que dota de más poder de diseño al tradicional CSS, también a cargo de la W3C [53].

3.2.2. Tecnología

El SDK de Ionic es de código abierto y está definido sobre Angular [55], lo que permite dotarle de todas las bondades que nos ofrece este *framework* multiplataforma creado por Google y ampliamente utilizado por la mayoría de desarrolladores en la creación de páginas web Modelo-Vista-Controlador (MVC).

Ionic ofrece, además, una gran cantidad de *plugins* definidos en su documentación *Native* [56] que permiten acceder al *hardware* de los dispositivos móviles a los que irá dirigida la aplicación, como por ejemplo, la cámara, bluetooth, almacenamiento interno, etc. Estos *plugins* están definidos en TypeScript para que sean muy fáciles de integrar en nuestra aplicación. Podemos encontrar los *plugins* que hemos utilizado en nuestra aplicación y la utilidad que le hemos dado en el anexo (A1.1).

3.2.3. Conversión a aplicación híbrida

A partir de estos lenguajes de programación, de la interfaz de usuario del *framework* Ionic y la potencia de Angular, habríamos diseñado una aplicación web funcional.

Para convertir esta aplicación web en una híbrida, entran en juego Apache Cordova [57], que es un entorno de desarrollo de aplicaciones móviles que permite construir aplicaciones para dispositivos móviles utilizando CSS3, HTML5 y JavaScript en vez de las APIs específicas de cada plataforma nativa como Android o iOS.

Ionic se ayuda de este entorno para transformar todo el código de su aplicación y *plugins* definidos en *TypeScript* y *Sass* en *JavaScript* y *CSS* que son los lenguajes que entiende Cordova.

Cordova, una vez tiene todo el código en el formato correcto, se encarga de transformarlo en el lenguaje correspondiente para cada sistema operativo que precisemos, por ejemplo, Java para Android y Swift para iOS. Estas aplicaciones se ejecutarán como una aplicación nativa haciendo uso de las vistas web (*webview*) que proporciona el sistema operativo.

Se puede encontrar la guía de comandos necesarios para crear el archivo *apk* (Android) de nuestra aplicación para subirla al *Play Store* en el anexo (A3.4).

3.3. Servicios y funcionalidades de la aplicación

Durante el transcurso de este proyecto, se han ido debatiendo y estudiando posibles servicios que se podían ofrecer a los estudiantes y profesores que casi a diario frecuentan el campus del Baix Llobregat de la UPC. Veremos algunos de ellos a continuación de manera resumida.

3.3.1. Avisos en tiempo real

Este tipo de avisos consisten en mostrarle al usuario, mediante una notificación en su *Smartphone*, información de interés. Estas notificaciones únicamente funcionan si el usuario tiene la aplicación abierta o en segundo plano. Para este tipo de avisos se ha utilizado el *plugin* de notificaciones locales (A1.1.5).

3.3.1.1. Aviso de próxima clase

El alumno o profesor puede configurar de manera rápida y sencilla su lista de asignaturas de toda la semana especificando el número de aula, el edificio donde se encuentra, la hora de inicio y un campo de observaciones.

La aplicación notifica al usuario la información de la clase y edificio al que debe dirigirse de manera automática si se encuentra en la zona de cobertura de los *beacons* entre 20 minutos antes y 10 minutos después de la hora de inicio de alguna de las asignaturas guardadas. Código en el anexo (A1.4.3).

Esta funcionalidad permite al alumno o profesor despreocuparse de saber sus horarios y aulas de cada día de la semana.

3.3.1.2. Avisos manuales

El usuario puede generar avisos y/o recordatorios para que se le notifiquen al entrar o al salir de la facultad. Útil, por ejemplo, para hacer que recuerde que tiene que ir a hacer algún trámite en secretaría.

3.3.1.3. Notificación de eventos guardados

Se notificará al usuario de la información básica, la hora y el lugar de algún evento que se haya guardado en sus favoritos. Ayuda a no olvidarse de ir a un evento al que está interesado en asistir.

3.3.2. Visualización de noticias o eventos por proximidad

Cada *beacon* puede llevar asociada una noticia o evento que se mostrará al usuario en la pantalla de inicio al pasar cerca de éste. Un evento lleva asociado una localización y el día y hora de inicio.

3.3.3. Chats por proximidad

Un administrador puede crear un nuevo chat desde el apartado de administración y asociarlo a un *beacon*. Los usuarios que se encuentren cerca de éste pueden acceder a ese chat desde la pestaña '*beacons*' y hablar con el resto de usuarios que estén en la misma situación.

Esta funcionalidad es útil para permitir la comunicación de personas que no tienen por qué conocerse en entornos cerrados como puede ser una conferencia, una clase, etc.

3.3.4. Recuento de personas por proximidad

Esta funcionalidad fue pensada para permitir al profesorado tener una lista de los alumnos que están en clase en ese momento, sin necesidad de pasar lista, pero nos dimos cuenta que era extrapolable a otro tipo de usos como recuento de asistentes a eventos o recuento de personas que entran a la facultad.

En nuestra aplicación únicamente sirve para recuento de usuarios cercanos al *beacon* que se seleccione.

3.3.5. Seguimiento y solicitudes de asistencia

Los administradores de la aplicación pueden, desde el panel de administración de usuarios, dotar a un usuario de un permiso de 'seguimiento' que permite que dicho usuario pueda enviar alertas a los administradores.

Estos usuarios deben ser seleccionados en función de si tienen alguna necesidad de asistencia específica, como podría ser padecer alguna discapacidad. Los administradores que se encuentren en la región de *beacons* y tengan la aplicación abierta recibirán una notificación sobre la entrada de dicho usuario al campus para que puedan ir, según el grado de discapacidad, a ayudarle en lo que sea conveniente.

Esta funcionalidad también es extrapolable a otros usos similares como el de avisar a los administradores de que ha llegado al campus un conferenciante o persona importante y poder así ir a recibirlo.

3.3.6. Localización en interiores

La aplicación no tiene una funcionalidad de localización en interiores basada en *beacons* puesto que se determinó que no era factible debido a la inestabilidad en la precisión de las medidas de distancia entre *beacon* y usuario. Aun así, se ha creado una pestaña 'mapa' en la que ubica al usuario mediante el servicio de localización de Google y muestra la localización predefinida de los *beacons* desplegados y los eventos creados para que los usuarios puedan situarse y dirigirse a ellos de manera más fácil.

3.3.7. Notificaciones *Nearby*

Como hemos visto en el apartado 1.3.3.6. el protocolo Eddystone nos permite registrar los *beacons* con Google y configurar mensajes y avisos desde un tablero web creado para tal cometido (Guía en el anexo A3.3).

Este tipo de notificaciones únicamente funcionan con el sistema operativo de Android y no requieren de la instalación de nuestra aplicación para que el usuario las pueda recibir. El único requerimiento es que el usuario tenga activada la ubicación y el *Bluetooth* en su dispositivo puesto que Google integra ese servicio de manera nativa en su sistema operativo desde la versión 4.4.

Este tipo de notificaciones se pueden utilizar para difundir el enlace a *Play Store* de nuestra aplicación o difundir páginas web mediante una URL.

3.3.8. Notificaciones *push*

Se ha integrado en la aplicación un servicio de notificaciones mediante *Firebase* [58] que permite enviar notificaciones aunque el usuario tenga la aplicación totalmente cerrada (Guía en el anexo A3.1).

La aplicación está configurada para que cuando un usuario se registra, pueda elegir el tipo de perfil que es (alumno, profesor o visitante) y si pertenece a alguna facultad del campus (EETAC o ESAB). Con esos datos, podemos crear notificaciones segmentadas a grupos de usuarios más reducidos.

Este tipo de notificaciones son muy intrusivas para el usuario, por eso hay que utilizarlas en la menor medida posible. El uso principal para el que han sido pensadas es para utilizarse en casos de emergencia como avisos de incendios o notificar acontecimientos importantes. Así, todos se pueden enterar a la vez de dicho acontecimiento independientemente de donde se encuentren.

3.3.9. Ideas y servicios no integrados en la aplicación

Debido a los conocimientos de programación que tenemos y las limitaciones de tiempo inherentes a la realización de un proyecto, algunas ideas de servicios posibles para integrar en la aplicación no se han podido hacer realidad. Los vamos a comentar resumidamente puesto que son servicios que se podrían haber implementado.

3.3.9.1. Navegación en interiores con beacons

Ayudándonos con los *beacons* y los puntos de acceso Wi-Fi que hay repartidos por el campus se podría haber creado un servicio de posicionamiento de usuarios en interiores que permitiera guiarle entre los pasillos y edificios en tiempo real para hacerle llegar a su destino de una manera fácil y cómoda al más puro estilo *Google Maps*.

3.3.9.2. Encuestas en tiempo real

Del mismo modo que tenemos implementado chats en tiempo real que únicamente pueden ver los usuarios que estén cerca del *beacon* asociado a éste, se podría permitir el envío de encuestas puntuales.

Un profesor podría utilizarlo para hacer una pregunta sobre el temario a sus alumnos y que estos contestaran “Sí” o “No” desde la aplicación en vez del método tradicional de levantar la mano. También se podría utilizar para hacer breves cuestionarios sobre la conferencia a la que se acaba de asistir, etc.

3.3.9.3. Servicio para conocer gente por proximidad

Existen multitud de aplicaciones en el mercado para “conocer gente” y “hacer amigos” como Tinder o Badoo. Estas aplicaciones muestran a gente a tu alrededor que se encuentra entre unos pocos cientos de metros hasta muchos kilómetros, pero es prácticamente imposible coincidir con alguna de esas personas que se muestran en tiempo real y en una misma ubicación.

Un servicio útil del estilo gracias a los *beacons* consistiría en mostrar todos los usuarios que están en el rango de dicho *beacon*, que estaría colocado, por ejemplo, en la sala de un evento, y permitir al usuario seleccionar algún o algunos usuarios con los que estaría interesado en conocer. Si, por otro lado, estos usuarios también lo seleccionan a él se notificaría un “*match*” y éstos podrían establecer una conversación por un chat privado desde la misma aplicación.

El hecho de haberse “conocido” estando en la misma sala donde estaba situado el *beacon* permite que sea luego más fácil establecer una conversación o primer contacto en persona.

3.4. Desarrollo y programación de la aplicación

Como hemos decidido desarrollar una aplicación híbrida utilizando el *framework* de Ionic, vamos a ver qué necesitamos antes de empezar a programar. A continuación, veremos cómo se ha estructurado nuestra aplicación y qué carpetas y archivos contiene. Para acabar, vamos a profundizar en la funcionalidad de cada página desarrollada en la aplicación y qué servicios se utilizan para dar mayores funcionalidades. Mucha información extra de este apartado se puede encontrar en el Anexo 1.

3.4.1. Requisitos previos

Antes de empezar a programar con Ionic debemos tener instalados una serie de programas y complementos que conformaran nuestro entorno de desarrollo y nos permitirán un correcto funcionamiento de dicho *framework*. Los vemos a continuación:

- *NodeJs* [59] (obligatorio): entorno de ejecución del lado de servidor basado en eventos que básicamente se encarga de compilar y ejecutar el código JavaScript de manera muy rápida. Node Js cuenta también con uno de los gestores de paquetes más grandes del mundo, npm [60].

- *Android SDK* (obligatorio para crear aplicaciones Android): Conjunto de herramientas de desarrollo para crear aplicaciones para el sistema operativo Android. La forma más sencilla de descargarlo es instalando su IDE *Android Studio* [61], que lo trae incorporado.

- *Xcode* [62] (obligatorio para crear aplicaciones iOS): IDE para macOS que contiene un conjunto de herramientas creadas por Apple destinadas al desarrollo de *software* para sus dispositivos. No se puede programar en Ionic para iOS si no se hace desde un ordenador Apple debido a *Xcode* no puede ser instalado en Windows.

- *Ionic CLI* [50] (opcional pero muy recomendado): Mediante el gestor de paquetes NPM instalado anteriormente, podemos descargar la línea de comandos de Ionic que integra comandos de ejecución para compilar código, crear componentes, instalar *plugins…*

- *IDE* (opcional pero muy recomendado): Un entorno de desarrollo integrado consiste normalmente en un editor de código fuente que incorpora herramientas de construcción automática o autocompletado y un depurador. Es importante elegir un IDE que tenga soporte para los lenguajes de programación con los que vamos a trabajar. Algunos de los más utilizados para Ionic porqué incorporan funcionalidades para *TypeScript* son los siguientes: *Visual Studio Code* [63], *Sublime Text* [64], *Atom* [65] y *Webstorm* [66].

Como sólo vamos a desarrollar la aplicación para Android, hemos instalado el SDK descargándonos *Android Studio* y hemos utilizado como entorno de desarrollo el IDE de *Atom* [65] por la sencillez de su interfaz y las herramientas de completado de código que incorpora.

3.4.2. Estructura de la aplicación

Ionic tiene predefinidas algunas plantillas de base como la de pestañas (*tabs*) o la de menú lateral (*sidemenu*). Para el diseño de nuestra aplicación, hemos decidido utilizar la plantilla de pestañas por su simplicidad y facilidad de uso de cara al usuario.

Al ejecutar el comando de creación de la plantilla, Ionic crea un proyecto base con una estructura de archivos y carpetas concreta. Lo vemos en detalle a continuación.

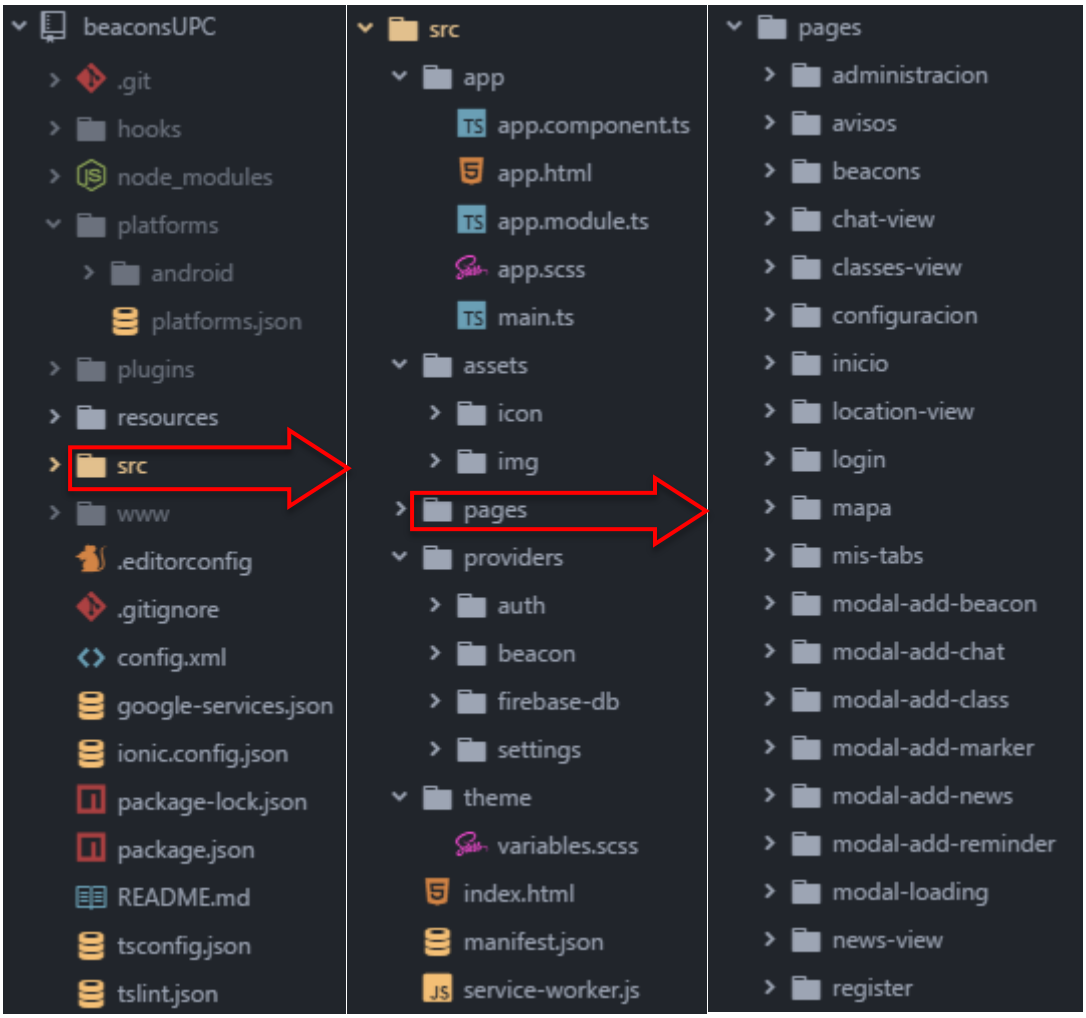


Fig. 3.2 Estructura de la aplicación beaconsUPC

3.4.3. Carpetas y archivos

Ionic nos genera una estructura de carpetas y archivos que, en conjunto, hace que la aplicación pueda compilar y ejecutarse correctamente. Veamos en detalle su función:

- Carpeta *hooks*: Contiene scripts que se ejecutan en el proceso de construcción de la aplicación. No es necesario modificar nada.

- Carpeta *node_modules*: Se genera automáticamente al instalar dependencias con el gestor de paquetes NPM [60]. Los módulos se definen en el archivo *package.json*.

- Carpeta *platforms*: Aquí se generan los proyectos nativos que hayamos añadido. Dentro de estos proyectos, están todos los archivos necesarios para ejecutar la aplicación en ese sistema operativo y generar el correspondiente archivo instalable en el dispositivo final.

- Carpeta *plugins*: Contiene los archivos de los *plugins* de Cordova (que son los de Ionic *Native*) que hayamos añadido a nuestro proyecto. Se puede consultar los *plugins* utilizados en este proyecto y su funcionamiento en el anexo (A1.1).

- Carpeta *resources*: Contiene los tamaños de imagen del icono de nuestra aplicación y de la pantalla de bienvenida de las plataformas añadidas. Estos archivos se pueden generar de manera automática con un comando de la CLI de Ionic [67].

- Carpeta *src*: Aquí está todo el contenido de nuestra aplicación y es donde trabajaremos el 90% del tiempo. En su interior, encontramos las siguientes carpetas:

    - Carpeta *app*: Aquí podemos encontrar los archivos principales de nuestra aplicación. El archivo *app.components.ts* es el primer archivo que se ejecuta al iniciar la aplicación y nos sirve para activar nuestros servicios (*providers*) y redireccionar a una u otra página en función de si hemos o no iniciado sesión. En *app.module.ts*, están definidos todos los módulos y proveedores importados para poder utilizarlos cuando los necesitemos.

    - Carpeta *assets*: Esta carpeta se suele utilizar para almacenar archivos y/o imágenes que se utilizan en la aplicación. Aquí almacenamos algunas imágenes por defecto que utiliza la aplicación y los marcadores de colores que se utilizan en el mapa.

    - Carpeta *pages*: Aquí están definidas todas las páginas o vistas que hemos creado para nuestra aplicación. Las veremos en detalle en el siguiente punto.

    - Carpeta *providers*: Aquí hemos definido nuestros servicios de autenticación, las funciones del *plugin* de detección de *beacons*, los accesos a la base de datos de *Firebase* y otras configuraciones. Se puede consultar la estructuración de nuestras bases de datos en el anexo (A1.5).

    - Archivo *variables.scss*: Archivo que contiene la configuración del diseño principal de la aplicación. Aquí están definidos los colores que se pueden usar dentro de la aplicación y variables *Sass* que hayamos definido para diferentes componentes de la aplicación.

    - Archivo *index.html*: Vista *html* que define algunos *plugins* y donde se engloba el resto de páginas de la aplicación.

- Carpeta *www*: Esta carpeta se genera automáticamente y contiene la versión actual del código cada vez que se efectúa algún cambio. No hay que modificar nada en ella.

- Archivo *config.xml*: Contiene información básica sobre la configuración de nuestro proyecto. Es importante cumplimentar bien esta información.

3.4.4. Páginas

En la aplicación se han creado muchas páginas, algunas de ellas principales y otras modales, que se superponen a la principal hasta que ésta es cerrada por el usuario). A continuación, vamos a ver las que hay y su funcionalidad. Se pueden consultar todas las capturas de imagen en el anexo (A1.3).

- Pantalla de inicio de sesión (Cap. **1**): Permite al usuario iniciar sesión con su cuenta ya registrada o acceder a la pestaña de registro. También se puede solicitar la restauración de contraseña si ésta ha sido olvidada.

    - Pantalla de registro de usuarios (Cap. **1**): Permite rellenar los datos para crearse una cuenta en la aplicación.

- Pestaña inicio (Cap. **2**): Ésta es la pantalla que aparece cuando el usuario inicia sesión. En ella se muestran las noticias asociadas a los *beacons* que detecta y avisos que puedan publicar los administradores.

    - Vista web (Cap. **2**): Al hacer clic en ‘más info’ se abre el enlace.

- Pestaña avisos: Compuesta por tres vistas:

    - Vista de asignaturas (Cap. **3**): Se visualiza la lista de asignaturas guardadas por el usuario cada día de la semana. Desde aquí, se puede acceder a la página de creación de clases y a la lista de configuración de alguno de los días concretos.

