



FACULTAD DE INGENIERÍAS Y CIENCIAS APLICADAS

DESARROLLO E IMPLEMENTACIÓN DE UNA APLICACIÓN
MULTIPLATAFORMA USANDO BEACONS PARA PROMOVER TRES
CARRERAS DE LA UDLA

AUTOR

Jonathan Raúl Muñoz Narvárez

AÑO

2018



FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

DESARROLLO E IMPLEMENTACIÓN DE UNA APLICACIÓN
MULTIPLATAFORMA USANDO BEACONS PARA PROMOVER TRES
CARRERAS DE LA UDLA.

Trabajo de Titulación presentado en conformidad con los requisitos
establecidos para optar por el título de Ingeniero en Redes y
Telecomunicaciones

Profesor guía
MSc. Johanna Rafaela Ortega Briones

Autor
Jonathan Raúl Muñoz Narvárez

Año
2018

DECLARACIÓN DEL PROFESOR GUÍA

“Declaro haber dirigido este trabajo, Desarrollo e implementación de una aplicación multiplataforma usando beacons para promover tres carreras de la UDLA, a través de reuniones periódicas con el estudiante Jonathan Raúl Muñoz Narváez, en el semestre 2018-2, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación.”

Johanna Rafaela Ortega Briones

Magister en Ingeniería de Redes de Comunicaciones

C.C. 1714578984

DECLARACIÓN DEL PROFESOR CORRECTOR

“Declaro haber revisado este trabajo, Desarrollo e implementación de una aplicación multiplataforma usando beacons para promover tres carreras de la UDLA, del estudiante Jonathan Raúl Muñoz Narváez, en el semestre 2018-2, dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación”.

Pablo Geovanny Palacios Játiva

Magister en Ingeniería de Redes de Comunicaciones

C.C. 0927864454

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes.”

Jonathan Raúl Muñoz Narváz

C.C. 1722301015

AGRADECIMIENTOS

Agradezco a Dios, mis padres, mis hermanos y amigos; quienes aportaron con su granito de arena para ser luz, motivación y apoyo en cada etapa de mi educación. A mis maestros y al cuerpo docente de la Universidad de Las Américas por su constante esfuerzo en impulsar educación de calidad.

DEDICATORIA

Dedico este trabajo a mis padres: Mary y Raúl, por su incondicional apoyo en todo momento, por demostrarme que nuestros sueños y metas las cumplimos con esfuerzo, valores y carácter. A mis hermanos: Naty y Jairo, por ser ese aire necesario para despejar la mente en momentos abrumadores.

RESUMEN

El presente documento tiene como objetivo el estudio de la tecnología *Bluetooth* y su protocolo de bajo consumo de energía en dispositivos que emiten señales de radiofrecuencia conocidos como *beacons* o balizas, que permiten programar el envío de contenido en base a su proximidad con un *smartphone*. Para su desarrollo, se precisa de una aplicación móvil para los sistemas operativos *Android* y *iOS* en las plataformas *Android Studio* y *Xcode* respectivamente, que permita mostrar información sobre las carreras de Telecomunicaciones, *Software* y Tecnologías de la Información facilitando la postulación de nuevos alumnos, acompañada de la integración con *Google Firebase Cloud Messaging* para el envío de notificaciones sobre eventos destacados a realizarse en la Universidad.

Palabras clave: *Beacons*, *Bluetooth*, *Android Studio*, *Xcode*, aplicaciones móviles, *Firebase*.

ABSTRACT

The objective of this document is the study of the Bluetooth technology and its protocol of low energy in devices that emit radio frequency signals known as beacons, which can be programmed to send content based on its proximity to a smartphone. It requires an application developed for Android and iOS operating systems on Android Studio and Xcode platforms respectively, which allows to show information about the careers of Telecommunications, Software and Information Technologies facilitating the application of new students, accompanied by an interface with Google Firebase Cloud Messaging for notifications about featured events to be held at the University.

Keywords: Beacons, Bluetooth, Mobile app development, Firebase, Android Studio, Xcode.

ÍNDICE

| | | |
|--------|---|----|
| 1. | INTRODUCCIÓN | 1 |
| 1.1. | Antecedentes | 1 |
| 1.1.1. | Alcance | 2 |
| 1.1.2. | Justificación | 3 |
| 1.2. | Objetivo General..... | 3 |
| 1.3. | Objetivos Específicos | 4 |
| 2. | ANÁLISIS DEL ESTADO DEL ARTE | 4 |
| 2.1. | Internet de las Cosas (IoT)..... | 4 |
| 2.2. | Tecnologías en IoT | 5 |
| 2.2.1. | Wireless Local Area Network (WLAN) | 6 |
| 2.2.2. | Red Ad-hoc inalámbrica..... | 6 |
| 2.2.3. | ZigBee | 7 |
| 2.2.4. | Bluetooth..... | 7 |
| 2.3. | Bluetooth Low Energy (BLE) | 14 |
| 2.3.1. | Jerarquía de perfiles Bluetooth basados en GATT | 16 |
| 2.3.2. | Perfil de Acceso Genérico (GAP)..... | 17 |
| 2.3.3. | Advertising y escaneo | 18 |
| 2.3.4. | Conexión..... | 21 |
| 2.3.5. | Seguridad | 21 |
| 2.3.6. | Modelos de asociación en el emparejamiento | 22 |
| 2.4. | Definición de las redes WPAN | 23 |
| 2.4.1. | Tipos de WPAN | 24 |
| 2.5. | High Rate Wireless Personal Networks (HR-WPAN) | 25 |
| 2.5.1. | Características de acceso al medio | 26 |
| 2.6. | Low Rate Wireless Personal Networks (LR-WPAN)..... | 27 |
| 2.6.1. | Red sin Beacon..... | 28 |
| 2.6.2. | Red con Beacon | 28 |
| 2.7. | Topología de red WPAN para BLE | 29 |