        - Ventana para crear, editar o eliminar una clase (Cap. **3**): Una clase se compone de los siguientes campos: asignatura, día de la semana, hora de inicio, número de aula, edificio y un campo opcional de observaciones. Si todos los campos están rellenados, se muestra una vista previa de la notificación que se le mostraría al usuario.

        - Ventana de configuración de clases (Cap. **3**): Muestra la lista de clases guardadas ese día concreto de la semana y permite al usuario eliminarlas o editarlas.

  - Vista de recordatorios (Cap. **4**): Aparece la lista de recordatorios que ha configurado el usuario. Deslizando un recordatorio de la lista hacia la izquierda, aparece el botón de eliminar recordatorio.

    - Ventana de configuración de recordatorios: Permite a un usuario crear un recordatorio definiendo título, descripción, si se debe ejecutar al entrar o al salir de la facultad y si se debe repetir una única vez o a diario.

  - Vista de noticias/eventos guardados (Cap. **4**): Muestra la lista de noticias o eventos que el usuario ha guardado. Deslizando hacia la izquierda, aparece el botón para eliminar. Deslizando hacia la derecha, aparece el botón para ver la vista previa.

    - Ventana de vista previa de noticia/evento (Cap. **4**): Muestra cómo se vería la noticia o evento en la pantalla inicio.

- Pestaña mapa (Cap. **5**): Al entrar a dicha pestaña, se muestra un mapa centrado en la ubicación aproximada del usuario (marcador azul). El mapa se carga con unos marcadores que muestran la posición de los *beacons* (marcadores negros) y los eventos creados por los administradores (marcadores rojos).

- Pestaña beacons (Cap. **5**): Muestra el listado de *beacons* que detecta el dispositivo a su alrededor ofreciendo información básica de éstos y una estimación de la distancia a la que se encuentra de ellos. Cada *beacon* dispone de tres botones de acceso directo: uno es para visualizar la vista previa de la noticia que tienen asociada; otro para visualizar en el mapa la localización donde está colocado; y, un último, para acceder al chat que tengan asociado.

  - Ventana de visualización en mapa (Cap. **2**): Muestra un mapa centrado en una ubicación concreta según se haya elegido.

  - Ventana de chat (Cap. **5**): Permite al usuario que ha entrado en esta ventana enviar mensajes a través del chat.

- Pestaña ajustes (Cap. **6**): Muestra ajustes básicos para el usuario como el botón de cerrar sesión o eliminar cuenta. Los administradores tienen en esta pestaña el botón de acceso al panel de administración.

- Panel de administración: En este panel, aparecen 5 pestañas y un botón desplegable que nos permite acceder a las ventanas de creación de *beacons*, noticias, ubicaciones o chats.

- Pestaña beacons (Cap. **6**): Muestra el listado de *beacons* que tenemos configurados en la aplicación. Deslizando un elemento de la lista hacia la izquierda o derecha, se puede eliminar o editar.

    - Ventana de creación o edición de un *beacon* (Cap. **6**): Permite crear un *beacon* especificando la región y sus valores *Major* y *Minor* igual que tiene el *beacon* real. Desde esta ventana, se le puede asociar al *beacon* una noticia o evento, su marcador de posición y un chat para que muestre esos datos cuando el usuario lo detecte desde la pestaña 'beacons'.

- Pestaña noticias/eventos (Cap. **7**): Muestra un listado de noticias y eventos creados por los administradores.

    - Ventana de creación o edición de una noticia (Cap. **7**): Para una noticia o evento, los campos comunes a rellenar son título, descripción, color de fondo, URL de la imagen y enlace web. Si se selecciona evento, se debe rellenar también la fecha y hora y la ubicación.

- Pestaña ubicaciones (Cap. **8**): Muestra la lista de ubicaciones creadas por los administradores. Deslizando hacia la izquierda, se puede eliminar y, hacia la derecha, se puede acceder a la ventana de creación o edición de éstas.

    - Ventana de creación o edición de ubicaciones: Para definir una ubicación, se debe completar un título, la latitud y la longitud de las coordenadas de esa localización.

- Pestaña chat (Cap. **8**): Muestra la lista de chats creados por los administradores. Desde esa lista, se pueden eliminar los chats o únicamente los mensajes escritos en él. También se puede activar o desactivar el chat.

    - Ventana de creación o edición de un chat: Aparecen los campos de título, descripción y si queremos que esté activo o inactivo.

- Pestaña usuarios (Cap. **8**): Muestra la lista de usuarios que están registrados en nuestra aplicación junto a algunos de sus datos principales. En amarillo, aparecen los usuarios que son administradores y, en azul, los que están en modo seguimiento. Desde esa pestaña se gestionan los permisos de usuario.

3.4.5. Proveedor de servicios Firebase

Como hemos comentado en muchas partes de este documento, hemos utilizado muchas de las funcionalidades que nos proporciona *Firebase* [68].

Este servicio de Google tiene multitud de funcionalidades para que las aplicaciones móviles saquen el máximo partido posible de una manera sencilla de implementar. A continuación, definiremos los servicios que hemos utilizado de *Firebase* y el uso que les hemos dado:

- *Firebase Analytics*: Solución gratuita de análisis de parámetros y eventos en aplicaciones que proporciona información sobre el uso de las mismas y la participación del usuario en ellas. En nuestro caso, la hemos utilizado principalmente para añadir perfiles de usuario. Desde el panel de *Firebase Analytics*, podemos ver estadísticas de registros de usuarios, inicios de sesión, etc. Este servicio será útil consultarlo una vez esté la aplicación funcionando al 100% y con una gran comunidad de usuarios utilizándola.

- *Firebase Auth*: Servicio de autenticación de usuarios que permite implementar registros de usuario a partir de proveedores como Facebook, Twitter y Google. Nosotros hemos utilizado para la aplicación el registro mediante correo electrónico. Se puede consultar un ejemplo del código implementado en la aplicación sobre este servicio en el anexo (A1.4.2).

- *Firebase Notifications*: Servicio gratuito de notificaciones personalizadas. Nos permite enviar notificaciones personalizadas a los usuarios de nuestra aplicación de manera segmentada y aunque éstos no tengan la aplicación abierta. En el anexo (A3.1) se definen los pasos para enviar estas notificaciones desde el panel de *Firebase*.

- *Firebase Realtime Database*: Servicio de base de datos y *backend* en tiempo real. Nos permite sincronizar los datos de la aplicación y almacenarlos en la nube. Este tipo de base de datos noSQL permite crear de manera dinámica y muy sencilla toda su estructura. Además, nos ha permitido crear un servicio de chat en tiempo real. Se puede ver un ejemplo de código en el anexo (A1.4.1). Toda la documentación sobre la estructura que hemos seguido para definir nuestras tablas de la base de datos está detallada en el anexo (A1.5).

Firebase tiene una muy buena documentación sobre todos sus servicios para diferentes entornos de programación. La implementación de algunos de estos servicios ha sido relativamente sencilla gracias, en parte, a algunos tutoriales que hemos encontrado en la web de ReviBlog [69].

CAPÍTULO 4. DESPLIEGUE Y VALIDACIÓN

En este último capítulo, vamos a detallar cómo hemos configurado nuestros *beacons*, dónde los hemos colocado (ver Anexo 4) y cómo han ido las pruebas de validación de los servicios más importantes integrados en la aplicación *beaconsUPC*, disponible en *Google Play* [48].

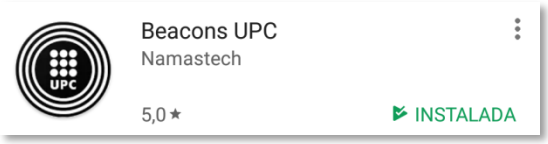


Fig. 4.1 Aplicación beaconsUPC en Google Play

4.1. Configuración de los beacons

El *plugin* que hemos utilizado para nuestra aplicación (A1.1.1) únicamente detecta paquetes *advertising* en formato *iBeacon*, por ese motivo debemos configurar nuestros *beacons* con ese protocolo visto en el apartado 1.3.1.

Necesitamos definir un identificador único (UUID) para nuestros *beacons* y una estructura lógica para el uso del campo *Major* y *Minor*.

Para generar nuestro identificador único hemos utilizado una web [70] que los genera de manera aleatoria. La repartición de parámetros quedaría de la siguiente manera:

Tabla 4.1. Parámetros de configuración para cada *beacon*

UUID CBL: 6a1a5d49-a1bd-4ae8-bdcb-f2ee498e609a					
***Beacons* Fijos**			***Beacons* Móviles**		
Nombre	**Major**	**Minor**	**Nombre**	**Major**	**Minor**
iBKS 1	0001	0001	Purple	0002	0001
iBKS 2	0001	0002	Green	0002	0002
iBKS 3	0001	0003	Blue	0002	0003
iBKS 4	0001	0004			
iBKS 5	0001	0005			

Está configuración está pensada únicamente para poner *beacons* en el Campus Baix Llobregat. Sería posible extrapolarlo a todos los campus o escuelas de la UPC utilizando, por ejemplo, el *Major* para definir la escuela concreta y el *Minor* para definir cada *beacon*. Recordemos que tanto el *Major* como el *Minor* disponen de 4 dígitos, lo que permite configurar 65536 combinaciones diferentes en cada uno, así que hay margen más que suficiente hasta para todas las universidades del mundo.

Los *beacons* fijos se han configurado a máxima potencia para intentar llegar al máximo de plantas y/o salas posibles y los móviles se han configurado para emitir a una distancia máxima de 15 metros y así asegurarnos que solamente emiten dentro de una sala o clase. La configuración más detallada de los *beacons* mediante sus aplicaciones propias se puede encontrar en el Anexo 2.

4.2. Colocación de los beacons

Puesto que disponíamos de pocos *beacons*, hemos optado por colocar los 5 iBKS en puntos donde la cobertura fuera máxima dentro de los tres edificios más frecuentados por estudiantes y profesores del campus (ver Fig. 4.3).

- Biblioteca CBL: se ha colocado un *beacon* en la sala central de la Biblioteca, encima de la pantalla de televisión informativa que hay, a una altura de unos 5 metros (Ver anexo A4.1). Esa posición tan elevada y la ausencia de techo entre las plantas permite prácticamente cubrir al 100% todas las zonas comunes de la biblioteca. Lamentablemente la zona del bar, situada en la planta -1 de ese edificio, no está cubierta.

- ESAB: Este edificio únicamente dispone de una entrada, por ese motivo se ha decidido instalar un *beacon* en el *hall*, a unos 4 metros de altura (Ver anexo A4.2). Esa posición permite dar cobertura a los dos pasillos principales y al pequeño patio que hay delante de la puerta.

- EETAC: Este edificio cuenta con dos entradas, una de ellas doble que permite entrar a las tres torres de docencia o a la torre de despachos de profesores. La parte docente del edificio cuenta con tres torres diferenciadas por un color: rojo, amarillo y azul. Debido a la gran dificultad de dar cobertura a las zonas de entrada al centro y a esas torres, se ha decidido colocar tres *beacons*.

  El primero de ellos se ha colocado en la entrada más cercana a la biblioteca, por la que suelen entrar los estudiantes y/o profesores que llegan en tren a la facultad (Ver anexo A4.3.1). Ese *beacon*, además, da cobertura a la escalera de entrada a la torre roja.

  El segundo se ha colocado entre la torre azul y la amarilla, encima de la máquina de café (Ver anexo A4.3.2). Es el lugar más abierto de todo el pasillo y permite dar cobertura a las entradas a dichas torres.

  El tercero se ha colocado en el *hall* de entrada a la torre de profesores (Ver anexo A4.3.3). La gran cristalera que envuelve esa sala permite dar cobertura a la doble entrada tanto del edificio de estudiantes como el de profesores.

Los otros tres *beacons* de la marca Estimote que se han configurado como *beacons* móviles permanecerán guardados en la conserjería de la EETAC y podrán ser prestados a estudiantes o profesores que deseen utilizar alguna de las funcionalidades para las que han sido pensados.

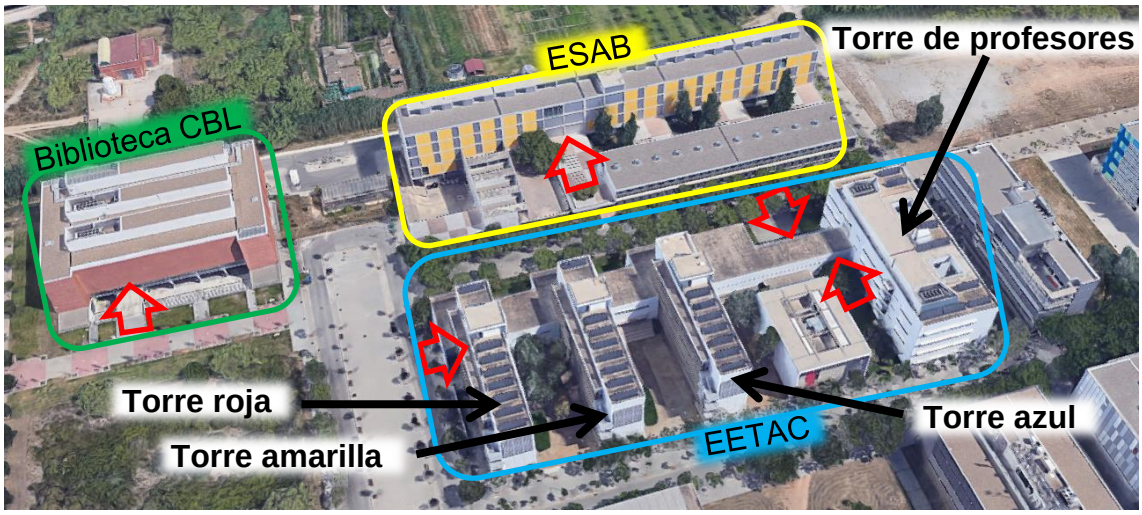


Fig. 4.2 Entradas a los edificios del Campus Baix Llobregat a los que hemos instalado *beacons* y distribución de las torres en la EETAC

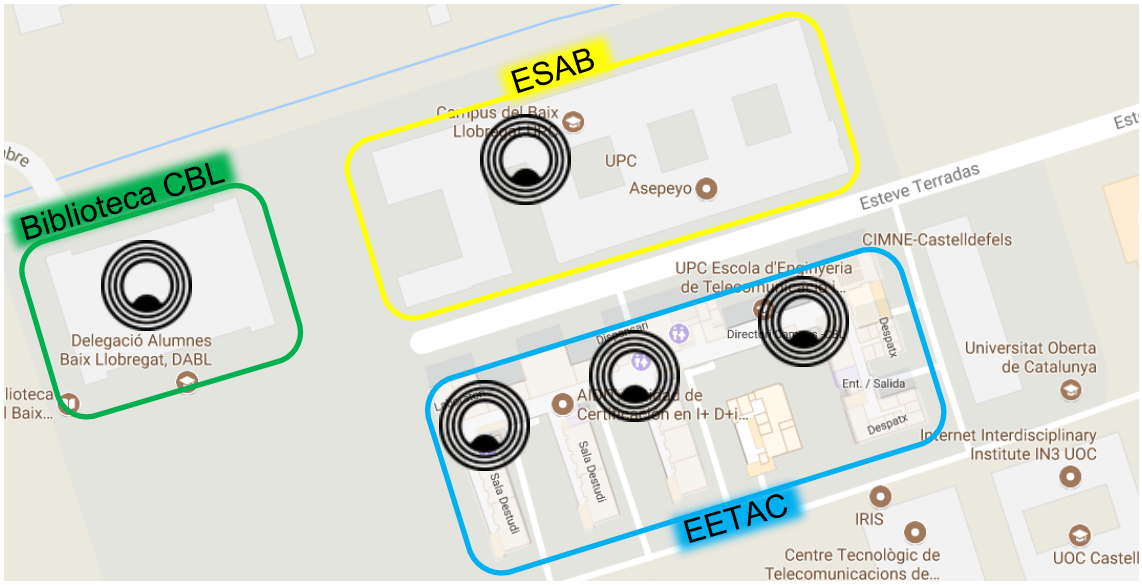


Fig. 4.3 Puntos de colocación de los *beacons* en los edificios del Campus. Fuente Google Maps [71]

Idealmente, para dar un correcto servicio, debería haberse colocado un *beacon* en cada planta de cada torre o, incluso, dentro de cada aula del edificio. Lamentablemente no disponíamos de tantos *beacons* y nos hemos tenido que adaptar asumiendo que la mayoría de zonas no se podrán detectar los mensajes de *advertising* de nuestros *beacons* y se pueden generar falsas salidas/entradas a nuestra región.

4.3. Validación del funcionamiento de la aplicación

Una vez instalados los *beacons* y terminada la primera versión operativa de la aplicación (v0.0.3), disponible en Play Store [48], se ha procedido a comprobar su correcto funcionamiento haciendo un simulacro de llegada al campus y entrada a la EETAC:

Al aparcar el coche en el parking del campus y bajarnos de él, nos ha llegado la primera notificación. Esta notificación se produce gracias a la geovalla que hemos definido alrededor de todo el campus (A1.1.2) e incentiva al usuario a abrir la aplicación para utilizarla o dejarla funcionando en segundo plano.

Al acercarnos a la puerta principal de la EETAC hemos recibido la segunda notificación que nos indica la clase y el color del edificio al que debemos dirigirnos. Esta notificación se ha producido al detectar la entrada del usuario en la región de nuestros *beacons*.

La tercera notificación la hemos generado nosotros, los administradores, desde el panel de notificaciones de *Firebase* para verificar el correcto funcionamiento de las notificaciones *push* (ver anexo A3.1).

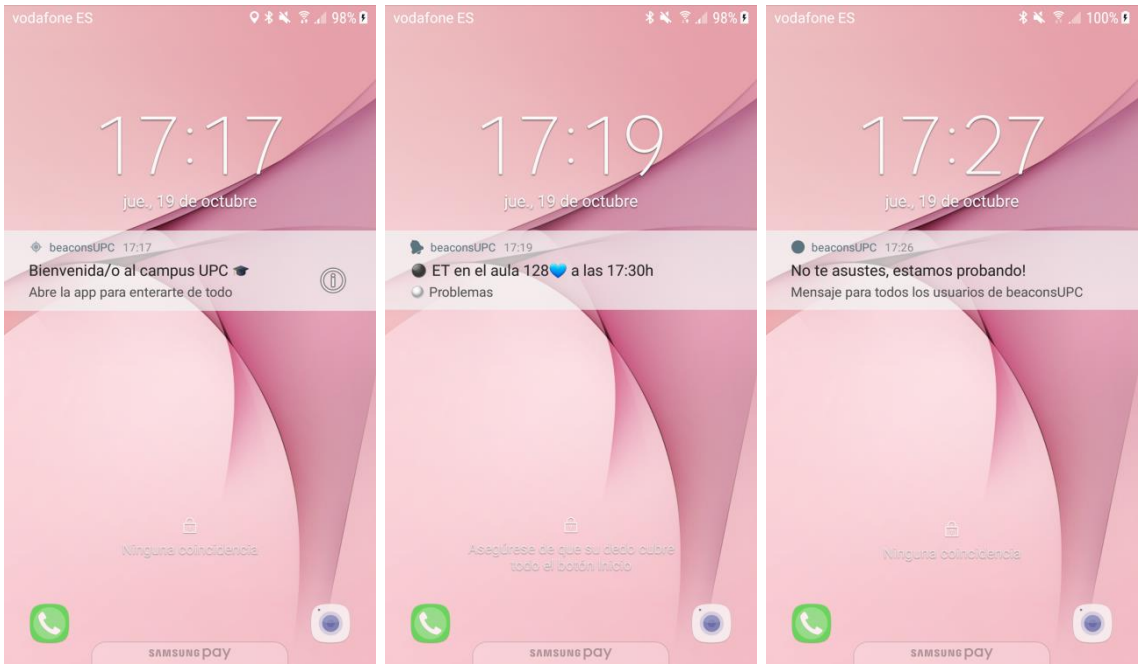


Fig. 4.4 Notificación por entrada en geovalla (izquierda), notificación automática de la aplicación (centro) y notificación *push* (derecha)

Por otro lado hemos probado el correcto funcionamiento de la visualización de noticias en la pantalla principal en función del *beacon* más cercano y ha salido según lo previsto. Cabe recalcar que la demora entre que se detecta un nuevo *beacon* cercano y se muestra la noticia asociada a éste, pueden pasar hasta 10 segundos para evitar las imprecisiones en las distancias con los *beacons*.

Al entrar en la pestaña de mapa, éste se nos abre situándose en nuestra ubicación actual y mostrando las localizaciones de los *beacons* y eventos cercanos.

Desde la pestaña *beacons* hemos seleccionado el chat más cercano y hemos podido establecer una pequeña conversación con una usuaria que estaba conectada en ese momento.

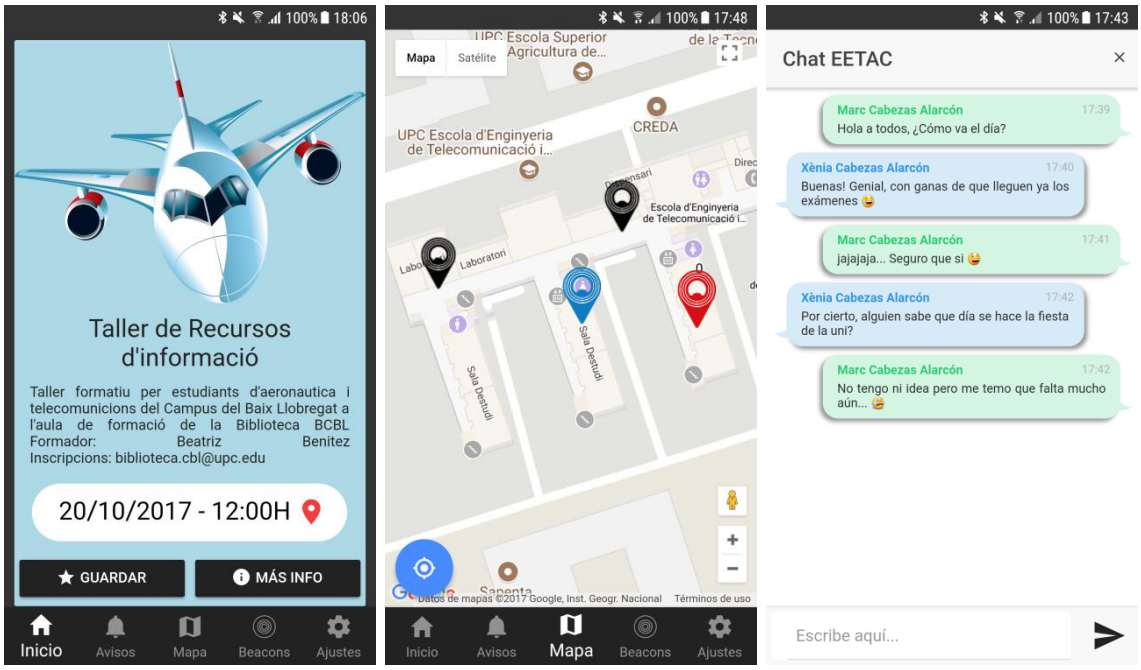


Fig. 4.5 Vista de noticia en pantalla de inicio (izquierda), vista de pantalla mapa (centro) y vista de un chat asociado a un *beacon* (derecha)

Cabe destacar que al principio del desarrollo de la aplicación nos encontramos con una dificultad que hacía peligrar el correcto desarrollo del proyecto: la aplicación únicamente detectava *beacons* cada 30 segundos, haciendo inviable el despliegue de servicios de proximidad habiendo tanta demora entre un escaneo y el siguiente. Por suerte, tras una intensiva búsqueda se pudo dar solución y el desarrollo de beaconsUPC pudo continuar. La descripción de la problemática y la solución se puede encontrar más en detalle en el anexo (A1.2).

CONCLUSIONES

El objetivo final del trabajo era desarrollar una aplicación que se ayudara de la tecnología de los *beacons* para dar servicios a la comunidad educativa del Campus del Baix Llobregat. Esa aplicación es ya una realidad y está disponible de forma gratuita en *Google Play* [48].

Para que nuestra aplicación fuera operativa, se han tenido que superar algunos problemas. Destacar el hecho de no poder interactuar con los *beacons* mediante la aplicación si el usuario no la tenía abierta o ejecutándose en segundo plano. Muchos de los servicios de aviso que nuestra aplicación da, no eran capaces de ejecutarse con la aplicación cerrada y quizá desprestigiaban un poco la utilidad real de éstos. Para superar este problema, hemos buscado maneras de incentivar a los usuarios a abrir la *app* al llegar al campus con otro tipo de notificaciones que no dependen directamente de nuestra aplicación.

Otra limitación inherente a esta tecnología es que el usuario debe llevar activadas la ubicación y el *Bluetooth* en su dispositivo. A día de hoy, por la mala percepción que se tiene de su consumo de batería, la mayoría de usuarios suelen llevarlos desactivados. Aquí se trata de fomentar un cambio de hábitos mediante el ofrecimiento de servicios útiles. Ésta ha sido una de las motivaciones que nos ha guiado en el momento de proponerlos.

También se ha cubierto el objetivo de documentar la tecnología que utilizan estos dispositivos (ver Capítulo 1). La tecnología base en la que se basan los *beacons* ya es, a día de hoy, suficiente potente para apostar por ella. La evolución de la tecnología irá de la mano de posibles mejoras en la integración con los sistemas operativos móviles actuales, pero no tanto en el *hardware*. La problemática principal de que esta tecnología no acabe de despegar viene dada por el hecho de que desarrollar una aplicación móvil de calidad y desplegar una cantidad importante de *beacons*, puede ser una inversión importante en primera instancia. Por otro lado, no suele haber un beneficio económico a corto plazo para la empresa que invierte en esta tecnología.

La dificultad está en saber proponer servicios que aun sabiendo que no aportarán un beneficio económico directo, lo aportará al usuario que la va a utilizar. De hecho, que nuestros clientes instalen en sus dispositivos móviles una aplicación, nos abre un canal de comunicación con ellos muy potente.

Líneas futuras

La aplicación desarrollada no deja de ser una buena base susceptible de mejora mediante la implementación de más servicios, algunos ya comentados en esta memoria, como el de localización y direccionamiento en interiores. En el caso que nos ha ocupado, sería interesante incrementar la cantidad de *beacons* colocados para poder cubrir el 100% del total de la superficie del Campus y evitar así falsas salidas de región que se producen actualmente por la pérdida de la señal en zonas de los edificios.

Bibliografía

[1] GitHub de beaconsUPC: https://github.com/piskitus/beaconsUPC.

[2] Reporte de Aislelabs: https://www.aislelabs.com/reports/beacon-guide/.

[3] Microcontroladores BLE de Texas Instruments: http://www.ti.com/lsds/ti/wireless-connectivity/bluetooth-low-energy/products.page.

[4] Microcontroladores BLE de Nordic Semiconductor: https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy.

[5] Especificación del nuevo Bluetooth 5: https://www.bluetooth.com/specifications/bluetooth-core-specification/bluetooth5.

[6] Web oficial de Bluetooth SIG: https://www.bluetooth.com/.

[7] Especificación principal de Bluetooth: https://www.bluetooth.com/specifications/bluetooth-core-specification.

[8] B. SIG, Especificación del sistema Bluetooth v4.2, 2014.

[9] Web de Argenox: http://www.argenox.com/a-ble-advertising-primer/.

[10] Números asignados en el GAP de Bluetooth: https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile.

[11] «Identificador de compañías del SIG: https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers.

[12] Radius Network: https://www.radiusnetworks.com/.

[13] Nearby Notifications: https://developers.google.com/nearby/notifications/overview.

[14] Physical Web de Google: https://google.github.io/physical-web/.

[15] UUIDs de 16 bits asignados a miembros de Bluetooth SIG: https://www.bluetooth.com/specifications/assigned-numbers/16-bit-uuids-for-members.

[16] Protocolo UriBeacon en el cual se basó Eddystone-URL: https://github.com/google/uribeacon.

[17] Eddystone-URL: https://github.com/google/eddystone/tree/master/eddystone-url.

[18] Acortador de URLs de Google: https://goo.gl/.

[19] Especificación para la generación del EID de Eddystone: https://github.com/google/eddystone/blob/master/eddystone-eid/eid-computation.md.

[20] Servicios GATT: https://www.bluetooth.com/specifications/gatt/services.

[21] Aplicación de Google Beacon Tools: https://play.google.com/store/apps/details?id=com.google.android.apps.location.beacon.beacontools.

[22] Notificaciones Nearby: https://developers.google.com/nearby/notifications/overview.

[23] Android Geovallas: https://developer.android.com/training/location/geofencing.html.

[24] Casos de exito en implementaciones con beacons: http://www.usingbeacons.com/category/casos-de-exito/.

[25] Pueba piloto con beacons en el Condis de Horta: https://www.youtube.com/watch?v=68jyZG8f24Q.

[26] Black Friday en Macy's utilizando Beacons: http://www.retaildive.com/ex/mobilecommercedaily/how-macys-maximized-in-store-traffic-through-beacons-during-black-friday.

[27] Aplicación Macy's en Play Store: https://play.google.com/store/apps/details?id=com.macys.android.

[28] Tecnología de Beacons en el aeropuerto de Barcelona - El prat: http://www.aena.es/csee/Satellite?Language=ES_ES&SiteName=BCN&c=Page&cid=1045569607459&pagename=Comunes%2FPopUpAvisosAeropuertos.

[29] Aplicación Aena: https://play.google.com/store/apps/details?id=es.aena.mobile.

[30] Beacons en el museo Guggenheim de NYC: http://blog.estimote.com/post/157200820650/the-icon-of-modern-art-puts-estimote-beacons-on.

[31] VolksWagen Connect: https://accent-systems.com/project/volkswagen-connect/.

[32] Samsonite Tack&Go: https://accent-systems.com/project/samsonite-introduces-trackgo/.

[33] Comprar beacons Estimote: http://estimote.com/#get-beacons.

[34] Comprar beacons Accent Systems: https://accent-systems.com/es/producto/ibks-105/.

[35] Beacon iBKS 105 de Accent Systems: https://accent-systems.com/es/producto/ibks-105/?gclid=CjwKEAjwgvfOBRD7_IDSuP3znTwSJAB4_t6Gk0gQYuTFdMz0n-56gLzmlq5Wn45NRgrllah5EhXiqRoCDAHw_wcB.

[36] Protocolo iBeacon: https://developer.apple.com/ibeacon/.

[37] Protocolo Eddystone: https://developers.google.com/beacons/eddystone.

[38] Microcontrolador nRF51822 de Nordic Semiconductor: https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51822.

[39] Bluetooth LE en modo emisión (Broadcast): https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/le-broadcast.

[40] Pack de pilas CR2477 con Amazon Prime: https://www.amazon.es/Panasonic-CR2477-bater%C3%ADa-unidades-10-pilas/dp/B01JR3D106/ref=sr_1_6?ie=UTF8&qid=1507642165&sr=8-6&keywords=cr2477.

[41] Aplicación Estimote para Android: https://play.google.com/store/apps/details?id=com.estimote.apps.main.

[42] Aplicación IBKS para beacons Accent Systems en Play Store:

https://play.google.com/store/apps/details?id=com.accent_systems.ibks_config_tool.
[43] Aplicación Estimote para iOS: https://itunes.apple.com/es/app/estimote/id686915066?mt=8.
[44] Aplicación IBKS para beacons Accent Systems en App Store: https://itunes.apple.com/es/app/ibks-config-tool/id929525388?mt=8.
[45] Panel Web de configuracion de Estimote: https://cloud.estimote.com/.
[46] Comunidad Estimote: https://community.estimote.com/hc/en-us.
[47] Web de StackOverflow: https://stackoverflow.com/.
[48] Aplicación beaconsUPC en Play Store: https://play.google.com/store/apps/details?id=com.beaconsUPC.android.
[49] Ionic Framwork: https://ionicframework.com/.
[50] Ionic CLI: https://ionicframework.com/docs/cli/.
[51] TypeScript: https://www.typescriptlang.org/.
[52] JavaScript: https://www.javascript.com/.
[53] Consorcio World Wide Web: https://www.w3c.es/.
[54] Sass: http://sass-lang.com/.
[55] Angular: https://angular.io/.
[56] Ionic Native: https://ionicframework.com/docs/native/.
[57] Apache Cordova: https://cordova.apache.org/.
[58] Web de Firebase: https://firebase.google.com/.
[59] NodeJs: https://nodejs.org/es/.
[60] Gestor de paquetes NPM: https://www.npmjs.com/.
[61] Descargar Android Studio: https://developer.android.com/studio/index.html.
[62] Descargar Xcode para MacOS: https://developer.apple.com/xcode/.
[63] Visual Studio Code: https://code.visualstudio.com/.
[64] Sublime Text: https://www.sublimetext.com/].
[65] Atom: https://atom.io/.
[66] Webstorm: https://www.jetbrains.com/webstorm/.
[67] Generar archivos de la carpeta Resources: https://ionicframework.com/docs/cli/cordova/resources/.
[68] Web de Firebase: https://firebase.google.com/.
[69] Web Reviblog: https://reviblog.net/.
[70] Generador de UUIDs: https://www.uuidgenerator.net/.
[71] Google Maps: https://www.google.es/maps.

ANEXOS

Anexo 1. Aplicación beaconsUPC

A1.1. Plugins utilizados en la aplicación

A1.1.1. IBEACON

Ionic Native: https://ionicframework.com/docs/native/ibeacon/

Repositorio plugin: https://github.com/petermetz/cordova-plugin-ibeacon

Descripción: *Plugin* que provee funciones para trabajar con iBeacons. Tiene funciones tanto para *monitoring* como para *ranging* de *beacons*.

Uso en el proyecto: Se ha definido un proveedor en el proyecto (*beacon.ts*) que ejecuta las diferentes funcionalidades del *plugin* y puede ser llamado desde cualquier página del proyecto.

Finalidad: La aplicación depende prácticamente por completo de este *plugin* para su correcto funcionamiento ya que es imprescindible poder detectar *beacons* para que la aplicación haga uso de la gran mayoría de sus funcionalidades como la distribución de noticias o eventos por proximidad, la notificación de número de aula de la próxima clase, entre otras.

Ejemplo de funcionamiento:

```
// define la región a inspeccionar (nombre de la Región, UUID)
let beaconRegion = this.ibeacon.BeaconRegion('UPC', '6a1a5d49-a1bd-4ae8-bdcb-
f2ee498e609a');

// Inicia el monitoreo de la región
this.ibeacon.startMonitoringForRegion(beaconRegion)
  .then(
    () => console.log('Native layer recieved the request to monitoring'),
    error => console.error('Native layer failed to monitoring: ', error)
  );

// crea un Nuevo delegado y lo registra con la capa nativa
let delegate = this.ibeacon.Delegate();

// ahora ya nos podemos suscribir a algunos manejadores de eventos definidos
por el plugin
delegate.didEnterRegion()
  .subscribe// esta función se ejecuta cada vez que se detecta la entrada a
la región
    data => {
      console.log('didEnterRegionUUID:       ',      data.region.identifier);
      // aquí añadiríamos las llamadas a las funciones que queremos que se
ejecuten cuando entramos en la región de beacons
    }
  );
```

Reporte de fallos tras su uso: Hubo una gran problemática con el *plugin* en los comienzos de desarrollo de la aplicación por un bug que no entendíamos por qué pasaba y que finalmente pudimos solventar aplicando ciertas medidas. La problemática, explicación y solución están explicadas en el apartado A1.2 del anexo.

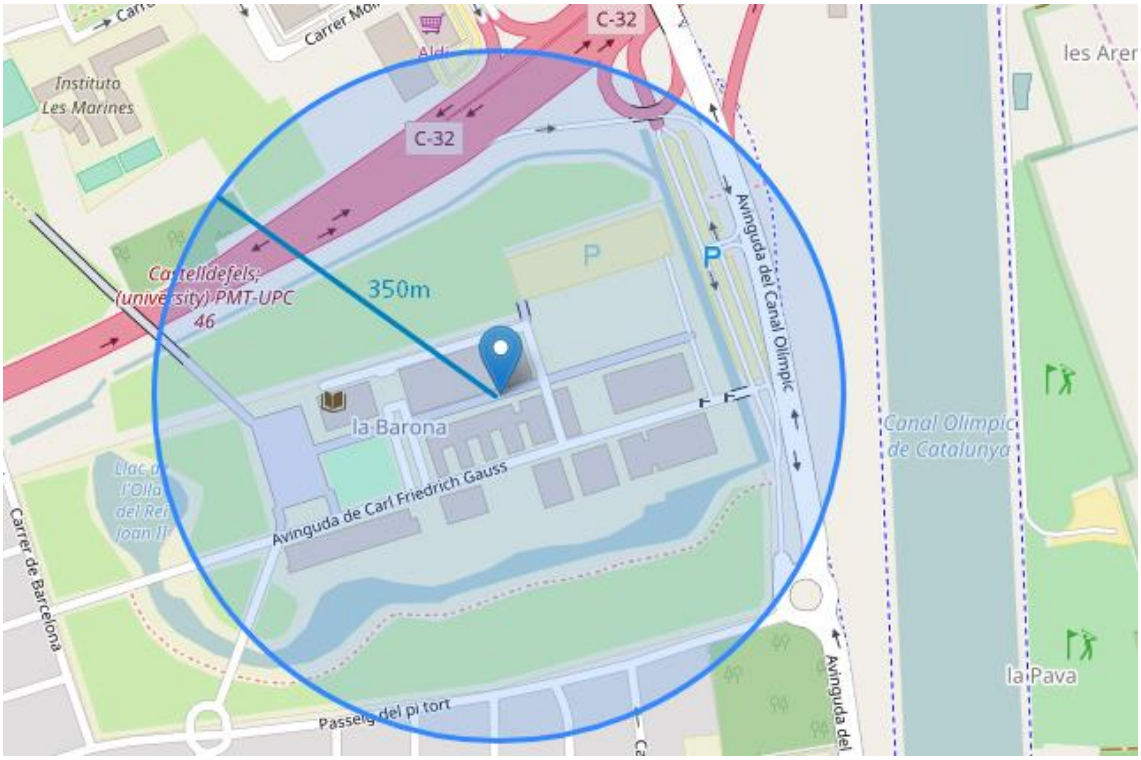
A1.1.2. GEOFENCE

Ionic Native: https://ionicframework.com/docs/native/geofence/

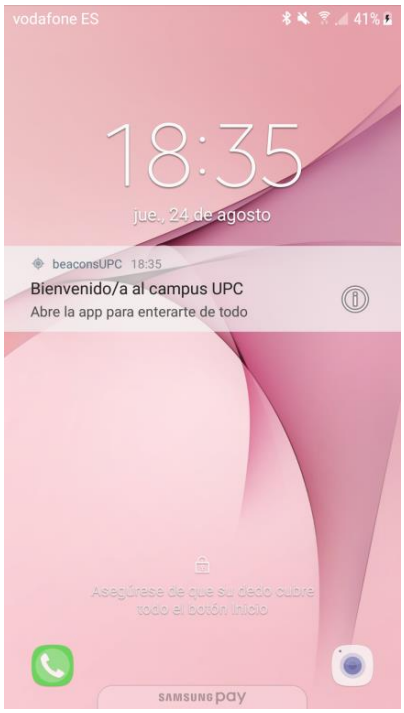
Repositorio plugin: https://github.com/cowbell/cordova-plugin-geofence

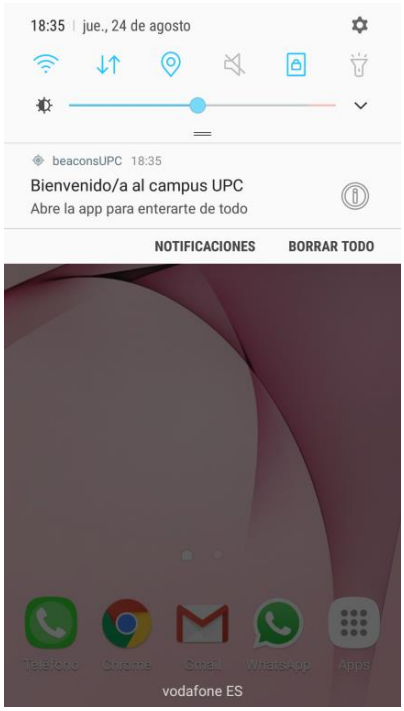
Descripción: Permite monitorizar las entradas y/o salidas de un usuario a un perímetro virtual circular de un área geográfica del mundo real (*geofence*) produciendo el envío de una notificación al entrar y/o salir de esa geovalla. Este monitoreo funciona aunque la aplicación se haya cerrado completamente o el dispositivo se haya reiniciado. Este *plugin* permite definir hasta 100 geovallas en Android y 20 en iOS.

Uso en el proyecto: Para este proyecto se ha creado una geovalla alrededor del campus UPC de Castelldefels con un radio de 350m. Se ha definido el envío de notificación únicamente al entrar a la geovalla para no resultar intrusivos. El radio elegido es más grande de lo necesario puesto que en las pruebas de validación se detectaron falsas entradas y salidas causando notificaciones redundantes y molestas.



Finalidad: Que el usuario que tenga instalada la aplicación en su móvil, al llegar al campus (entrada a la geovalla) reciba una notificación que le invite a abrir la aplicación para que se ejecute en primer o segundo plano, ya que la detección de *beacons* no funciona con la aplicación totalmente cerrada.





Ejemplo de funcionamiento:

```
// definición de una geovalla
let fence = {
    id: 6a1a5d49-a1bd-4ae8-bdcb-f2ee498e609a', //ID único (utilizo UPC UUID)
    latitude:       41.275737, //centro del radio de la geovalla
    longitude:      - 1.986996,
    radius:         350, //radius de la geovalla en metros
    transitionType: 1, // Tipo de transición (1: entrar, 2: salir, 3: ambas)
    notification: { //configuración de notificación
        id:      1, //ID de notificación único
        title:   ' Bienvenida/o al campus UPC, //título de notificación
        text:    ' Abre la app para enterarte de todo', //subtítulo notif.
        openAppOnClick: true // si es true, abre la app al hacer click en la
notificación
    }
```

Reporte de fallos tras su uso: Hemos detectado que en algunos sistemas operativos como Android 7 el sistema elimina el proceso tras el paso de algunas horas. El proceso se restablece tras el reinicio del dispositivo.