| | | |
|--------|--|----|
| 3. | DISEÑO Y CONFIGURACIÓN DE LOS BEACONS... | 31 |
| 3.1. | Diseño de la red..... | 31 |
| 3.2. | Introducción a los beacons Bluetooth Low Energy | 32 |
| 3.3. | Estimote Beacons | 33 |
| 3.3.1. | Características técnicas..... | 33 |
| 3.3.2. | Interfaz Estimote Cloud..... | 34 |
| 3.4. | Protocolos de los beacons | 36 |
| 3.4.1. | Google Eddystone | 36 |
| 3.4.2. | Apple iBeacon..... | 38 |
| 3.4.3. | Apple iBeacon vs Google Eddystone..... | 40 |
| 3.5. | Configuración de beacons desde la aplicación móvil de Estimote | 40 |
| 4. | DESARROLLO DE LA APLICACIÓN..... | 44 |
| 4.1. | Diagrama de flujo de la aplicación | 44 |
| 4.2. | Android Studio..... | 45 |
| 4.2.1. | Estructura de los proyectos en Android Studio | 46 |
| 4.2.2. | Estimote Proximity SDK para Android | 47 |
| 4.3. | Aplicación en Android..... | 47 |
| 4.3.1. | Diseño de la interfaz gráfica en Android | 50 |
| 4.3.2. | Notificaciones locales en Android | 53 |
| 4.4. | Xcode | 55 |
| 4.4.1. | Estructura de los proyectos en Xcode | 55 |
| 4.4.2. | Estimote Proximity SDK para iOS..... | 57 |
| 4.5. | Aplicación en iOS | 57 |
| 4.5.1. | Diseño de la interfaz gráfica en iOS..... | 60 |
| 4.5.2. | Notificaciones locales en iOS..... | 62 |
| 5. | INTEGRACIÓN DE LA APLICACIÓN CON GOOGLE FIREBASE CLOUD MESSAGING | 64 |

| | | |
|--------|--|-----|
| 5.1. | Google Firebase Cloud Messaging..... | 64 |
| 5.1.1. | Tipos de mensajes..... | 65 |
| 5.1.2. | Arquitectura de FCM..... | 66 |
| 5.2. | Envío de mensajes con Firebase Console..... | 67 |
| 5.3. | Integración de FCM con la aplicación para Android | 70 |
| 5.4. | Integración de FCM con la aplicación para iOS..... | 73 |
| 6. | IMPLEMENTACIÓN DE LA SOLUCIÓN (BEACONS Y APLICACIÓN) | 79 |
| 6.1. | Implementación de beacons..... | 79 |
| 6.2. | Implementación de la aplicación..... | 80 |
| 6.2.1. | Preparación de la aplicación para distribución en Google Play Store | 80 |
| 6.2.2. | Publicación de la aplicación en Google Play Store | 83 |
| 6.2.3. | Preparación de la aplicación para distribución en App Store | 87 |
| 6.2.4. | Publicación de la aplicación en App Store | 89 |
| 7. | PROTOCOLO DE PRUEBAS..... | 91 |
| 7.1. | Pruebas de la aplicación en Android | 91 |
| 7.1.1. | Evaluación de parámetros de proximidad..... | 91 |
| 7.1.2. | Evaluación de notificaciones remotas y uso de la aplicación..... | 93 |
| 7.2. | Pruebas de la aplicación en iOS..... | 94 |
| 7.2.1. | Evaluación de notificaciones push y uso de la aplicación | 95 |
| 8. | CONCLUSIONES Y RECOMENDACIONES | 98 |
| 8.1. | Conclusiones..... | 98 |
| 8.2. | Recomendaciones | 99 |
| | REFERENCIAS | 102 |
| | ANEXOS | 105 |

ÍNDICE DE FIGURAS

| | |
|--|----|
| Figura 1. Ilustración de dispositivos interconectados. | 5 |
| Figura 2. Proceso de una conexión Bluetooth..... | 8 |
| Figura 3. Arquitectura Bluetooth de alto nivel..... | 12 |
| Figura 4. Arquitectura Bluetooth detallada. | 14 |
| Figura 5. Pila de protocolos BLE..... | 15 |
| Figura 6. Jerarquía del perfil GATT..... | 17 |
| Figura 7. Proceso de advertising y escaneo en BLE..... | 19 |
| Figura 8. Escaneo activo y pasivo en BLE. | 20 |
| Figura 9. Eventos durante los intervalos de conexión (Maestro-Esclavo)..... | 21 |
| Figura 10. Cobertura y tasas de transferencia de varias tecnologías inalámbricas. | 23 |
| Figura 11. Estructura de las super tramas..... | 26 |
| Figura 12. Topologías en LR-WPAN..... | 27 |
| Figura 13. Proceso de transferencia de datos sin beacon. | 28 |
| Figura 14. Proceso de transferencia de datos con beacon. | 29 |
| Figura 15. Topología de conexión..... | 30 |
| Figura 16. Topología Broadcast. | 31 |
| Figura 17. Diseño de la red WPAN en topología broadcast..... | 32 |
| Figura 18. Beacons Bluetooth de Estimote. | 33 |
| Figura 19. Visualización de beacons en la plataforma Estimote Cloud. | 35 |
| Figura 20. Visualización de información adicional del beacon Software en la plataforma Estimote Cloud. | 35 |
| Figura 21. Estructura de trama Eddystone-UID..... | 37 |
| Figura 22. Estructura de trama Eddystone-EID..... | 38 |
| Figura 23. Estructura de la trama Eddystone-URL..... | 38 |
| Figura 24. Visualización de beacons y sus configuraciones. | 41 |
| Figura 25. Visualización de las opciones disponibles para iBeacon en Estimote App. | 42 |
| Figura 26. Visualización de opciones disponibles para Eddystone UID, URL y TLM. | 43 |
| Figura 27. Diagrama de flujo de la aplicación móvil. | 45 |