A1.1.3. IN APP BROWSER

Ionic Native: https://ionicframework.com/docs/native/in-app-browser/

Repositorio plugin: https://github.com/apache/cordova-plugin-inappbrowser

Descripción: Permite abrir una ventana de navegación directamente desde la aplicación, pasándole como parámetros la URL a abrir y algunas opciones que queramos configurar.

Uso en el proyecto: Para abrir los links a las noticias o eventos que tengan asociados una URL.

Ejemplo de funcionamiento:

```
// función que recibe una URL y la abre en la app
openURL(newsURL){
const browser = this.iab.create(newsURL); // se abre vista con esa URL
}
```

Finalidad: Dotar a los usuarios de la posibilidad de ampliar la infomación de la noticia o evento que detectan en su teléfono a partir del *beacon* más cercano.

A1.1.4. DIAGNOSTIC

Ionic Native: https://ionicframework.com/docs/native/diagnostic/

Repositorio plugin: https://github.com/dpa99c/cordova-diagnostic-plugin

Descripción: Puede comprobar si algunas características de *hardware* del dispositivo como cámara, GPS, bluetooth… están habilitadas o disponibles para la aplicación.

Uso en el proyecto: Utilizado en la aplicación para verificar que GPS y bluetooth estén activos mientras se utiliza la aplicación.

Finalidad: Que la aplicación pueda estar constantemente inspeccionando el entorno para encontrar *beacons*. Bluetooth y GPS activos son un requerimiento para que la aplicación pueda detectarlos.

Ejemplo de funcionamiento:

```
this.diagnostic.getBluetoothState()
  .then((state) => {
    if (state == this.diagnostic.bluetoothState.POWERED_ON){
      // si está activado no hago nada
    } else {
      // llamo a la función para activar Bluetooth
    }
  }).catch(e => console.error(e));
```

A1.1.5. LOCAL NOTIFICATIONS

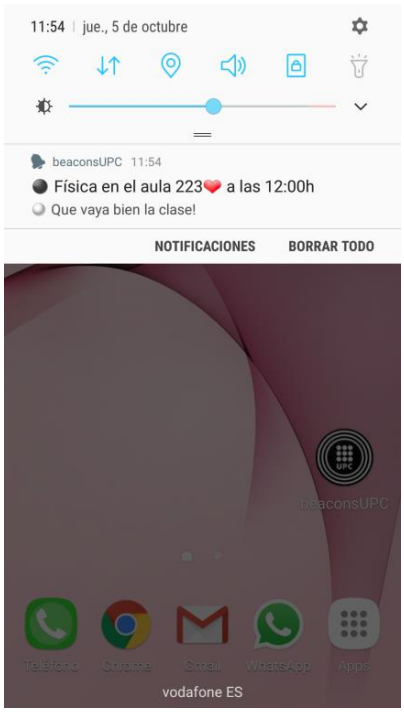
Ionic Native: https://ionicframework.com/docs/native/local-notifications/

Repositorio plugin: https://github.com/katzer/cordova-plugin-local-notifications

Descripción: *Plugin* que permite mostrar notificaciones locales en el dispositivo del usuario.

Uso en el proyecto: Se utiliza en las funcionalidades de información de la clase a la que debe dirigirse, avisos pre-programados, etc. Estas notificaciones son ejecutables automáticamente con la aplicación en primer plano o en modo *background*. Si la aplicación está cerrada, no se puede notificar.

Finalidad: Informar de manera poco intrusiva y en tiempo real de información útil para el usuario.



Ejemplo de funcionamiento:

```
// función que recibe un id, un título y un texto (descripción) y ejecuta una notificación local con esos parámetros

setLocalNotification(id, title, text){
 this.localNotifications.schedule({
  id: id,
  title: title,
  text: text
 });
}
```

A1.1.6. STATUS BAR

Ionic Native: https://ionicframework.com/docs/native/status-bar/

Repositorio plugin: https://github.com/apache/cordova-plugin-statusbar

Descripción: *Plugin* para administrar la apariencia de la barra de estado nativa.

Uso en el proyecto: Se utiliza para ocultar la barra en algunas vistas y para cambiar el color de la misma para integrarse en el entorno de colores de la aplicación.

Finalidad: Que no resulte molesta para el usuario pero pueda seguir ofreciéndole información de notificaciones y del estado de sus conexiones.

Ejemplo de funcionamiento:

```
// pongo la barra de estado del color de fondo de mi app
this.statusBar.backgroundColorByHexString('#333');

// oculto la barra de estado
this.statusBar.hide();
```

A1.1.7. FIREBASE ANALYTICS

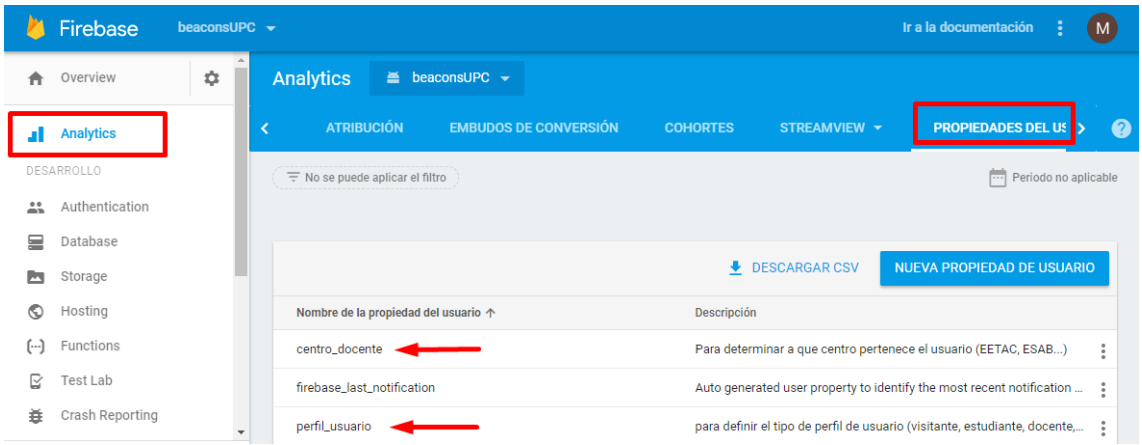
Ionic Native: https://ionicframework.com/docs/native/firebase-analytics/

Repositorio plugin: https://github.com/chemerisuk/cordova-plugin-firebase-analytics

Descripción: *Plugin* para el uso de *Firebase Analytics*.

Uso en el proyecto: Se utiliza enviar propiedades del usuario a *Firebase*.

Finalidad: Gracias a esas propiedades de usuario que se añaden (centro docente y perfil de usuario), podemos hacer notificaciones *push* de manera segmentada. El detalle de notificaciones *push* está explicado en el anexo (A3.1)



Ejemplo de funcionamiento:

```
//añade el perfil y la escuela a las propiedades de usuario de Firebase
setUserAnalytics(profile, school){
this.firebaseAnalytics.setUserProperty('perfil_usuario', profile);
this.firebaseAnalytics.setUserProperty('centro_docente', school);
}
```

A1.2. Problemática encontrada en Android N y solución aplicada

Cuando ejecuté por primera vez el *plugin* para detectar iBeacons en mi *Smartphone*, vi como hacía una búsqueda de beacons mediante bluetooth cada 1 segundo aproximadamente, tal y como especifica que hace el *plugin*.

Mi sorpresa fue que ese escaneo únicamente me mostraba datos de beacons encontrados durante 5 segundos y luego dejaba de detectar durante unos 20. En consecuencia, mi frecuencia de obtención de datos real era de unos 20 segundos, algo intolerable para una aplicación basada en la detección rápida de beacons para ejecutar funciones.

Empecé a investigar a ciegas y prácticamente no encontraba a casi nadie que tuviera el mismo problema que yo, y alguno que encontraba aún no había obtenido una respuesta de algún usuario que le solucionara el problema.

Finalmente, me di cuenta que sólo me pasaba en mi móvil, un S7 de Samsung con sistema operativo Android v7.0. Los otros *smartphones* con los que lo probé tenían todos versiones inferiores.

A partir de ahí, focalicé mi búsqueda en torno a los posibles cambios en términos de *bluetooth* de la versión 6 a la 7 y acabé encontrando que, en esta nueva versión, únicamente se podían hacer 5 escaneos de *bluetooth* cada 30 segundos.

Para solucionar esa problemática, configuré el *plugin* de detección de iBeacons de la aplicación para que realizara un escaneo cada 5 segundos, en vez de cada segundo. Ahora la aplicación funciona de manera constante pero tarda más en detectar los posibles cambios.

Estas son las dos líneas de código, introducidas dentro del archivo *config.xml* que permiten configurar el *plugin* para que escanee 5 segundos:

```
<preference
name="com.unarin.cordova.beacon.android.altbeacon.ForegroundBetweenScanPeriod
" value="5000" />
<preference
name="com.unarin.cordova.beacon.android.altbeacon.ForegroundScanPeriod"
value="5000" />
```

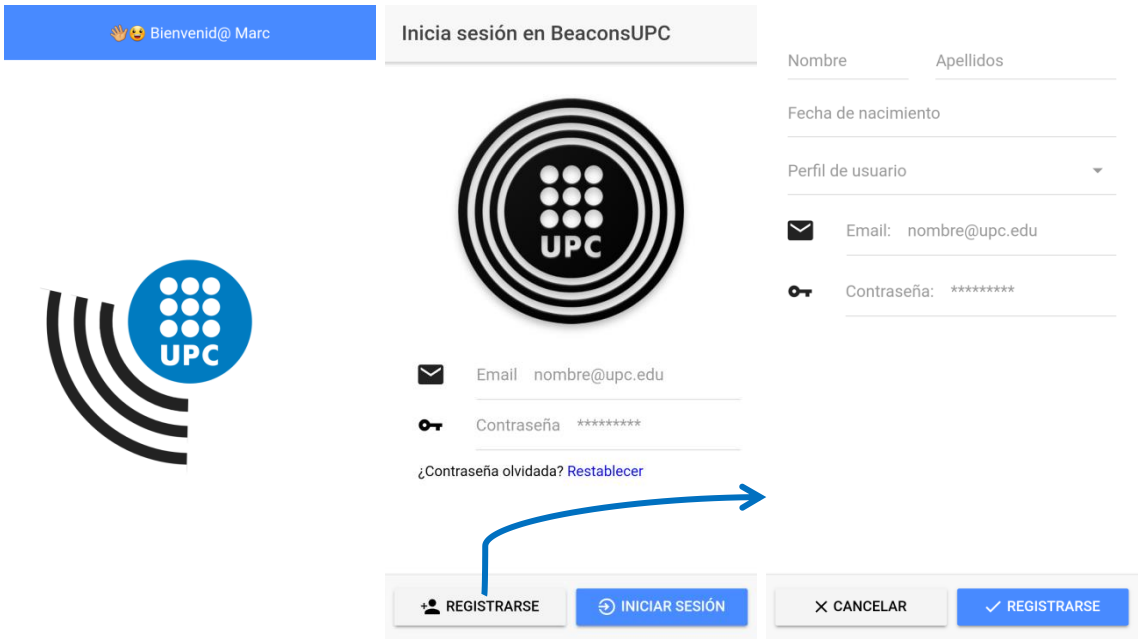
También podemos ver una clara representación en escala de tiempos de cómo funcionaba antes de encontrar el problema y después de solucionarlo. Se han utilizado únicamente los 3 *beacons* de Estimote para esta prueba.

Podemos apreciar también que el escaneo de *beacons* no es siempre efectivo, encontrando, a veces, menos *beacons* de los que realmente hay en el entorno. En la gráfica, vemos que hay veces que no se detectaban los 3 *beacons* que teníamos al lado.

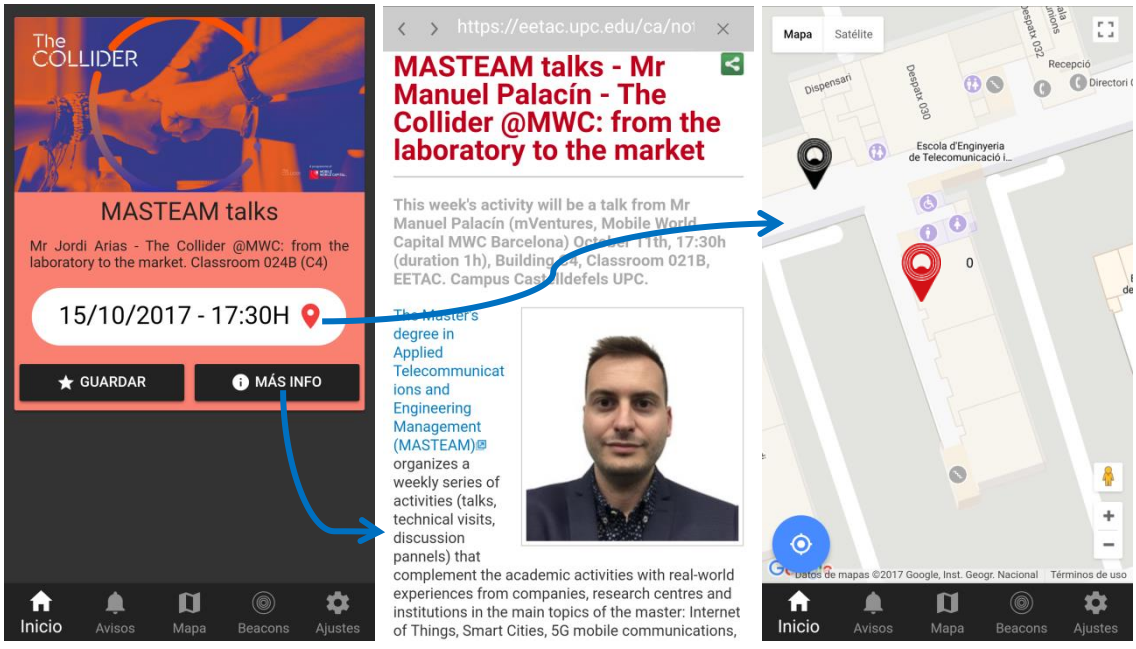
Configuración por defecto (1s)	Línea del tiempo	Configuración escaneo cada 5s
Entrada en la región		Entrada en la región
26:21.908Z Detected Beacons: 3	← INICIO CONTADOR →	41:19.114Z Detected Beacons: 3
26:23.227Z Detected Beacons: 3		41:25.201Z Detected Beacons: 3
26:24.534Z Detected Beacons: 1		41:31.339Z Detected Beacons: 3
26:25.820Z Detected Beacons: 3		41:37.422Z Detected Beacons: 3
26:27.094Z Detected Beacons: 2		41:43.554Z Detected Beacons: 3
26:28.383Z Detected Beacons: 0	30 segundos→	41:49.634Z Detected Beacons: 3
26:29.647Z Detected Beacons: 0		41:55.751Z Detected Beacons: 3
26:30.945Z Detected Beacons: 0		42:01.871Z Detected Beacons: 3
26:32.274Z Detected Beacons: 0		42:07.938Z Detected Beacons: 3
26:33.537Z Detected Beacons: 0		42:14.047Z Detected Beacons: 3
26:34.846Z Detected Beacons: 0	60 segundos→	42:20.147Z Detected Beacons: 3
26:36.128Z Detected Beacons: 0		42:26.233Z Detected Beacons: 2
Salida de la región	←Falsa salida	42:32.330Z Detected Beacons: 2
26:37.437Z Detected Beacons: 0		42:38.447Z Detected Beacons: 2
26:38.695Z Detected Beacons: 0		42:44.547Z Detected Beacons: 2
26:39.986Z Detected Beacons: 0	90 segundos→	42:50.635Z Detected Beacons: 2
26:41.286Z Detected Beacons: 0		42:56.706Z Detected Beacons: 2
26:42.568Z Detected Beacons: 0		43:02.827Z Detected Beacons: 3
26:43.856Z Detected Beacons: 0		43:08.943Z Detected Beacons: 2
26:45.132Z Detected Beacons: 0		43:14.999Z Detected Beacons: 3
26:46.417Z Detected Beacons: 0	120 segundos→	43:21.118Z Detected Beacons: 3
26:47.697Z Detected Beacons: 0		[…]
26:48.957Z Detected Beacons: 0		
26:50.257Z Detected Beacons: 0		
26:51.537Z Detected Beacons: 0	←30 segundos	
Entrada en la región		
26:52.855Z Detected Beacons: 3		
26:54.145Z Detected Beacons: 3		
26:55.406Z Detected Beacons: 2		
26:56.677Z Detected Beacons: 3		
26:57.989Z Detected Beacons: 2		
26:59.256Z Detected Beacons: 0		
27:00.546Z Detected Beacons: 0		
27:01.777Z Detected Beacons: 0		
27:03.030Z Detected Beacons: 0		
27:04.344Z Detected Beacons: 0		
27:05.629Z Detected Beacons: 0		
27:06.897Z Detected Beacons: 0		
Salida de la región	←Falsa salida	
27:08.248Z Detected Beacons: 0		
27:09.499Z Detected Beacons: 0		
27:10.764Z Detected Beacons: 0		
27:12.026Z Detected Beacons: 0		
27:13.336Z Detected Beacons: 0		
27:14.607Z Detected Beacons: 0		
27:15.914Z Detected Beacons: 0		
27:17.184Z Detected Beacons: 0		
27:18.454Z Detected Beacons: 0		
27:19.695Z Detected Beacons: 0		
27:20.996Z Detected Beacons: 0		
27:22.287Z Detected Beacons: 0		
27:23.544Z Detected Beacons: 0	←60 segundos	
Entrada en la Region		
[…]		

A1.3. Capturas de las páginas y vistas de la aplicación

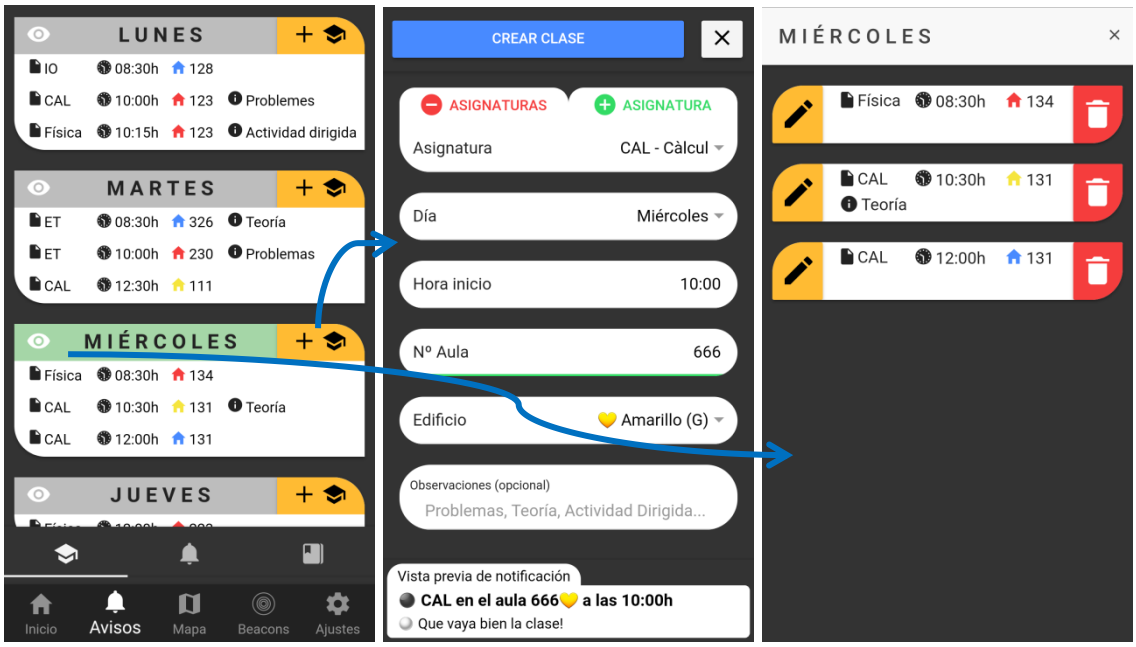
Las capturas de pantalla que se muestran a continuación corresponden a la de la versión 0.0.3. Puede haber cambios de diseño en versiones posteriores.