| | |
|--|----|
| Figura 28. Archivos del proyecto en la vista de proyectos de Android. | 46 |
| Figura 29. Visualización de las credenciales obtenidas del portal Estimote Cloud. | 48 |
| Figura 30. Asignando identificadores adjuntos a un beacon desde el portal Estimote Cloud. | 49 |
| Figura 31. Visualización de URL acortados mediante la página Google URL Shortener..... | 50 |
| Figura 32. Actividades prediseñadas disponibles al crear una nueva aplicación en Android Studio. | 51 |
| Figura 33. Plano de pantalla de la actividad “Noticias”..... | 52 |
| Figura 34. Plano de pantalla y navegación de la actividad “Beacon Scanner” | 53 |
| Figura 35. Captura de pantalla con las notificaciones de proximidad recibidas en la bandeja de notificaciones del sistema de Android al entrar y salir del rango de alcance de un beacon. | 54 |
| Figura 36. Diagrama de distribución de métodos, clases y frameworks para el desarrollo de la aplicación en Android. | 55 |
| Figura 37. Archivos que conforman la estructura de un proyecto en Xcode. ... | 56 |
| Figura 38. Visualización de frameworks de Estimote Proximity SDK en la interfaz de Xcode..... | 57 |
| Figura 39. Habilitación de opción Bluetooth para modo de ejecución en segundo plano. | 59 |
| Figura 40. Plano de pantalla de la vista “Noticias” | 61 |
| Figura 41. Plano de pantalla y navegación de la vista “Beacon Scanner” | 62 |
| Figura 42. Captura de pantalla con la notificación de proximidad recibida en la bandeja de notificaciones del sistema iOS al entrar y salir del rango de alcance del beacon. | 63 |
| Figura 43. Diagrama de distribución de métodos, clases y frameworks para el desarrollo de la aplicación en iOS. | 64 |
| Figura 44. Flujo de componentes que influyen en el proceso de envío de mensajes mediante FCM..... | 66 |
| Figura 45. Campos disponibles para personalizar un mensaje desde la | |

| | |
|---|----|
| consola de FCM. | 68 |
| Figura 46. Campos opcionales para enviar datos personalizados clave-valor y opciones adicionales del compositor de notificaciones..... | 69 |
| Figura 47. Procedimiento para inclusión del archivo google-services.json al proyecto..... | 70 |
| Figura 48. Diagrama de distribución de métodos y clases para implementar Google Firebase Cloud Messaging. | 72 |
| Figura 49. Archivo GoogleService-Info.plist en el directorio raíz del proyecto Xcode. | 73 |
| Figura 50. Directorio del proyecto en Xcode al agregar el framework de Firebase mediante cocoapods. | 74 |
| Figura 51. Proceso de entrega de notificación remota hacia una aplicación cliente. | 76 |
| Figura 52. Habilitación de notificaciones push para crear el certificado para el servicio APN desde el portal de desarrolladores de Apple. | 76 |
| Figura 53. Certificado criptográfico firmado para servicios push. | 77 |
| Figura 54. Proceso para establecer una conexión confiable para notificaciones push basadas en tokens. | 78 |
| Figura 55. Diagrama de distribución de métodos y clases para implementar Google Firebase Cloud Messaging. | 79 |
| Figura 56. Captura de pantalla del modo de acceso protegido y desplegado para asegurar el acceso a los beacons en un entorno público..... | 80 |
| Figura 57. Proceso de firma de una aplicación utilizando la firma de aplicaciones de Google Play. | 82 |
| Figura 58. Proceso de firma de una aplicación al utilizar una clave de firma de aplicaciones propia. | 83 |
| Figura 59. Captura de pantalla obtenida de la consola de Google Play durante la creación de la ficha de Play Store para la aplicación. | 84 |
| Figura 60. Captura de pantalla obtenida del informe previo al lanzamiento de la aplicación en la consola de Google Play. | 85 |
| Figura 61. Captura de pantalla obtenida de la tienda Google Play que muestra a la aplicación publicada..... | 86 |

| | |
|---|----|
| Figura 62. Captura de pantalla tomada durante la validación de integridad del archivo de la aplicación. | 88 |
| Figura 63. Captura de pantalla del número de invitaciones e instalaciones de compilación de la versión beta de la aplicación en TestFlight. | 90 |
| Figura 64. Lista de beacons más visitados durante la etapa de desarrollo e implementación. | 92 |
| Figura 65. Cuadro de tendencia estadística del número de visitantes que ingresaron en el rango de los beacons en un periodo de tiempo ... | 93 |
| Figura 66. Gráfico de barras del porcentaje de usuarios que abrieron la notificación enviada durante el evento del 17 de mayo. | 94 |
| Figura 67. Listado del número de visitas generadas por aplicación (Android y iOS). | 95 |
| Figura 68. Gráfico de barras del porcentaje de usuarios que abrieron la notificación de prueba enviada a los usuarios de la aplicación en iOS. | 96 |

ÍNDICE DE TABLAS

| | |
|--|-----|
| Tabla 1. Evolución de la especificación Bluetooth..... | 9 |
| Tabla 2. Resumen de características de <i>Bluetooth</i> | 11 |
| Tabla 3. Propiedades de los paquetes de <i>advertising</i> | 20 |
| Tabla 4. Características técnicas de <i>beacons</i> de proximidad Estimote..... | 34 |
| Tabla 5. Estados de proximidad iBeacon. | 39 |
| Tabla 6. Comparación de protocolos iBeacon y Eddystone. | 400 |
| Tabla 7. Resumen de claves enviadas por la consola de <i>Firebase</i> y su descripción. | 69 |

1. INTRODUCCIÓN

1.1. Antecedentes

La tecnología *Bluetooth* ha ido evolucionando significativamente con el paso de los años, desde su lanzamiento con la versión 1.0 (2002) que ofrecía una velocidad de transferencia de 1 Mbps y alcance limitado a 1 metro, pasando por la versión 4.0 (2010) que es el estándar actual para todos los *smartphones* y dispositivos del mundo, con velocidades de hasta 32 Mbps, protocolos de bajo consumo energético (BLE) y alcance de hasta 100 metros (Bluetooth SIG, s.f.). Con la versión 5, lanzada en 2016, la tecnología *Bluetooth* pretende mejorar la velocidad de transferencia hasta en 100 Mbps, ofreciendo 4 veces más alcance que la versión 4.0, además de la coexistencia con otras tecnologías inalámbricas para evitar interferencias y brindar facilidades para aplicaciones de IoT.