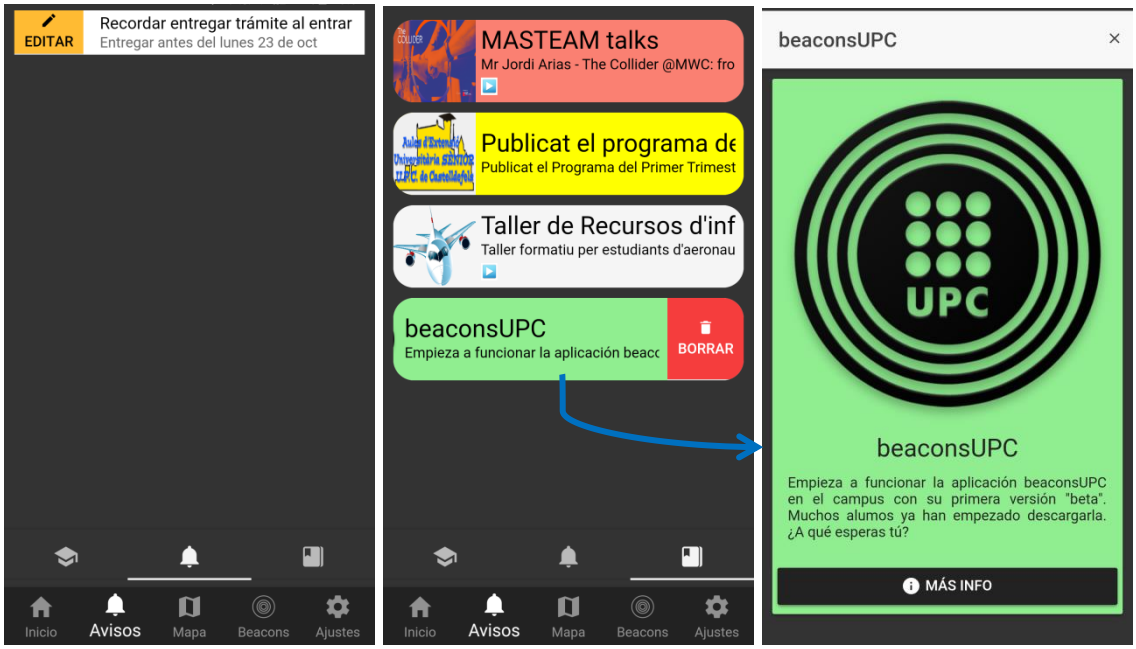
Cap. 1 Ventana de carga de la app (izquierda), pantalla de inicio de sesión (centro) y pantalla de registro (derecha)



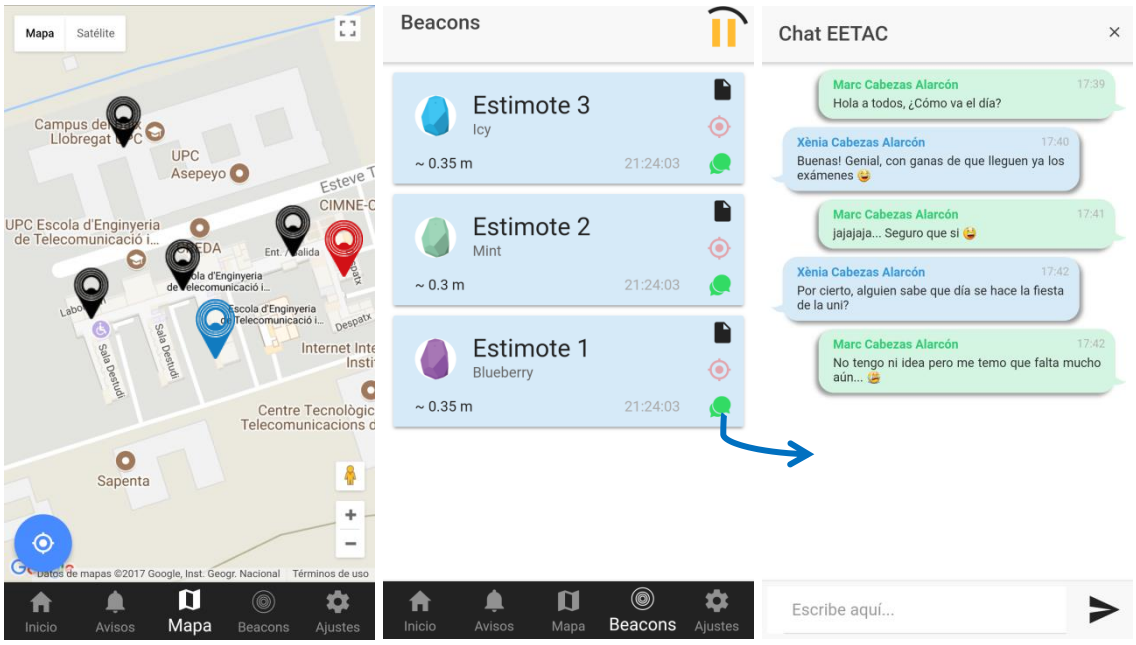
Cap. 2 Pestaña de inicio visualizando un evento (izquierda), vista web de la información del evento (centro), vista en el mapa de la localización del evento (derecha)



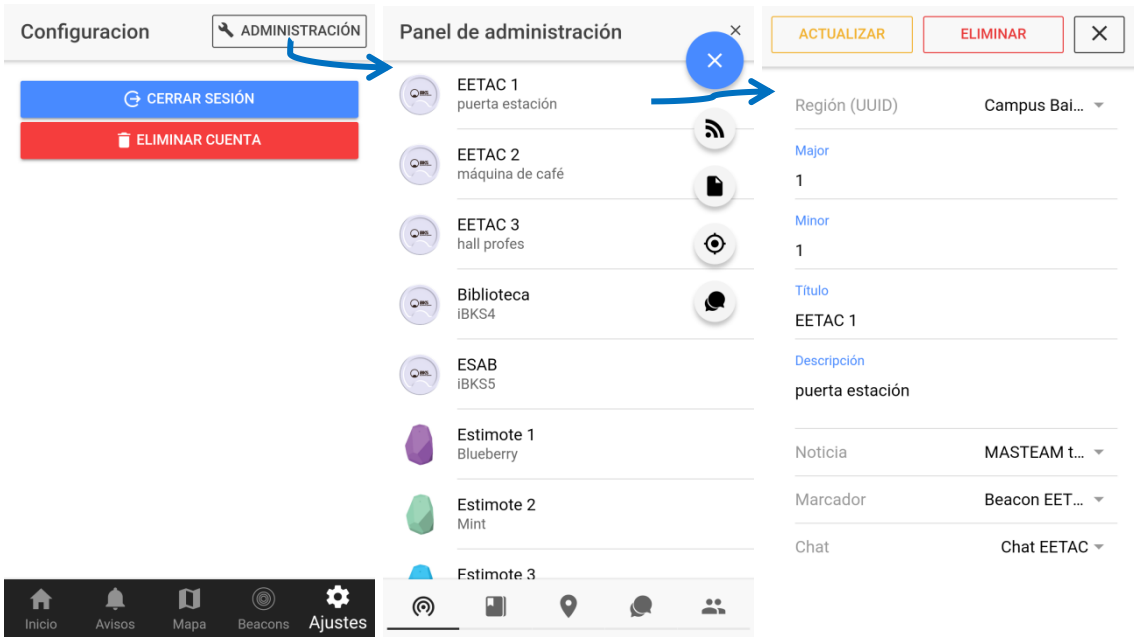
Cap. 3 Vista de asignaturas de la pestaña Avisos (izquierda), ventana para crear o editar una clase (centro), ventana de configuración de clases para un día de la semana (derecha)



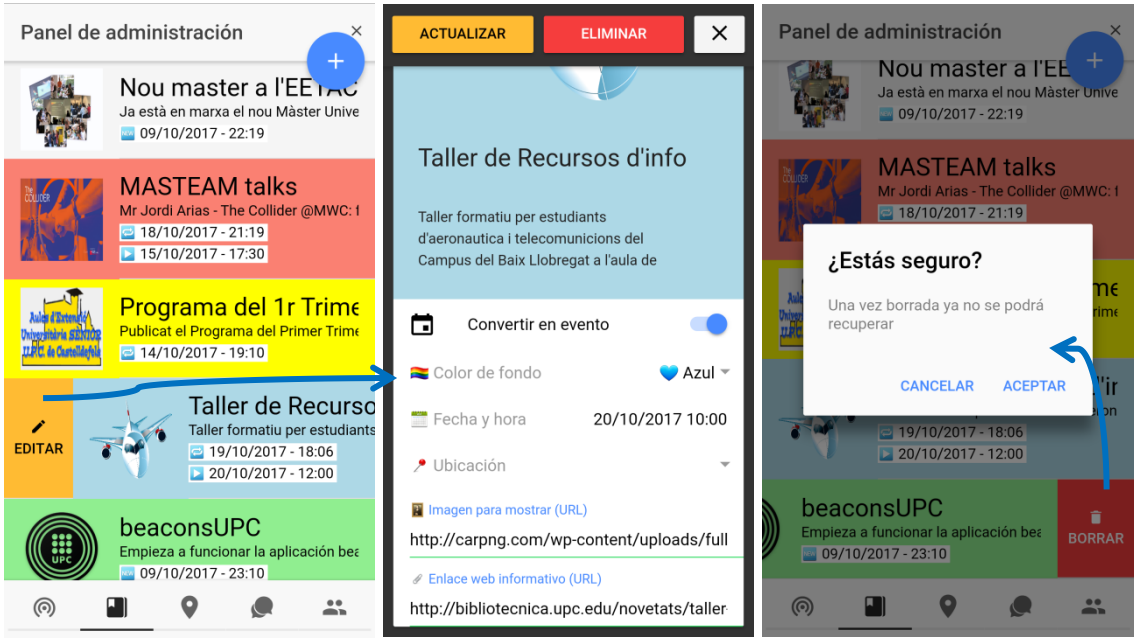
Cap. 4 Ventana de recordatorios (izquierda), ventana de noticias/eventos (centro) y ventana de vista previa de noticia/evento (derecha)



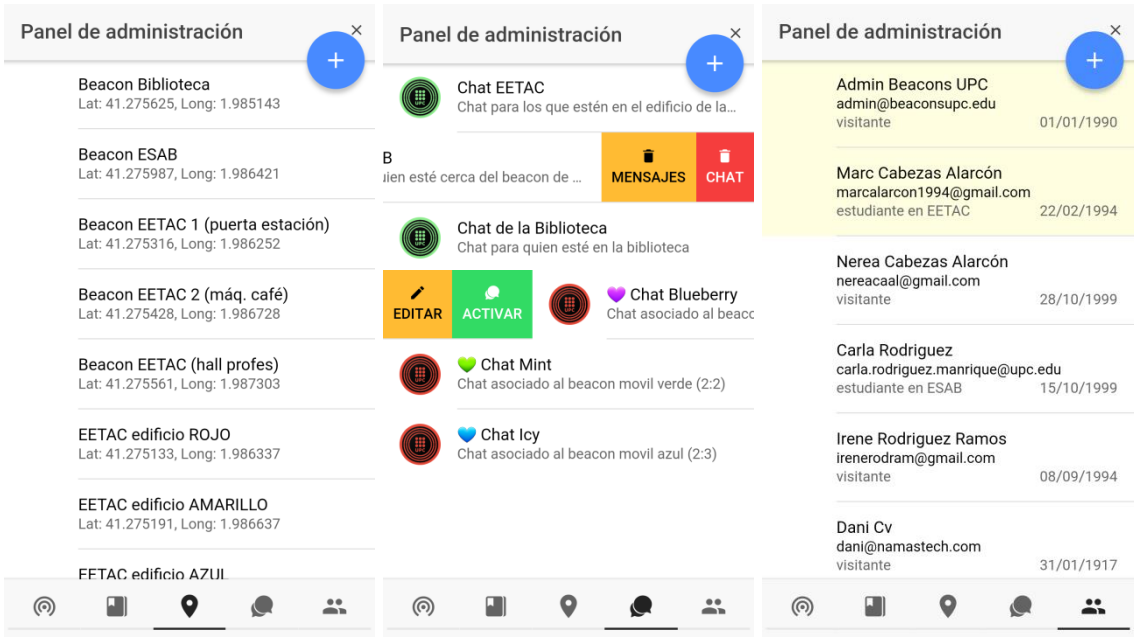
Cap. 5 Pestaña Mapa (izquierda), pestaña Beacons (centro) y ventana chat asociado a un *beacon* (derecha)



Cap. 6 Pestaña de configuración (izquierda), pestaña Beacons del panel de administración (centro) y ventana de edición de un *beacon* (derecha)



Cap. 7 Pestaña de noticias/eventos (izquierda), ventana de edición de una noticia/evento (centro), vista del *pop-up* que aparece al dar clic en borrar una noticia/evento (derecha)



Cap. 8 Pestaña de ubicaciones (izquierda), pestaña de chats y los botones que aparecen al deslizar (centro) y pestaña de usuarios (derecha)

A1.4. Funciones programadas en la aplicación

A continuación, se muestran algunas capturas de funciones implementadas en el código de la aplicación. Todo el código se puede encontrar en GitHub [1]:

A1.4.1. Funciones de creación, actualización o eliminación de clases de la BD

En la siguiente captura, se puede apreciar un grupo de funciones típico que nos permite interactuar con *Firebase* e insertar, actualizar o eliminar tablas o datos.

firebase-db.ts

```
createClass(clase){
  if(!clase.id){
    clase.id = Date.now();
  }
  return this.afDB.database.ref('users/'+this.auth.getUser()+'/classes/'+clase.day+'/'+clase.id).set(clase)
}

updateClass(clase){
  return this.afDB.database.ref('users/'+this.auth.getUser()+'/classes/'+clase.day+'/'+clase.id).update(clase)
}

getUserClasses(){
  return this.afDB.list('users/'+this.auth.getUser()+'/classes')
}

getUserClassesDay(day){
  return this.afDB.list('users/'+this.auth.getUser()+'/classes/'+day)
}

getDayClasses(day){
  return this.afDB.list('users/'+this.auth.getUser()+'/classes/'+day)
}

public deleteClass(day, id){
      this.afDB.database.ref('users/'+this.auth.getUser()+'/classes/'+day+'/'+id).remove();
}
```

A1.4.2. Funciones de registro o login de usuario con Firebase

auth.ts

```
// Registro de usuario
registerUser(email:string, password:string){
return this.afAuth.auth.createUserWithEmailAndPassword( email, password)
.then((res)=>{
 // El usuario se ha creado correctamente.
})
.catch(err=>Promise.reject(err))
}

// Login de usuario
loginUser(email:string, password:string){
  console.log("🔧 loginUser")
  return this.afAuth.auth.signInWithEmailAndPassword(email, password)
    .then(user => Promise.resolve(user))
    .catch(err => Promise.reject(err))
}
```

A1.4.3. Función de notificación de próxima clase

Esta función se ejecuta al detectar que se ha entrado en la región de *beacons* y comprueba si el usuario tiene alguna clase guardada cercana a esa hora para ejecutar la notificación.

beacon.ts

```
classesDisplayNotifications(){
  // si estoy dentro de la región y no he notificado aún o la última notificación fué hace más de 30 minutos entro
  if(this.insideRegion && (this.lastClassNotification == null || this.lastClassNotification+1800000 < Date.now())){// 1800000 = 30min
    // miro que hora es y la paso a minutos
    var d = new Date();
    this.minutesNow = (d.getHours())*60+(d.getMinutes());

    this.dbFirebase.getUserClasses().subscribe(classes=>{
      this.classes = classes;

      for(let i=0; i<classes.length; i++){
        if(classes[i].$key == this.today){
          this.dbFirebase.getUserClassesDay(this.today).subscribe(dia=>{
            // recorro el objeto que me llega transformando la hora en minutos para luego poder compararla con minutesNow (hora actual
            for (let i=0; i < dia.length; i++){
              let hora = dia[i].startTime.split(':');// startTime format -> HH:mm
              let minutos = (+hora[0])*60+(+hora[1]);
              dia[i].minutos = minutos;
            }
            //recorro el día que ha coincidido para ver si encuentro alguna clase entre
            //los 20 minutos antes y los 10 despues de la hora actual
            for(let j=0; j < dia.length; j++){
              if(dia[j].minutos-20 <= this.minutesNow && dia[j].minutos+10 >= this.minutesNow){
                // monto la notificación
                let edificio=null
                if(dia[j].building == 'rojo'){edificio='❤️'}
                else if(dia[j].building == 'amarillo'){edificio='💛'}
                else if(dia[j].building == 'azul'){edificio='💙'}
                else{edificio=''}
                let observaciones=null
                if(dia[j].obs != ''){observaciones=dia[j].obs}
                else{observaciones='Que vaya bien la clase!'}

                let title = '⚫ '+dia[j].subject+' en el aula '+dia[j].classroom+edificio+' a las '+dia[j].startTime+'h'
                let description = '⚪ '+observaciones

                // envío la notificación
                this.setLocalNotification(dia[j].id, title, description);

                // guardo la hora de esta notificación
                this.lastClassNotification = Date.now();
              }else{}
            }
          })
        }
        else{
          //console.log("NO COINCIDE NINGÚN DÍA")
        }
      }
    })

  }
  else{
    //console.log("No ejecuto la función de notificación de clase")
  }
}
```

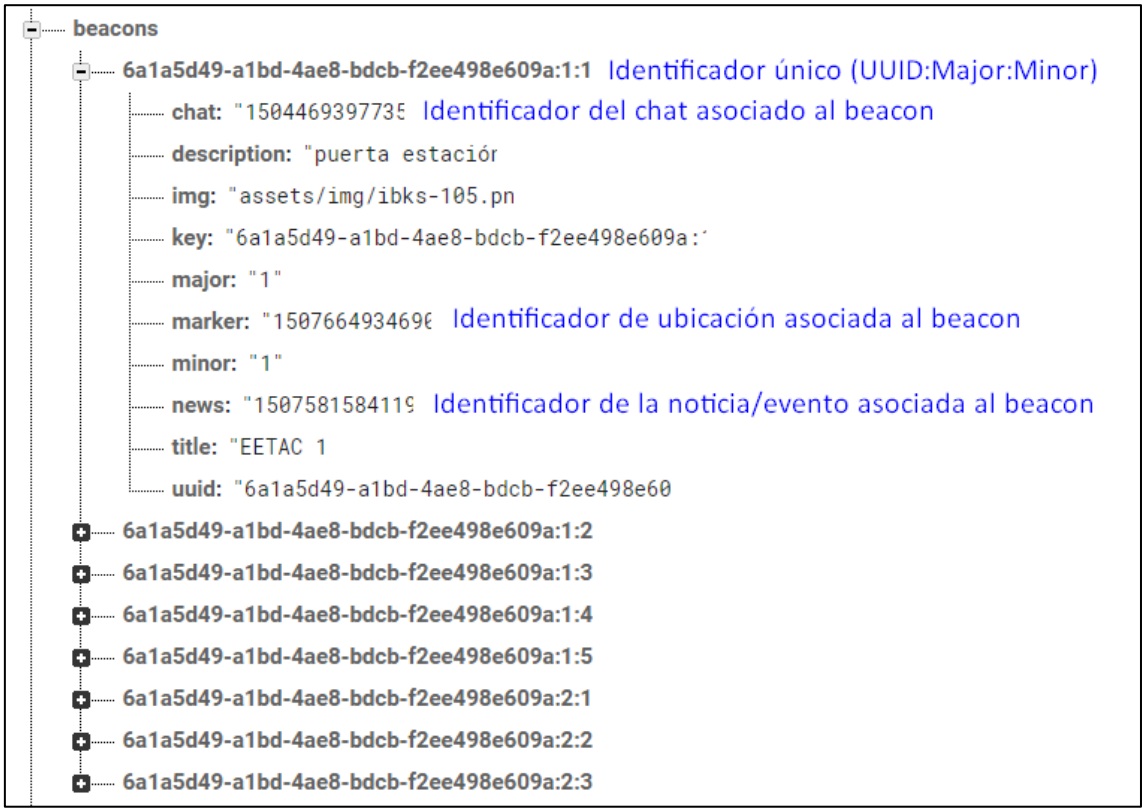
A1.5. Estructura de las bases de datos

Como ya se ha comentado en la memoria, trabajamos con el servicio de bases de datos no-SQL de *Firebase*. Hemos definido por el momento 6 tablas diferentes:

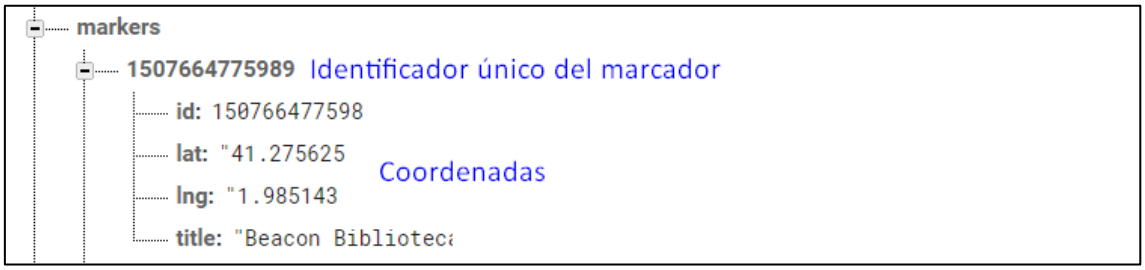
Todos los identificadores únicos que se utilizan, excepto los que definen los *beacons*, se generan obteniendo el número de milisegundos que han pasado desde el 1/1/1970, haciendo imposible la repetición de alguno de ellos.

A1.5.1. Tabla de beacons

En esta tabla, se definen los parámetros básicos de nuestros *beacons* y sus elementos asociados (chat, noticia o evento y marcador de ubicación).