Gartner Inc. (2017) afirma que: “se pronostica que 8.4 billones de aparatos estarán en uso alrededor del mundo en 2017, más del 31% con relación al 2016, y alcanzará 20.4 billones para el año 2020”, entre estos se encuentran aparatos integrados a IoT, *smartphones* y varios dispositivos inteligentes.

Los *beacons* son dispositivos *Bluetooth* de bajo consumo de energía que tienen variedad de usos como: sensores de proximidad, rastreo de clientes y envío de información. Esta tecnología permite determinar la ubicación física de un dispositivo utilizando para su comunicación un identificador universal único que es tomado por una aplicación compatible o un sistema operativo, de manera que se pueda entregar notificaciones en los dispositivos cuando estos estén en el rango de alcance.

La Facultad de Ingenierías y Ciencias Aplicadas (FICA) de la Universidad de Las Américas, se encuentra en la búsqueda de una manera interactiva de promocionar las nuevas carreras para aquellas personas que visiten nuestra sede, considerando los eventos que se realizan en las instalaciones, como: UDLA *Tech Day*, talleres vivenciales, día de admisiones, eventos, charlas, entre otras; por lo cual el uso de esta tecnología ayudará a alcanzar esta meta.

1.1.1. Alcance

Con este proyecto se busca desarrollar e implementar una aplicación móvil multiplataforma enfocada a promocionar las carreras de Software, Tecnologías de Información y Telecomunicaciones de la FICA durante eventos realizados por la Universidad, mediante el uso de *beacons* de proximidad que permitan mostrar al usuario información básica referente a duración, modalidad, ventajas, malla curricular y competencias de cada carrera de la facultad, haciendo uso de la interfaz gráfica de la aplicación.

Para el desarrollo de la aplicación se utilizará los entornos de desarrollo integrado (IDE) *Android Studio* para la aplicación en Android y *Xcode* para la aplicación en *iOS*; la metodología de programación a ser utilizada será en cascada, pasando por las fases de: recopilación de requisitos, diseño de interfaz, implementación, verificación y corrección de errores.

Se realizará la configuración de los *beacons* para que cada uno esté asociado a una carrera específica y la información puede ser mostrada dinámicamente en base a la cercanía del *smartphone* con el *beacon*. De esta forma también será posible enviar notificaciones *push* hacia los dispositivos con mensajes personalizados mediante la integración con *Google Firebase Cloud Messaging* por ejemplo: mensajes de agradecimiento por la asistencia o notificaciones sobre charlas durante un evento.

Además, permitirá al usuario registrar sus datos personales direccionándolo a la página de admisiones en caso de estar interesado en alguna de las carreras, de esta forma la información del postulante quedará disponible para que la Universidad en su debido momento pueda tomar contacto con el aspirante.

Finalmente, se realizarán las pruebas respectivas con el objetivo de comprobar la comunicación de los *beacons* con los *smartphones* al estar dentro del rango de proximidad, verificando que las notificaciones se estén implementando.

1.1.2. Justificación

En la actualidad tanto el uso de *smartphones*, como de aplicaciones han marcado una tendencia en la última década sobre nuestros estilos de vida, gracias a estos dispositivos tenemos al alcance de nuestra mano herramientas que facilitan nuestras tareas diarias, por esta razón podemos aprovechar estas ventajas y facilitar la interacción mediante aplicativos móviles directamente con los posibles alumnos durante las ferias que organiza la FICA para dar a conocer las competencias de las carreras de Software, Tecnologías de Información y Telecomunicaciones, así como también promover y motivar el ingreso de nuevos alumnos.

IoT ha ido ganando terreno en los últimos años gracias a la popularización de las tecnologías inalámbricas que brindan ventajas como: mayor alcance, incremento en las velocidades de transferencia de datos, capacidad mejorada de soporte de conexión simultáneo a varios dispositivos y menores interferencias. Estas razones demuestran que el camino está preparado para integrar aplicaciones de envío y recepción de información usando *beacons Bluetooth* de bajo consumo de energía (BLE). Además, el uso del aplicativo móvil permitirá brindar información específica al usuario sobre la carrera, al encontrarse en el rango de alcance de un beacon.

Mediante la aplicación se pretende alcanzar mayor difusión de las carreras de Software, Tecnologías de Información y Telecomunicaciones, dando a conocer la información relevante de cada carrera, motivando y facilitando la postulación de nuevos alumnos.

La implementación de esta solución presenta una forma innovadora de acercarse hacia el futuro estudiante, brindarle la posibilidad de tener en su propio dispositivo el perfil de la carrera y facilitarle el registro de sus datos personales para iniciar el proceso de admisión, generando una forma interactiva, rápida y que integre las tecnologías modernas.

1.2. Objetivo General

Desarrollar e integrar una aplicación móvil multiplataforma usando beacons de

proximidad, que permita mostrar notificaciones e información de las carreras de Software, Tecnologías de Información y Telecomunicaciones en *smartphones*.

1.3. Objetivos Específicos

- Analizar las tecnologías existentes para dispositivos *Bluetooth Low Energy* (BLE).
- Diseñar la red WPAN para la interconexión de los beacons.
- Programar los parámetros de proximidad de los beacons utilizando el SDK del fabricante.
- Desarrollar una aplicación móvil para la recepción de señales enviadas por los beacons utilizando los entornos de desarrollo integrado (IDE) de *Android Studio* y *Xcode* para *Android* y *iOS* respectivamente.
- Integrar la API de Google *Firebase Cloud Messaging* con la aplicación para el envío de notificaciones *push*.
- Implementar la solución propuesta, que incluye la aplicación desarrollada, las notificaciones y la programación de los beacons.
- Realizar el protocolo de pruebas de comunicación y la recepción de información.

2. ANÁLISIS DEL ESTADO DEL ARTE

2.1. Internet de las Cosas (IoT)

Internet es un sistema interconectado de dispositivos que utiliza el protocolo TCP/IP, con lo cual se facilita la comunicación en la red. Los protocolos de cada capa fueron diseñados para facilitar la transferencia de paquetes desde una dirección de origen hacia su destino. Con el avance de la tecnología inevitablemente más y más dispositivos buscan conectarse, por lo que hubo la necesidad de incrementar la cantidad de direcciones IP disponibles; es así como IPv6 fue desarrollada con el objetivo de soportar a un número casi infinito de dispositivos mediante direcciones de 128 bits. Este mecanismo de direccionamiento hace posible expandir el número de dispositivos conectados (Aftab, 2017).

dispositivos a realizar acciones de forma autónoma. Funcionalidades como la navegación automática en vehículos, o aspiradoras robóticas que limpian el hogar evitando obstáculos, no serían capaces de obtener estas habilidades de no ser por su constante aprendizaje sobre el entorno a su alrededor. La conectividad es una parte integral de estos dispositivos inteligentes (Aftab, 2017).

En alguno de los casos, un dispositivo de IoT puede contener más de un modo de conexión, de manera que pueda desempeñar múltiples tareas. Un ejemplo de ello son los smartwatches, que utilizan *Bluetooth* para comunicarse con el *smartphone* y Wi-Fi para utilizar las funcionalidades de Internet. En IoT, existe mayor énfasis hacia los estándares inalámbricos para asegurar su portabilidad.

2.2.1. Wireless Local Area Network (WLAN)

La red inalámbrica de área local o WLAN, implementa un sistema de comunicación de datos flexible. La transmisión y recepción de datos se realiza mediante radio frecuencia utilizando el aire como medio, minimizando las conexiones cableadas (Cisco, s.f.). WLAN está basada en el estándar IEEE 802.11, en el cual la comunicación se realiza de forma centralizada mediante un *router* o *hub* y los dispositivos que se conectan al punto de acceso inalámbrico llegan a ser los nodos que pueden conectarse libremente dentro del área de cobertura del punto de acceso que tiene una ubicación fija.

Wi-Fi (*Wireless Fidelity*) es un tipo de WLAN presente en el *hardware* de los dispositivos inalámbricos y es parte fundamental de IoT por su interoperabilidad en aparatos como *smartwatches*, reproductores de medios y vehículos autónomos que con frecuencia lo utilizan para transferir o recibir información de control remotamente (Aftab, 2017).

2.2.2. Red Ad-hoc inalámbrica

Esta tecnología se encuentra bajo el estándar IEEE 802.11, y a diferencia de WLAN no posee un *router* centralizado, ya que los dispositivos son responsables por el enrutamiento de los paquetes dentro de la red. Cada dispositivo se conecta en malla, brindando rutas alternativas para los paquetes

entrantes. Estas redes se auto organizan, de manera que, si un dispositivo abandona la red se reconfigura para reenviar los paquetes por nuevas rutas (Aftab, 2017).

Las redes Ad-hoc, también conocidas como redes P2P (*peer-to-peer*) presentan ventajas al ser rápidas, fáciles de implementar y escalables, pero también desventajas al ser propensas a ataques de red como "*Man-in-the-Middle*".

2.2.3. ZigBee

ZigBee es una red inalámbrica de malla (WMN) concebida bajo el estándar IEEE 802.15.4. La tecnología *ZigBee* trabaja en la red de área personal inalámbrica (WPAN), su enfoque son las aplicaciones que requieren comunicación con baja tasa de transmisión de datos y optimización de la vida útil de sus baterías. Por estas razones esta tecnología cumple con los requisitos para dispositivos de IoT en lo que respecta a vida útil de la batería, confiabilidad y flexibilidad de conexión (Aftab, 2017).

ZigBee tiene cobertura entre 10 a 100 metros para conectividad con línea de vista. Generalmente trabaja en la banda de 2.4 GHz con velocidades de transferencia de datos que varían desde los 20 Kbps hasta 300 Kbps. Esta tecnología aprovecha la capacidad de adaptarse a arquitecturas inalámbricas de estrella, árbol y malla (Aftab, 2017).

Una desventaja significativa yace en la adopción que tiene esta tecnología en otros dispositivos como *laptops* y *smartphones*, ya que al carecer de este módulo no es posible establecer comunicación directa y se depende de hardware adicional para poder controlar algún equipamiento que contenga un dispositivo *ZigBee* (Aftab, 2017).

2.2.4. Bluetooth

Bluetooth es un estándar global para tecnologías inalámbricas de corto alcance, bajo consumo de energía, bajo costo y tamaño reducido que permite la comunicación entre dispositivos mediante radio enlaces. Esta tecnología fue desarrollada en los laboratorios Ericsson en Suecia en 1994, se originó con el

objetivo de estudiar alternativas a ciertas conexiones cableadas al conectar dispositivos móviles con sus accesorios (Gupta, 2013).

Con el paso de los años, los usos de esta tecnología han crecido exponencialmente, desde el intercambio de archivos entre dispositivos móviles, computadores personales, impresión de documentos, manos libres, por nombrar algunos. Hoy en día, su principal uso se da entre teléfonos celulares, *tablets*, *laptops*, audífonos inalámbricos, parlantes, consolas de juego, periféricos e impresoras.

En la figura 2 se puede observar el proceso que lleva a cabo un dispositivo para establecer una conexión *Bluetooth*, donde el dispositivo que busca establecer la conexión debe contar previamente con *Bluetooth* habilitado, de esta forma se vuelve visible para otros aparatos que están cerca y es posible efectuar la conexión en topología maestro-esclavo.