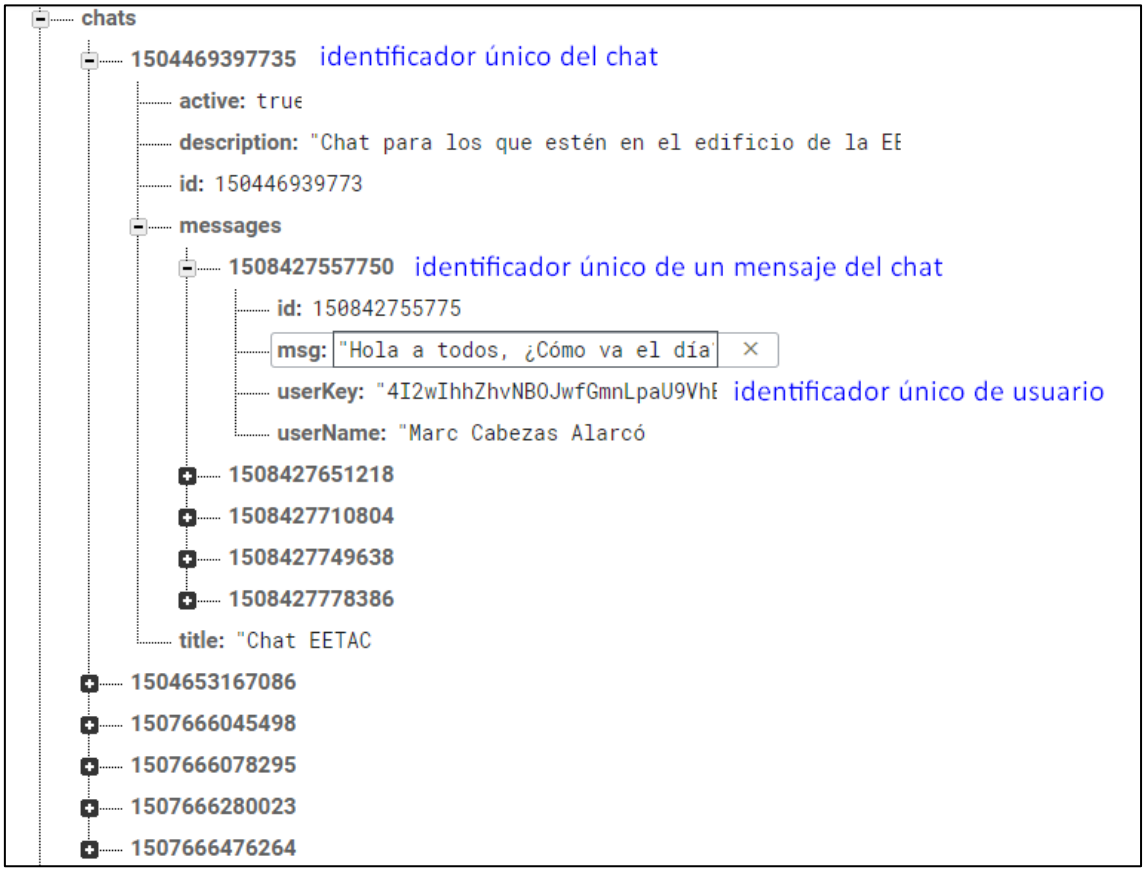


A1.5.2. Tabla de marcadores



A1.5.3. Tabla de chats

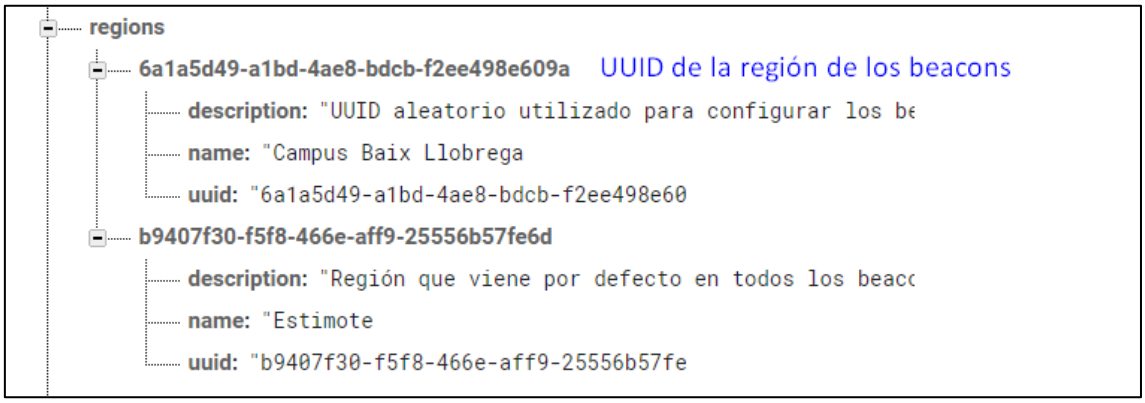
Podemos ver que dentro del chat se encuentra la tabla mensajes.



A1.5.4. Tabla de noticias o eventos

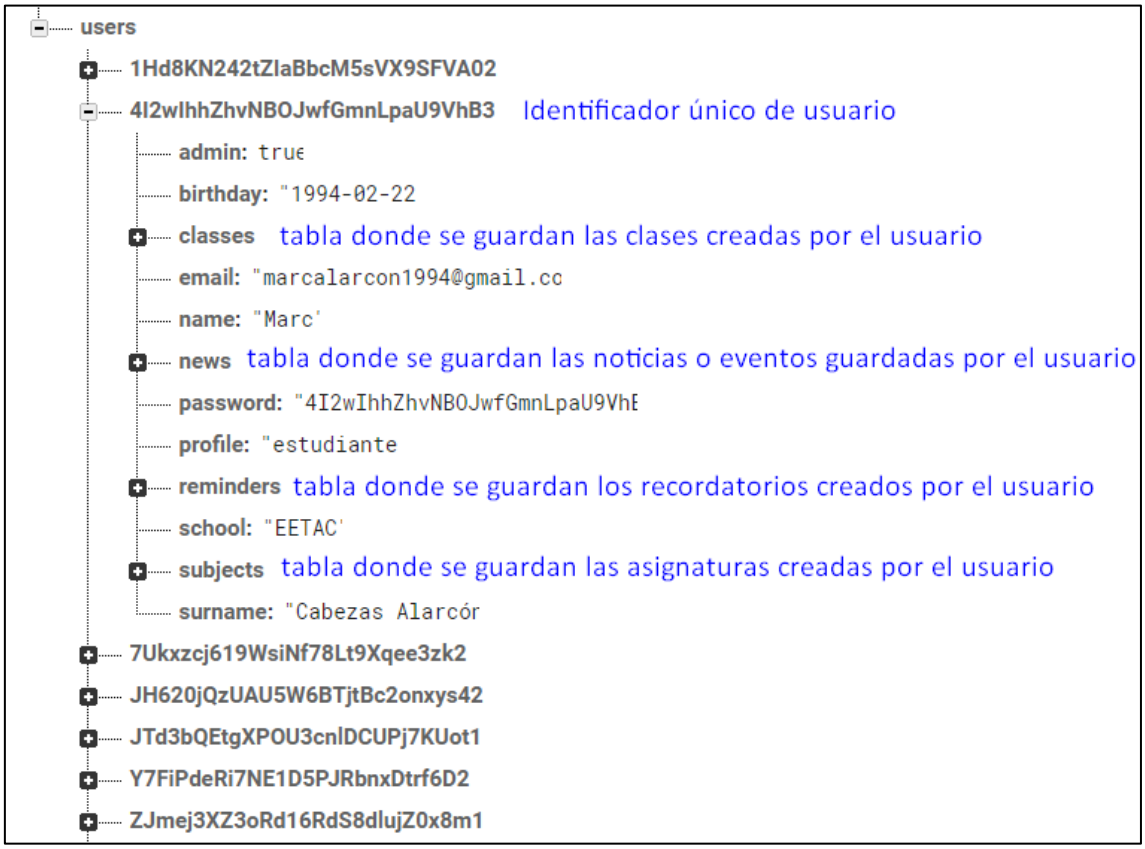
La hora del evento se guarda en formato ISO 8601.

A1.5.5. Tabla de regiones

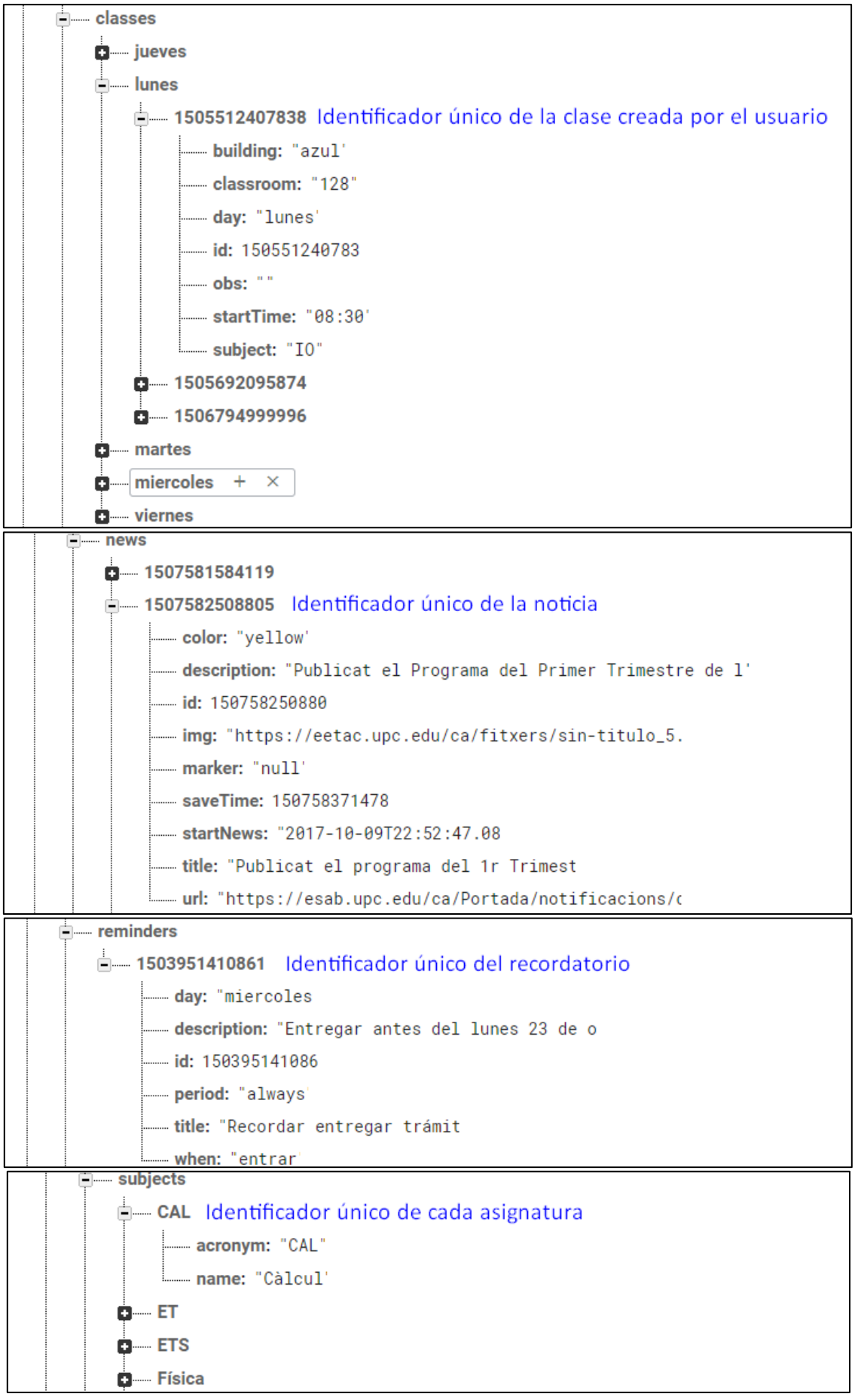


A1.5.6. Tabla de usuarios

La tabla de usuarios es la más compleja y más extensa debido a la multitud de información que se debe almacenar.



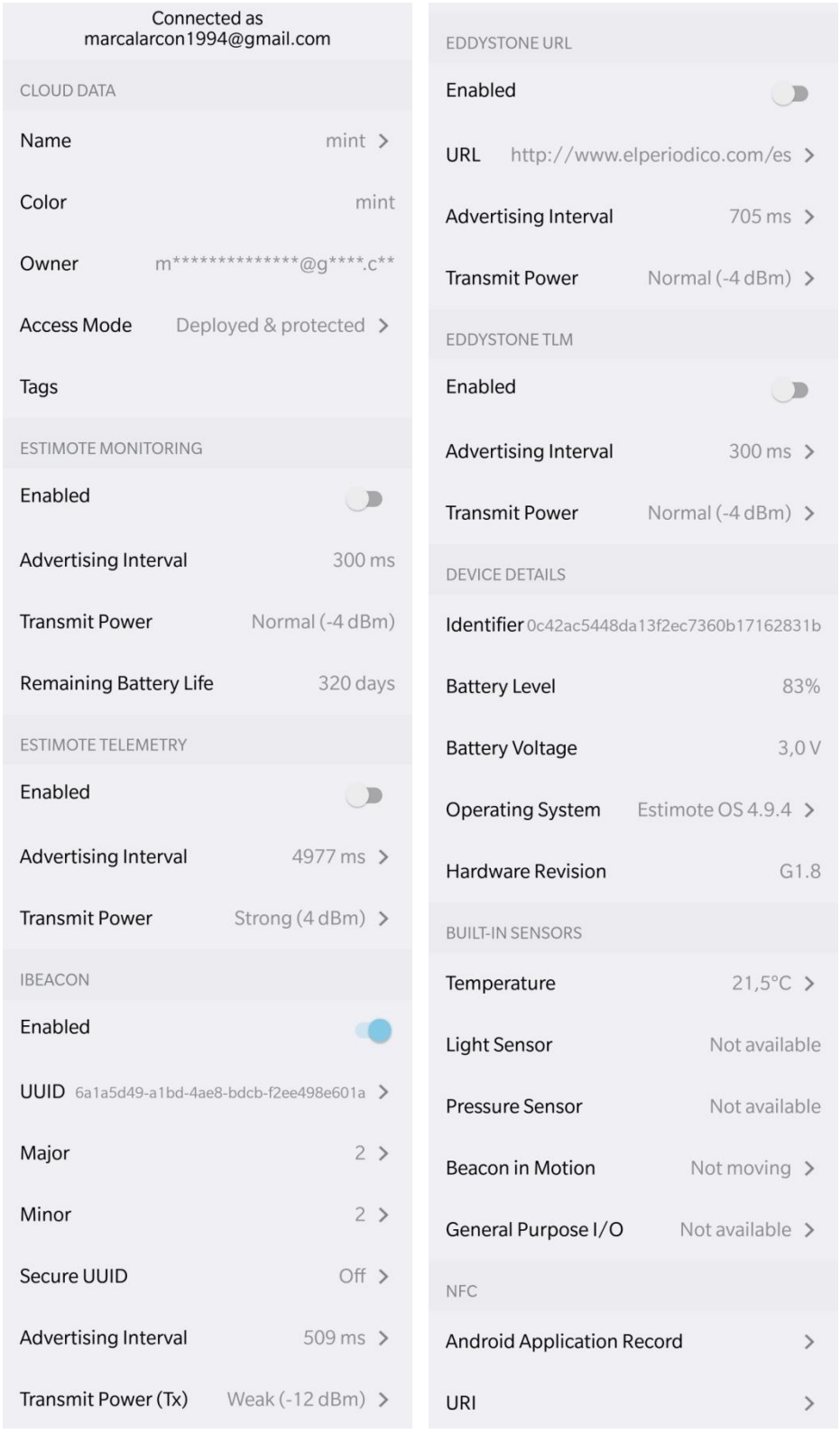
A continuación, vemos con más detalle las tablas que hay dentro de un usuario:


classes
jueves
lunes
1505512407838 Identificador único de la clase creada por el usuario
building: "azul"
classroom: "128"
day: "lunes"
id: 150551240783
obs: ""
startTime: "08:30"
subject: "IO"
1505692095874
1506794999996
martes
miercoles
viernes
news
1507581584119
1507582508805 Identificador único de la noticia
color: "yellow"
description: "Publicat el Programa del Primer Trimestre de l'
id: 150758250880
img: "https://eetac.upc.edu/ca/fitxers/sin-titulo_5.
marker: "null"
saveTime: 150758371478
startNews: "2017-10-09T22:52:47.08
title: "Publicat el programa del 1r Trimest
url: "https://esab.upc.edu/ca/Portada/notificacions/c
reminders
1503951410861 Identificador único del recordatorio
day: "miercoles
description: "Entregar antes del lunes 23 de o
id: 150395141086
period: "always"
title: "Recordar entregar trámit
when: "entrar"
subjects
CAL Identificador único de cada asignatura
acronym: "CAL"
name: "Càlcul"
ET
ETS
Física

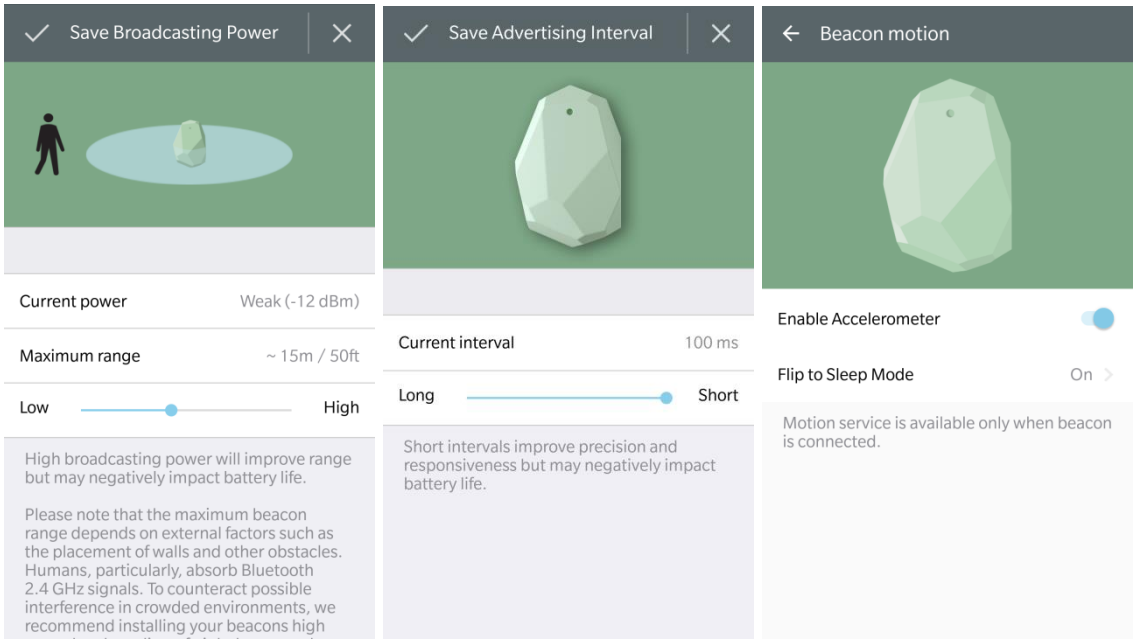
Anexo 2. Configuración de los beacons

A2.1. Beacons Estimote

Panel de configuración completo de la aplicación Estimote con un *beacon* seleccionado:

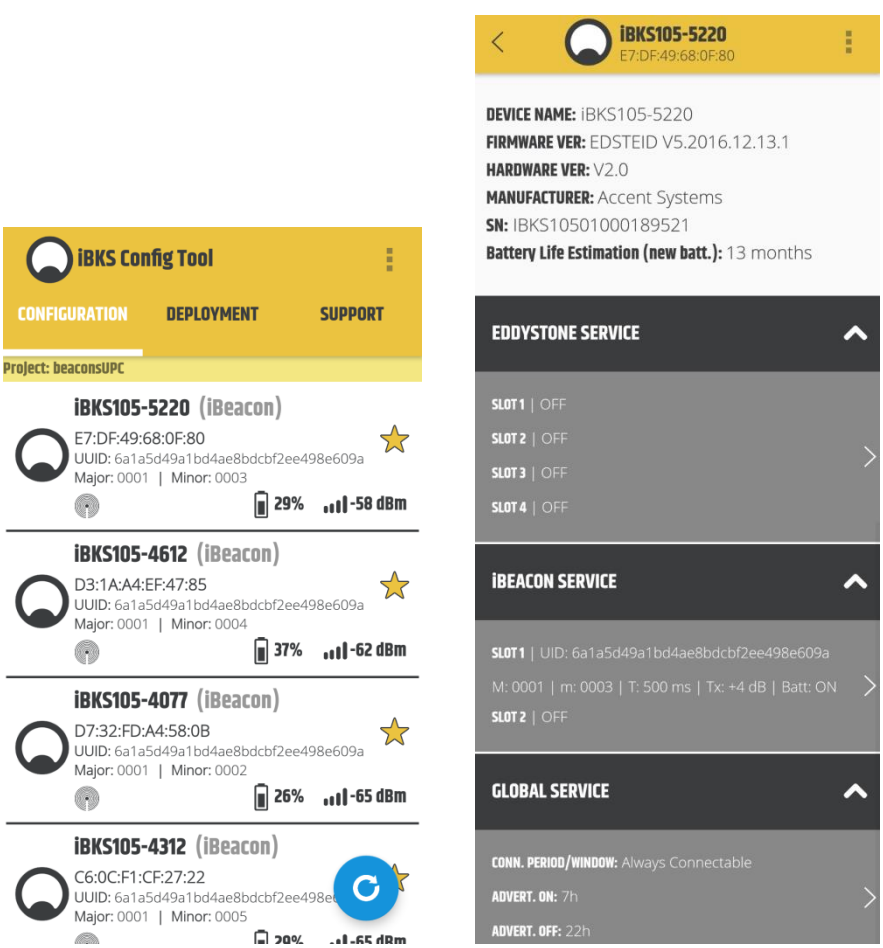


Algunas de las pantallas de configuración de la aplicación:

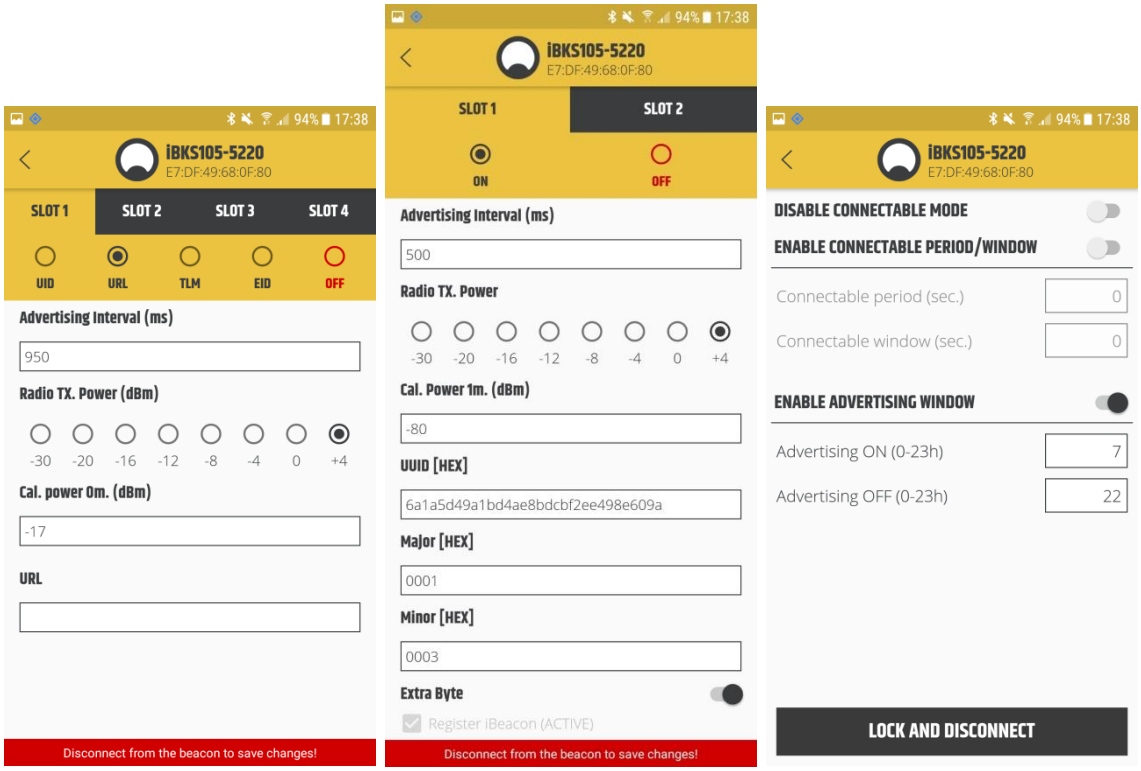


A2.2. Beacons Accent Systems

Panel de búsqueda y configuración de la aplicación iBKS *Config Tool*:



Pantallas de configuración de un *beacon* iBKS:



En la primera captura, vemos la pantalla de configuración del servicio *Eddystone*. Vemos que cuenta con 4 *slots* e integra las 4 posibles tramas existentes.

En la segunda captura, tenemos el panel de configuración de parámetros del servicio iBeacon. Se ha habilitado el *byte* extra para que nos informe del estado de la batería del *beacon*.

En el servicio global, hemos implementado la ventana de *advertising* para ahorrar así batería durante las horas en las que la facultad no da servicio. Se ha programado el encendido de éstos a las 7:00h de la mañana y, el apagado, a las 22:00h de la noche.

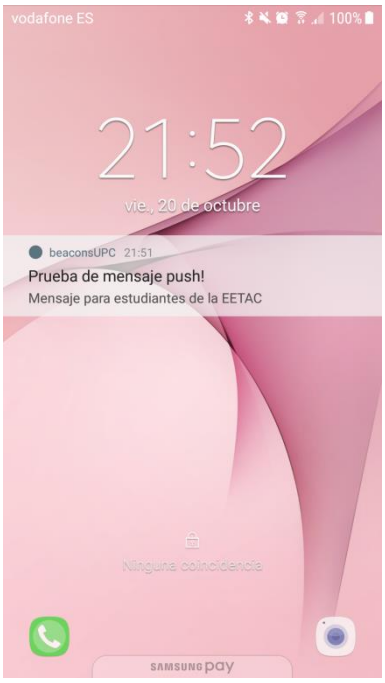
Anexo 3. Guías

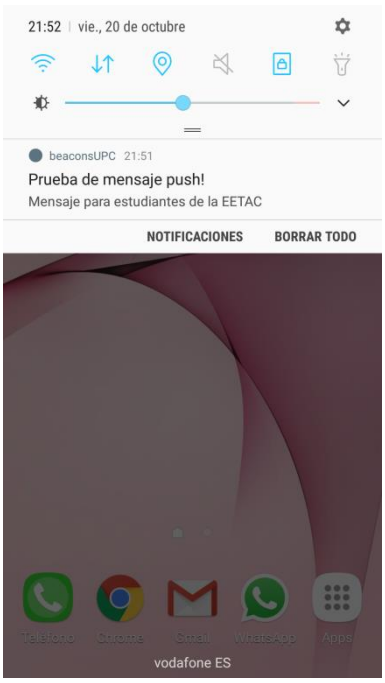
A3.1. Enviar notificaciones push desde Firebase

A continuación, se muestra la mayoría de campos que permite rellenar *Firebase* para las notificaciones *push*:

URL: https://console.firebase.google.com/project/beaconsupc/notification

Hemos hecho la prueba enviando la notificación únicamente a los alumnos de la EETAC y ésta ha sido la notificación que he recibido:

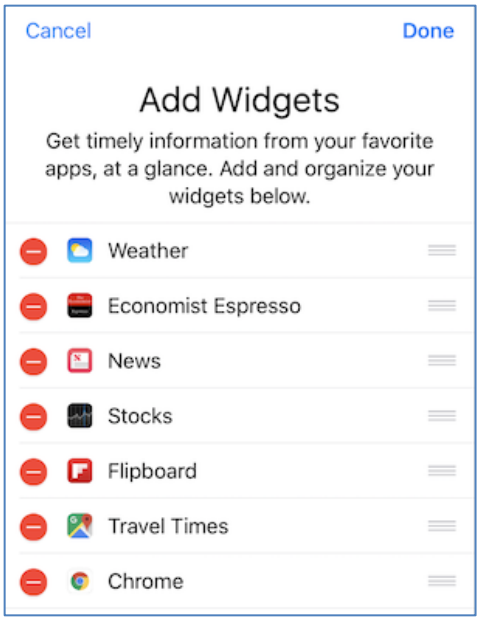


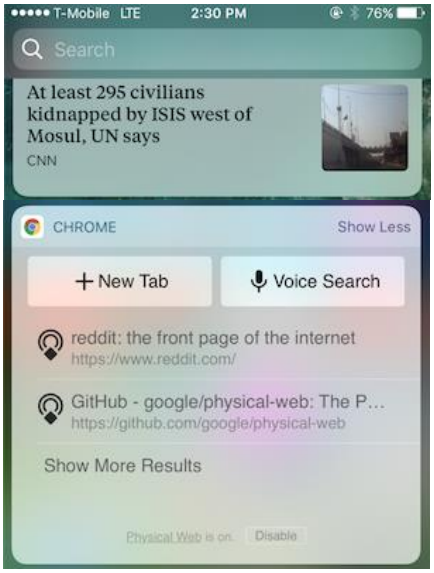


A3.2. Configurar un iPhone para recibir notificaciones Eddystone

Un *beacon* puede estar configurado para emitir una URL gracias al protocolo Eddystone-URL. Estas URLs se captan de manera automática en un dispositivo Android si éste tiene activada la ubicación y el *bluetooth.*

El sistema operativo iOS no puede captar por sí solo estas notificaciones. Para poder hacerlo, hay que añadir el Widget de Google Chrome al panel de notificaciones del iPhone y activar *bluetooth*.



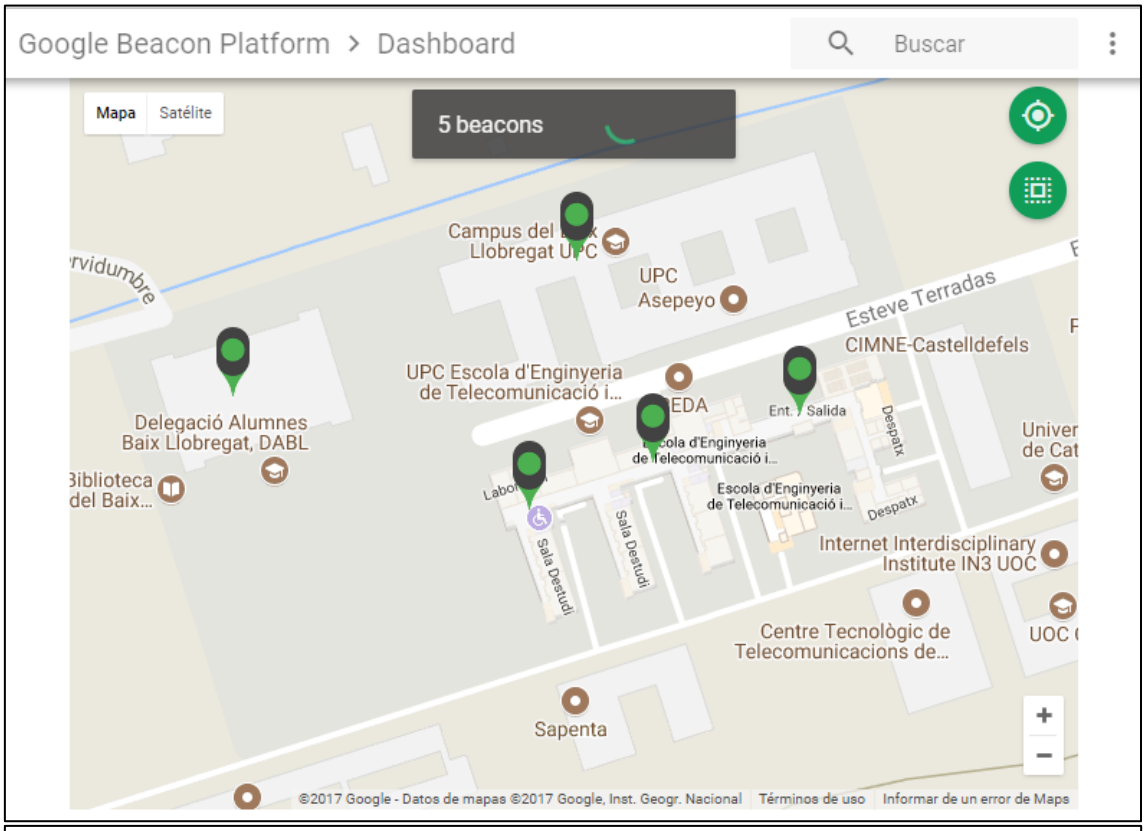


A3.3. Programar notificaciones Nearby desde Beacon Dashboard

URL: https://developers.google.com/beacons/dashboard/

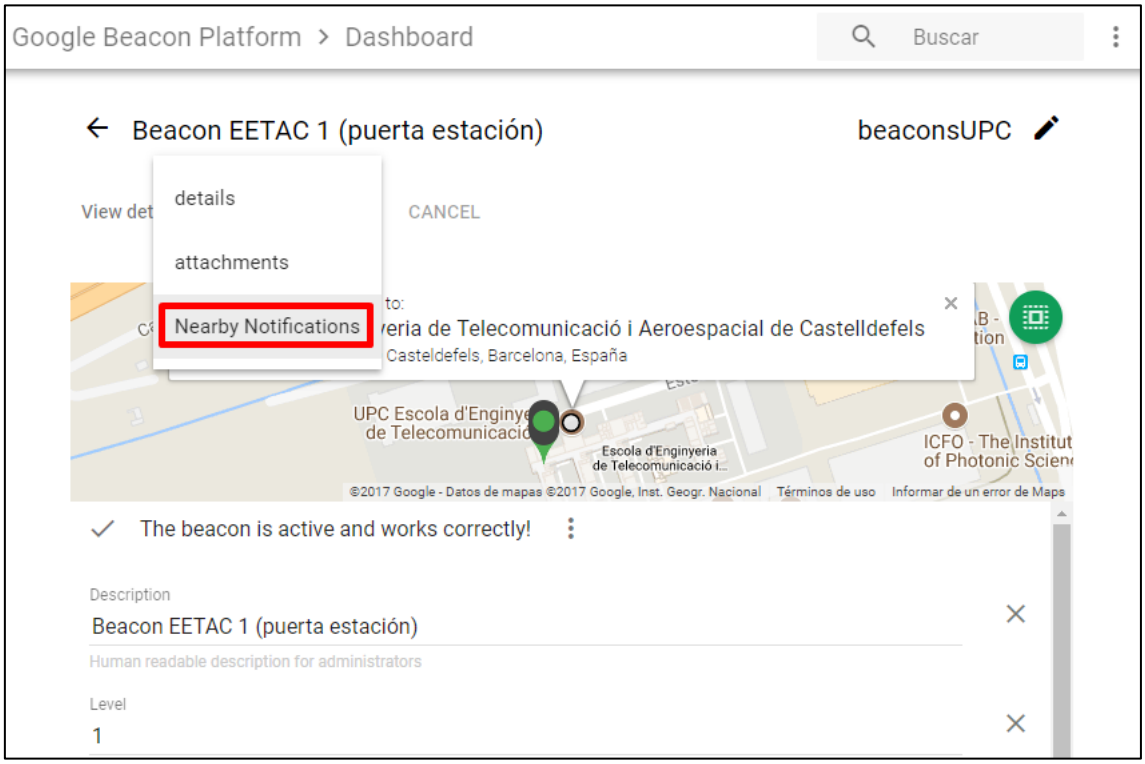
Tenemos configurados con el protocol Eddystone los 5 *beacons* iBKS105 de Accent Systems. Los tenemos registrados en el tablero de configuración de *beacons* que tiene Google y desde allí podemos configurar los mensajes *Nearby* que deben mostrar nuestros *beacons* cuando un usuario pasa cerca.

Cabe recordar que estas notificaciones no dependen de que el usuario tenga instalada la aplicación beaconsUPC, únicamente dependen de que sea un dispositivo móvil Android con versión 4.4 o superior y tenga el *bluetooth* y la ubicación activadas.

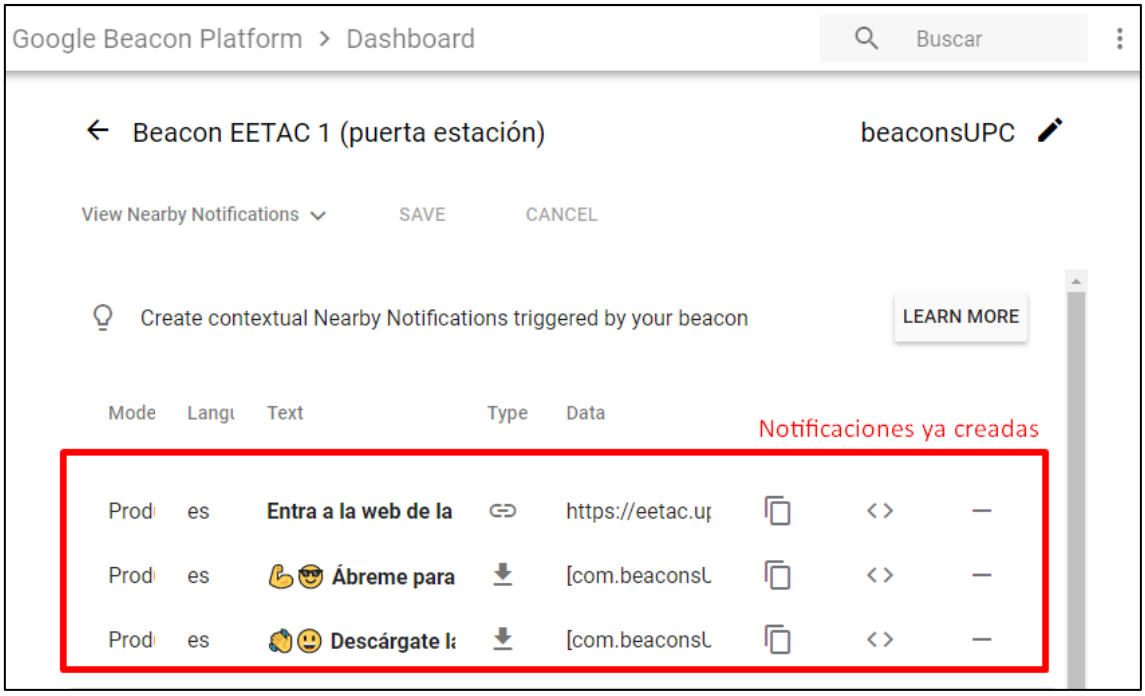


Description	Level	Status
Beacon EETAC 1 (puerta estación)	1	Active
Beacon EETAC 2 (máquina de café)	1	Active
Beacon EETAC 3 (hall profes)	1	Active
Beacon Biblioteca CBL	1	Active
Beacon ESAB	1	Active

Al hacer clic en uno de los *beacons* que tenemos registrados, entramos en su panel de configuración. Ahí se rellenan los parámetros básicos de ese *beacon* como su localización exacta, en qué planta se ha colocado, etc.

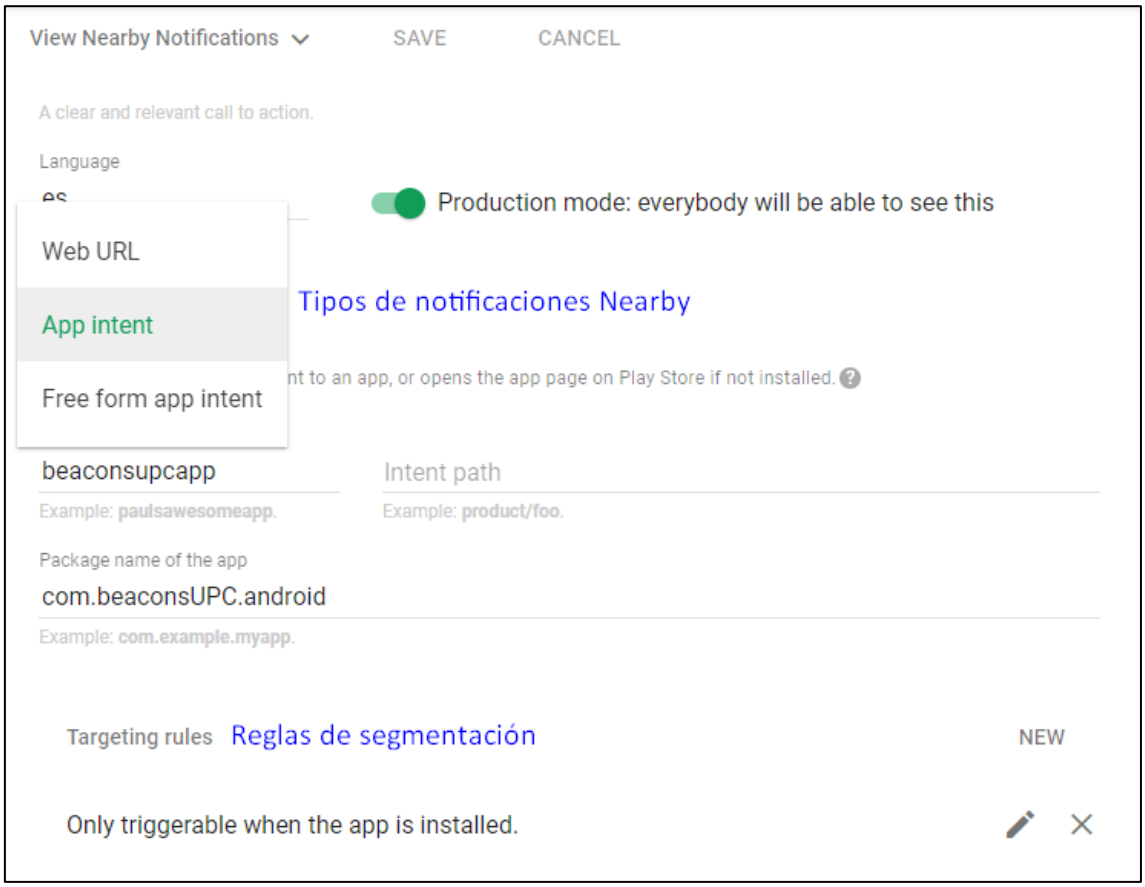


Si entramos en el panel de notificaciones *Nearby*, podremos ver las ya creadas o crear nuevas:

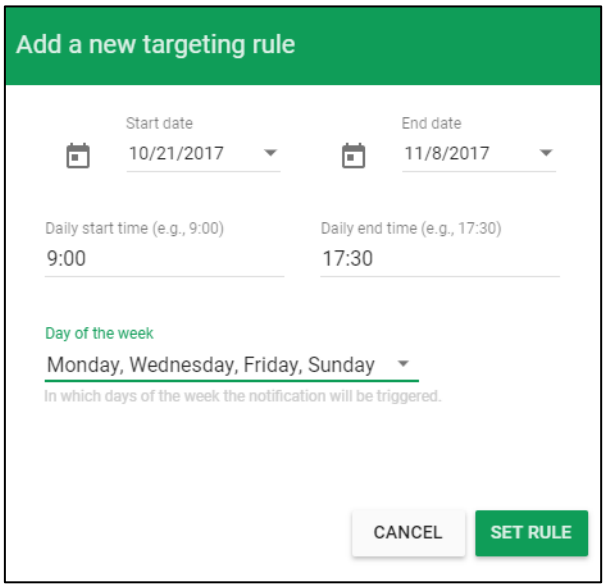


Nuestros *beacons* emiten un enlace URL y una notificación de instalación de nuestra aplicación (si no la tienen instalada) o de abertura.