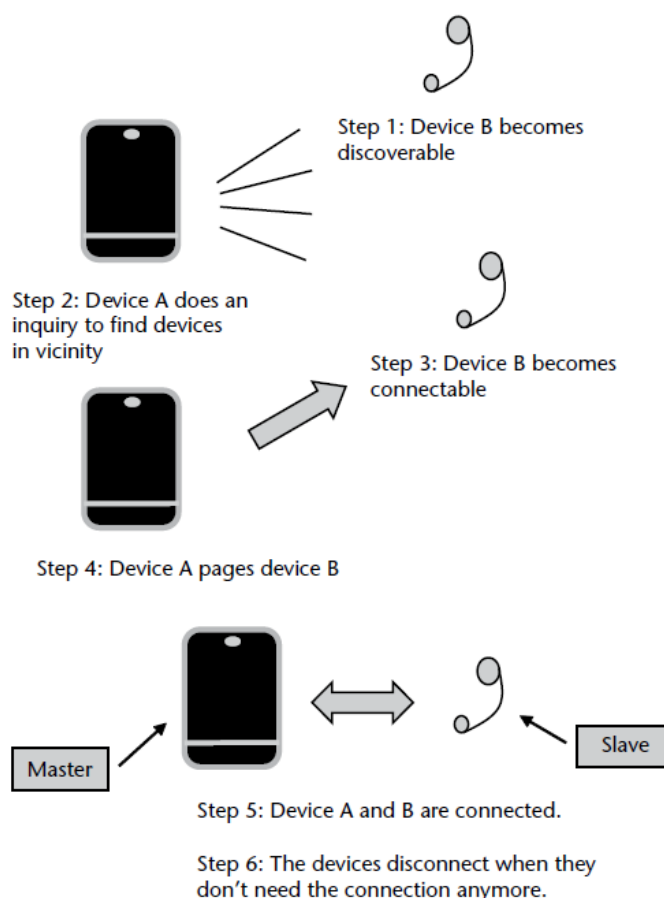


Figura 2. Proceso de una conexión Bluetooth
Tomado de Gupta, N., 2013, p. 30.

Algunas de las principales ventajas de *Bluetooth* son las siguientes:

- Seguras.
- Interoperable.
- Estándar global.
- Facilidad de uso.
- No requiere línea de vista.
- No sufre de interferencia de obstáculos, como paredes.
- Puede coexistir con otras tecnologías inalámbricas.

La tecnología *Bluetooth* está estandarizada por la norma IEEE 802.15.1 que define las especificaciones para conectividad inalámbrica de corto alcance para la capa física y de control de acceso al medio (MAC), dentro del grupo de trabajo del estándar IEEE 802.15 que describe los estándares de las redes WPAN (Aftab, 2017).

Bluetooth Special Interest Group (SIG) o *Bluetooth SIG*, es una organización privada, sin fines de lucro, encargada de publicar las especificaciones de *Bluetooth*, administrar el programa de calificación y masificar la tecnología inalámbrica *Bluetooth*.

Bluetooth soporta redes Ad-hoc. Esto quiere decir que no incurre en infraestructura preexistente como *routers* en redes cableadas o *Access Points* en redes inalámbricas. Los dispositivos intercambian datos de forma dinámica al estar dentro del rango de alcance (Aftab, 2017).

La tecnología *Bluetooth* ha ido evolucionando en varias especificaciones desde su concepción a lo largo de los años, cada una de las características de cada versión se muestra en la tabla 1.

Tabla 1.

Evolución de la especificación Bluetooth.

| Versión especificación | Fecha publicación | Características de la versión |
|------------------------|-------------------|-------------------------------|
|------------------------|-------------------|-------------------------------|

| | | |
|------------|----------------|--|
| 1.0 y 1.0a | Julio 1999 | Fueron las primeras versiones de la especificación <i>Bluetooth</i> . Su objetivo principal fue reemplazar los cables seriales por enlaces inalámbricos. |
| 1.0b | Diciembre 1999 | Esta versión agregó actualizaciones menores y arreglo de algunas fallas. |
| 1.1 | Febrero 2001 | <i>Bluetooth</i> fue ratificado como estándar IEEE 802.15.1-2002. |
| 1.2 | Noviembre 2003 | Esta entrega del estándar <i>Bluetooth</i> agregó las siguientes facilidades: <ul style="list-style-type: none"> • Salto de frecuencia adaptivo (AFH por sus siglas en inglés) provee mejor resistencia a la interferencia en ambientes ruidosos. • Enlaces eSCO para proveer mejor calidad de voz. Bajo la versión IEEE 802.15.1-2005 (la última versión emitida por la IEEE) la tecnología ha ido evolucionando de manera independiente. |
| 2.0 + EDR | Noviembre 2004 | El lanzamiento de esta versión introdujo mejoras en el desempeño utilizando EDR (<i>Enhanced Data Rate</i>). Esta versión incrementó el desempeño de 721 Kbps a 2.1 Mbps. La mejora favoreció a aplicaciones que requerían transferencia de datos veloz, navegación, impresión, etc. |
| 2.1 + EDR | Julio 2007 | Esta versión trajo consigo varias optimizaciones, además de la adición de SSP (<i>Secure Simple Pairing</i>) para simplificar el mecanismo de emparejamiento y mejorar la seguridad. |
| 3.0 + HS | Abril 2009 | Esta versión proporcionó un incremento significativo en el rendimiento, mediante la introducción del soporte para múltiples radiofrecuencias. Referidos como MAC/PHY Alterno (AMP). El <i>throughput</i> máximo soportado se disparó hasta los 24 Mbps. Con ello dispositivos como Laptops, celulares y <i>tablets</i> poseían ambos chips, tanto <i>Bluetooth</i> como 802.11. Esta versión de la especificación permitió conexiones usando <i>Bluetooth</i> y luego transfiriéndola al chip 802.11 para alcanzar altas velocidades de transferencia. |
| 4.0 | Junio 2010 | Esta versión tomó una dirección muy diferente a la de su predecesora. En versiones previas el enfoque estaba en introducir nuevas características y mejoras en el <i>throughput</i> , con esta versión se consideró a aquellos mercados que no requerían de alto rendimiento; pero si de muy bajo consumo de energía, conocido como <i>Bluetooth Low Energy</i> (BLE). |

Adaptado de Gupta, N., 2013, p. 20.

Bluetooth soporta una distancia máxima de hasta 100 metros, aunque generalmente esta tecnología es utilizada en distancias cortas. La

especificación *Bluetooth* brinda soporte para diferentes niveles de potencia basados en la aplicación que se pretende darle al dispositivo (Gupta, 2013).

En la tabla 2 se detalla un resumen de las características más relevantes de esta tecnología.

Tabla 2.

Resumen de características de Bluetooth

| Tipo de Conexión | Espectro ensanchado por salto de frecuencia (FHSS) |
|-----------------------------------|--|
| Espectro | 2.4 GHz. Rango regulatorio: 2400-2483.5 MHz. |
| Sato de frecuencia | 1600 saltos por segundo a lo largo de 79 canales RF. Canales separados por 1 MHz. |
| Modulación | GFSK |
| Potencia máxima de salida | 1 mW a 100 mW. |
| Potencia de transmisión | Nominal = 0dBm. Alcanza hasta 20 dBm con control de potencia. |
| Sensibilidad de receptor | -70 dBm al 0.1% BER |
| Velocidad máxima de datos | 721.2 kbps velocidad base. 2.1 Mbps (BT Espec. 2.0+EDR). 24 Mbps (BT Espec. 3.0+HS). |
| Alcance | 10 m hasta 100 m. |
| Topología | Hasta 8 dispositivos incluyendo 1 maestro y hasta 7 esclavos. |
| Canales de voz | 3 |
| Seguridad: Llave de autenticación | Llave de 128 bits. |
| Seguridad: Llave de encriptación | Configurable 8-128 bits. |
| Aplicabilidad | No requiere línea de vista. Destinado a funcionar en cualquier parte del mundo dado a que utiliza la banda no licenciada. |

Adaptado de Gupta, N., 2013, p. 20.

Bluetooth posee una arquitectura por capas. Como se puede observar en la figura 3.

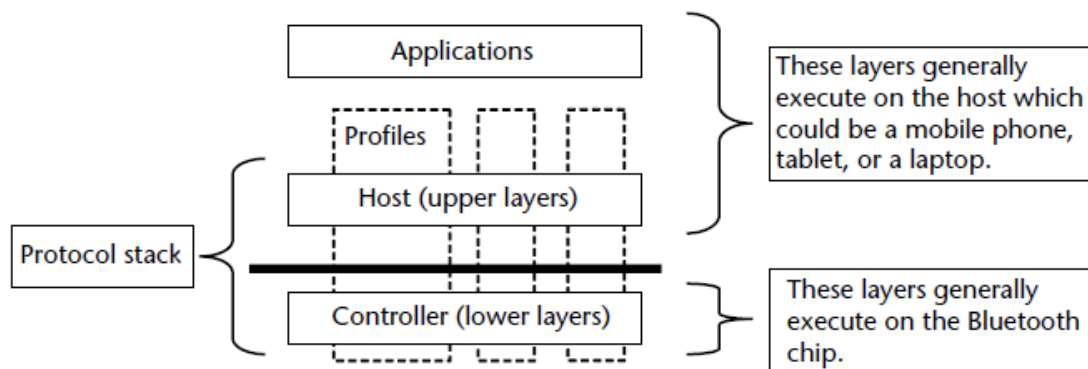


Figura 3. Arquitectura Bluetooth de alto nivel.
Tomado de Gupta, N., 2013, p. 22.

A nivel muy general la arquitectura *Bluetooth* está compuesta por los siguientes componentes:

- **Capas inferiores (*lower layers*):** son las responsables de desempeñar operaciones de bajo nivel como: descubrir dispositivos vecinos, establecer conexiones, intercambio de paquetes, seguridad, etc. Esta funcionalidad viene integrada generalmente en el chip *Bluetooth* y se la conoce como *Bluetooth Controller* (Gupta, 2013).
- **Capas superiores (*upper layers*):** utilizan las funcionalidades provistas por las capas de nivel inferior para brindar una funcionalidad más compleja, como la emulación de un puerto serial, transferencia de grandes cantidades de datos, *streaming* de audio, etc (Gupta, 2013).
- **Perfiles (*profiles*):** proveen información sobre como cada capa de la pila de protocolos se juntan para implementar un modelo de uso específico. Esto ayuda a garantizar que la implementación de un fabricante sea compatible con la de otro, en términos de interoperabilidad. Un dispositivo está en la capacidad de soportar uno o más de un perfil al mismo tiempo (Gupta, 2013).
- **Aplicación:** es la entidad encargada de establecer la interface hombre-máquina para hacer uso de las funcionalidades Bluetooth, entre algunos ejemplos podemos encontrar:

- Transferencia de archivos vía *Bluetooth*.
- Búsqueda de dispositivos *Bluetooth* vecinos y muestra de contenidos.
- Establecer conexión con un dispositivo al presionar un botón.

Uno de los puntos fuertes de la tecnología *Bluetooth* radica en reutilizar características que ya se encuentran disponibles, en vez de especificar todo desde cero. Es decir, toma componentes de estándares existentes, los adapta a sus necesidades, y solamente define los componentes centrales que son necesarios para las operaciones *Bluetooth*. Los protocolos pueden ser agrupados en dos categorías: Protocolos centrales y protocolos adoptados (Gupta, 2013).

Los protocolos definidos desde cero por *Bluetooth* SIG se los define como protocolos centrales. Algunos de ellos son:

- L2CAP.
- *Link Manager*.
- SDP.

En cambio, aquellos denominados protocolos adoptados son acogidos desde otros estándares, algunos ejemplos son:

- RFCOMM: protocolo adoptado del Instituto Europeo de Normas de Telecomunicaciones (ETSI).
- OBEX: protocolo adoptado del protocolo IrORBEX, definido por la Asociación de Datos Infrarojos (IrDA).
- TCS-BIN: utilizada para protocolos de telefonía.

En la figura 4 se puede observar la arquitectura detallada considerando la presencia de los protocolos centrales y adoptados.