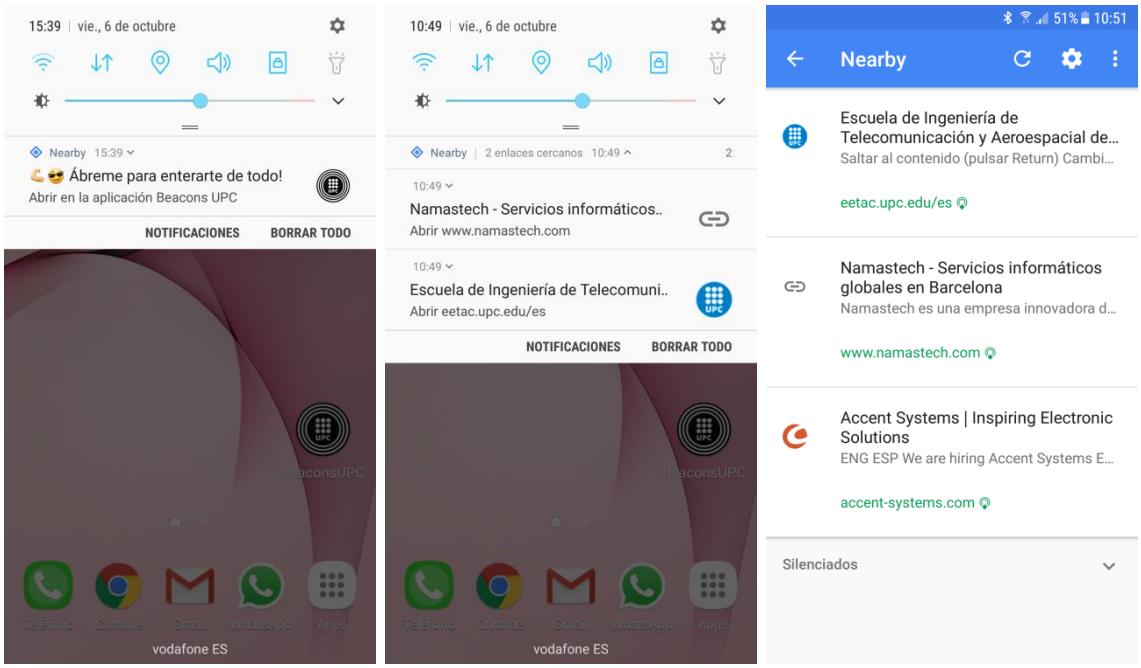
Se pueden crear diferentes tipos de notificaciones. Las más comunes son la de *Web URL* que consiste en emitir un enlace web igual que se hace con el protocolo Eddystone-URL o emitir el enlace a una aplicación de Play Store, *App intent*.



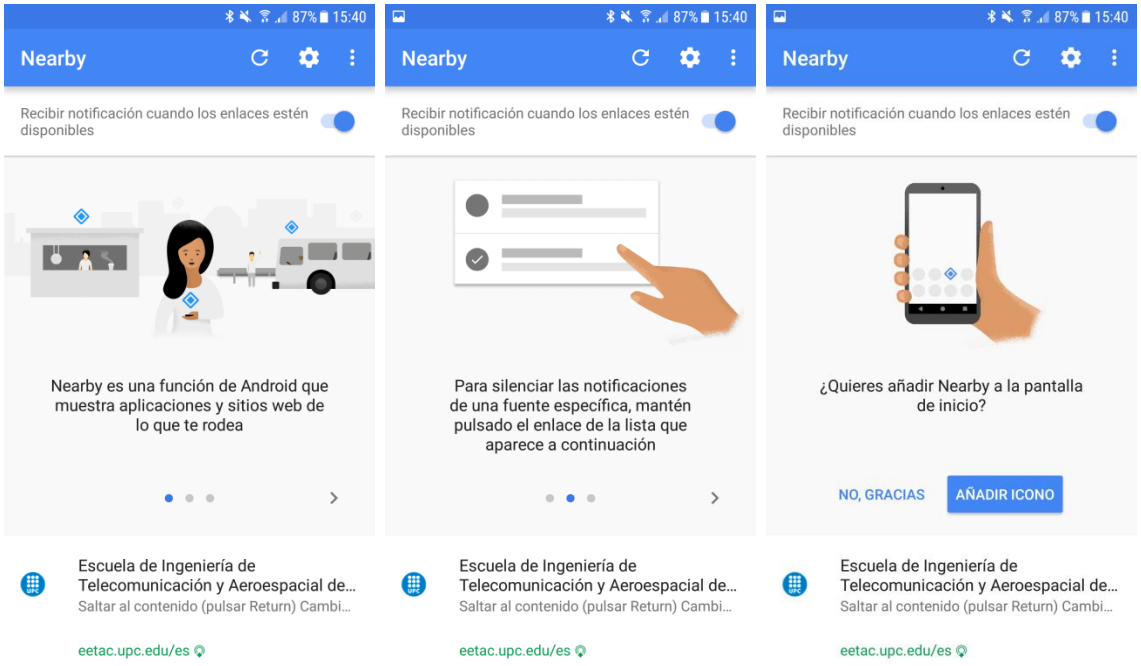
También existe la posibilidad de segmentar el grupo de usuarios a los que van dirigidos este tipo de notificaciones o programarlas únicamente un periodo de tiempo concreto.



Aquí tenemos algunos ejemplos de las notificaciones *Nearby* que podemos recibir sin tener la aplicación abierta.



El usuario que recibe por primera vez en su dispositivo móvil una notificación *Nearby* debe pasar este pequeño tutorial:



Nearby viene activado por defecto en todos los dispositivos Android con la versión 4.4 o superior.

A3.4. Pasos para generar el archivo apk para Play Store

Estos son los pasos que he seguido para crear el archivo *.apk* de mi aplicación antes de publicarlo en *Play Store*.

1) Aumentar la versión de la app (se hace en el archivo config.xml).

2) Eliminar el *plugin* de consola (ejecutarlo siempre aunque dé error).

```
$ ionic cordova plugin rm cordova-plugin-console
```

3) Montar la aplicación (se genera un archivo sin firmar ni optimizar).

```
$ ionic cordova build android --release --prod
```

Nos habrá generado un archivo en platforms/android/build/outputs/apk

4) Generar la clave (Esta clave sólo es necesaria generarla una vez y hay que guardarla bien para luego poder usarla en posteriores actualizaciones).

```
$ keytool -genkey -v -keystore my-release-key.keystore -alias
beaconsupc -keyalg RSA -keysize 2048 -validity 10000
```

Se habrá generado la clave en platforms/android/build/outputs/apk
(alias_name: beaconsupc).

5) Nos dirigimos a ese directorio y ejecutamos el siguiente comando de firma (vigilamos que el nombre del apk que ponemos en el comando sea el que se ha generado anteriormente).

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore
my-release-key.keystore android-release-unsigned.apk beaconsupc
```

Al ejecutar este comando, te pedirá la contraseña que pusiste para la clave.

6) Una vez firmado nuestro apk con nuestra llave, optimizamos el archivo apk con zipAlign (Se recomienda copiar el archivo zipalign.exe de su carpeta en el SDK de Android y pegarlo en platforms/android/build/outputs/apk para poder ejecutar el comando de zipalign directamente en ese directorio).

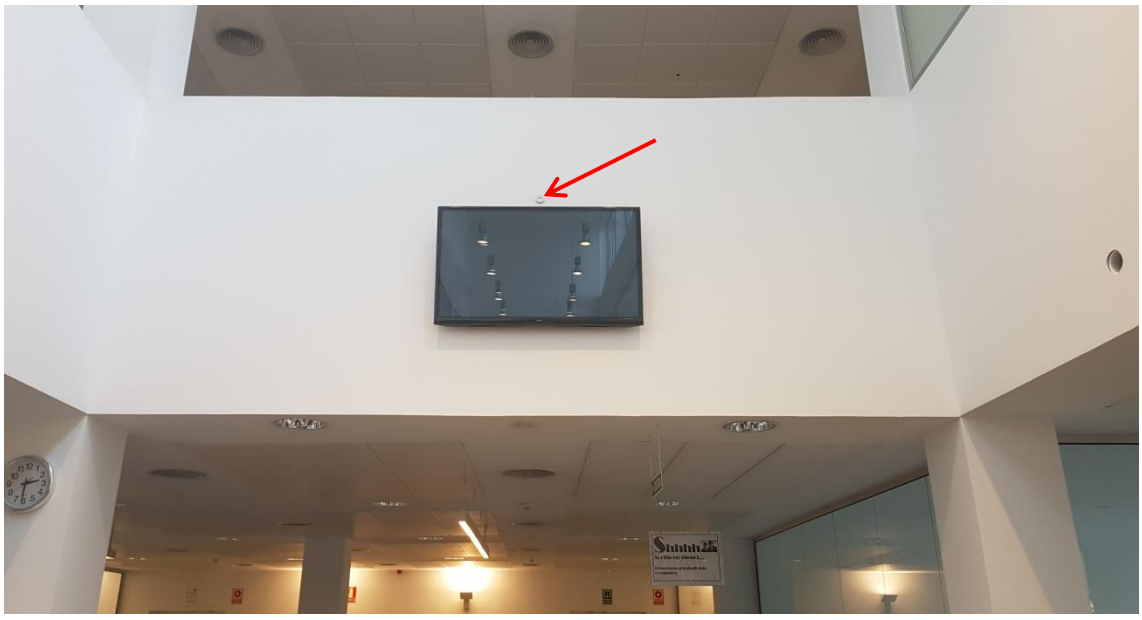
```
C:\Users\Marc\AppData\Local\Android\sdk\build-tools\20.0.0 → zipalign.exe
$ zipalign -v 4 android-release-unsigned.apk beaconsUPC.apk
```

En verde, el nombre de nuestro apk sin comprimir, en rojo, el nombre final de nuestro apk comprimido. Ese será el archivo que debemos subir al *Play Store*.

Anexo 4. Colocación de los beacons

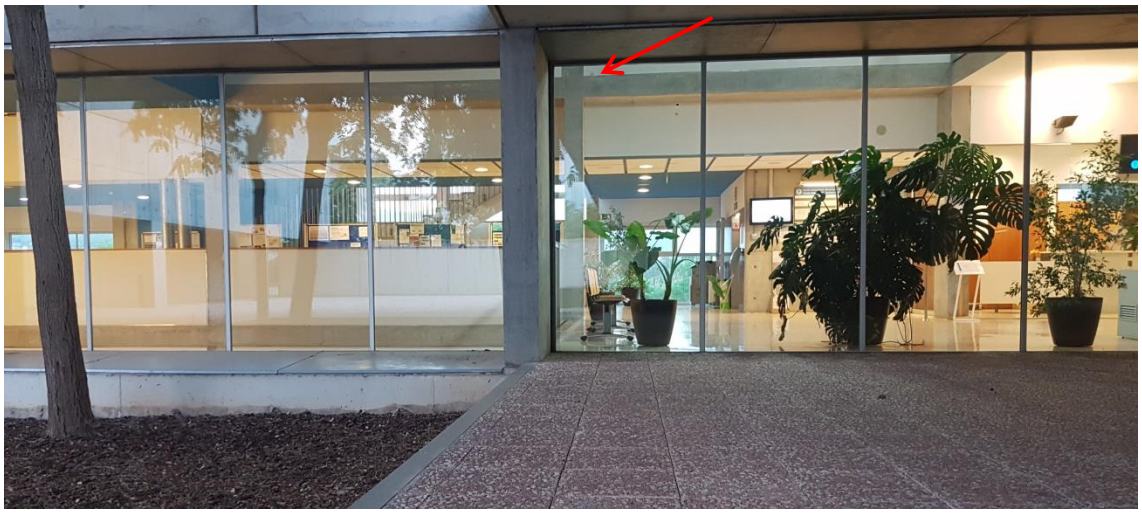
A4.1. Beacon de la biblioteca CBL

Este *beacon* se ha colocado encima de la televisión informativa de la primera planta de la biblioteca, en un espacio abierto que comunica con las 2 plantas principales.



A4.2. Beacon de la ESAB

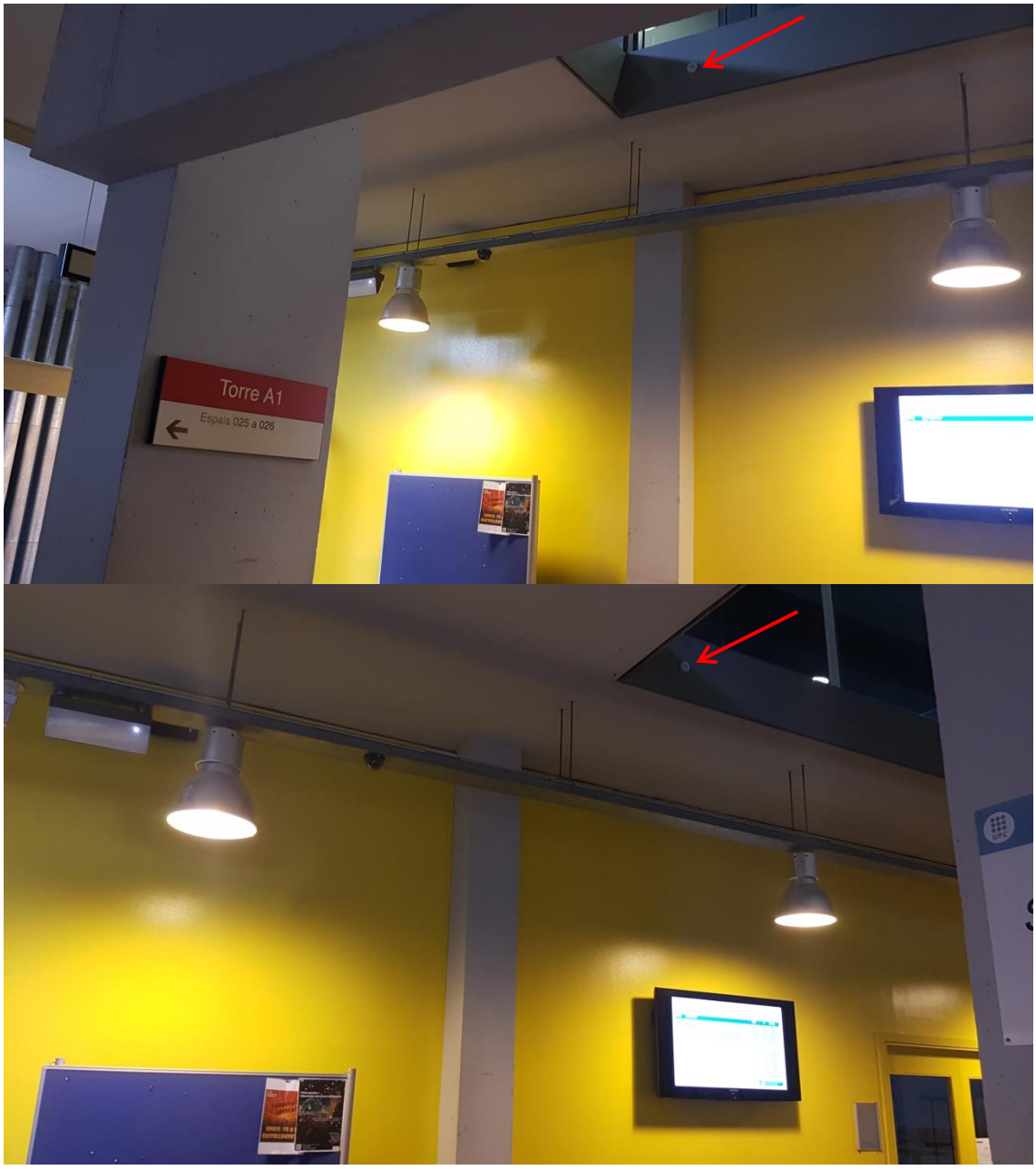
Este *beacon* se ha colocado en el *hall* principal puesto que sólo existe una entrada a ese edificio.



A4.3. Beacons de la EETAC

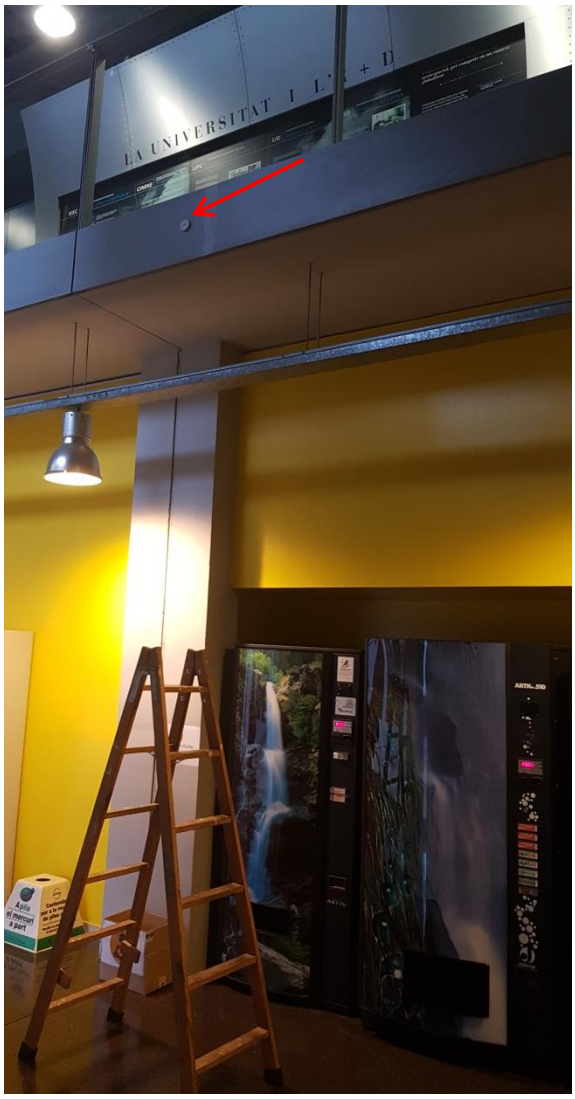
A4.3.1. Beacon en la entrada a la torre roja

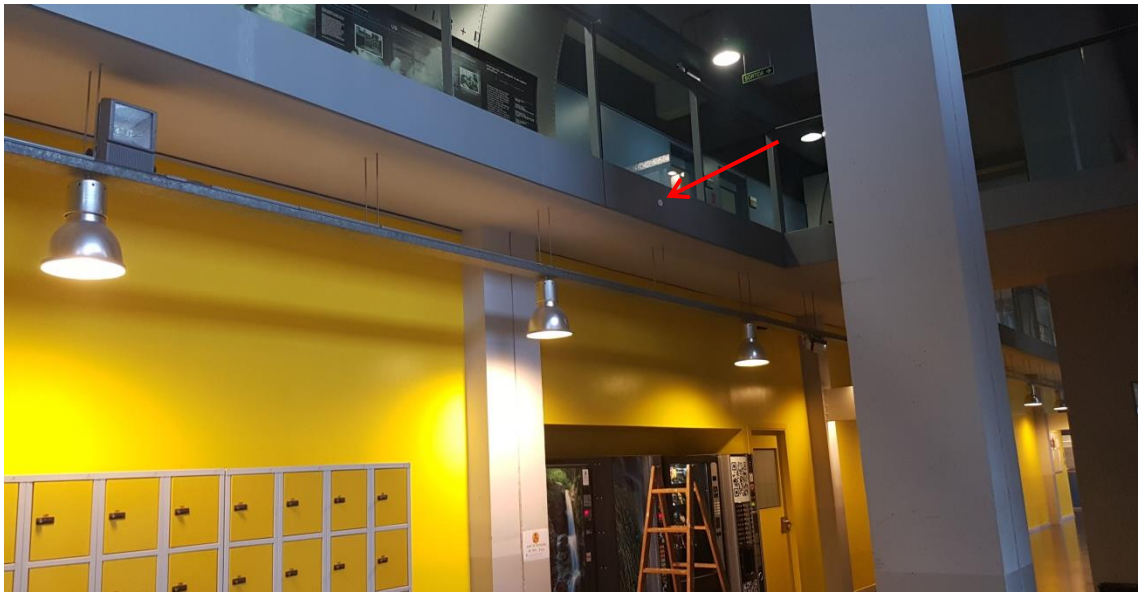
Este *beacon* está colocado en la entrada más próxima a la biblioteca y cubre un poco del patio central y los pasillos principales de la torre roja.



A4.3.2. Beacon entre la torre amarilla y azul

Este *beacon* está colocado entre la torre amarilla y la azul y da cobertura a todos los pasillos colindantes de la zona.





A4.3.3. Beacon en el hall de la torre de profesores

Este *beacon* está colocado en el *hall* de entrada a la torre de profesores de la EETAC. Este *hall* está acristalado por los laterales y permite cubrir gran parte de la zona exterior.