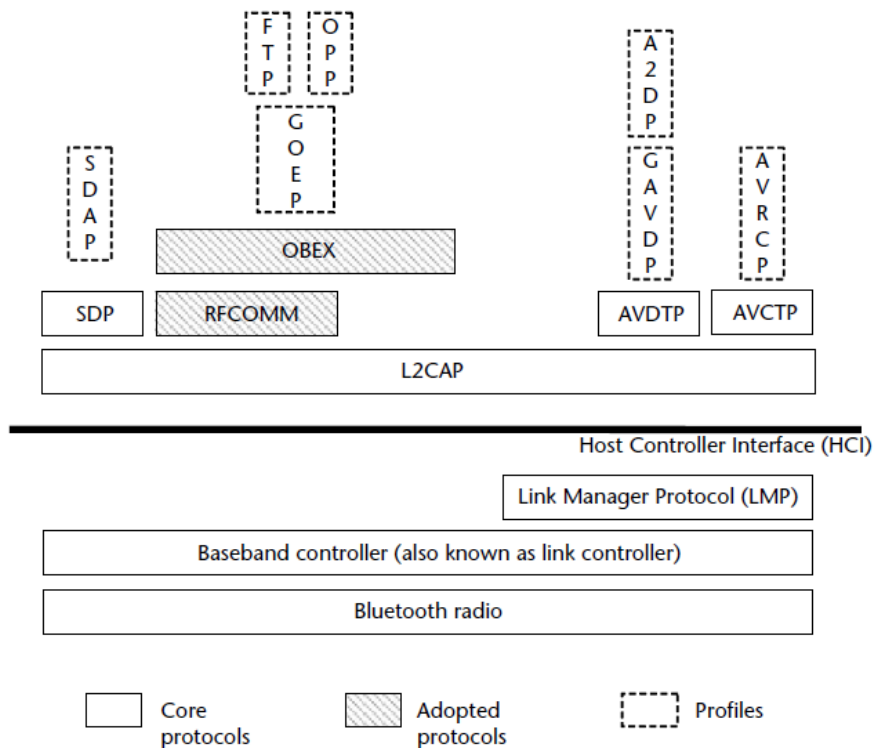


Figura 4. Arquitectura Bluetooth detallada.
 Tomado de Gupta, N., 2013, p. 23.

3.3. Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) contiene dos especificaciones centrales, estas son:

- Protocolo de Atributo (ATT)
- Perfil de Atributo Genérico (GATT).

El Protocolo de Atributos es una capa de bajo nivel que define como se transfieren los datos. Identifica el descubrimiento de dispositivos y atributos de lectura-escritura. Por otro lado, el Perfil de Atributos Genéricos se construye sobre el ATT para entregar al fabricante servicios de alto nivel al implementar *Low Energy* (LE). GATT define el rol de cliente o servidor en un dispositivo.

Como se muestra en la figura 5, la pila de protocolos BLE se divide en tres partes:

- **Aplicación:** como en otros tipos de sistemas, es la capa más alta y es la responsable de contener la lógica, la interfaz de usuario y el manejo de

los datos y todo aquello relacionado con el caso de uso que se implementa en la aplicación. La arquitectura de una aplicación es altamente dependiente en cada implementación (Davidson, Akiba, Cufí, & Townsend, 2014).

- **Host:** Incluye las siguientes capas:
 - Perfil de Acceso Genérico (GAP).
 - Perfil de Atributo Genérico (GATT).
 - Protocolo de adaptación y control de enlace lógico (L2CAP).
 - Protocolo de atributo (ATT).
 - Protocolo administrador de seguridad (SMP).
 - Interfaz *Host*-Controlador (HCI), del lado host.
- **Controlador:** incluye las siguientes capas:
 - Interfaz *Host*-Controlador (HCI), del lado controlador.
 - Capa de enlace (LL).
 - Capa física (PHY).

Cada uno de estos bloques de la pila de protocolos es separada en capas que proveen la funcionalidad requerida para operar.

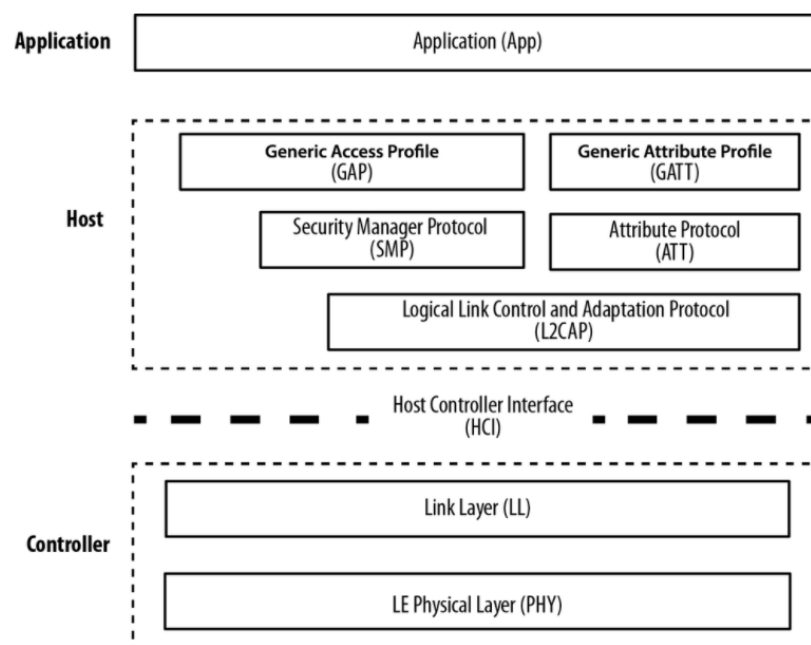


Figura 5. Pila de protocolos BLE.

Tomado de Davidson, R., Akiba, Cufí, C., & Townsend, K., 2014.

GATT es una entidad obligatoria en LE utilizada para descubrir servicios y características. Cualquier dispositivo que busque transferir datos sobre *Bluetooth* debe adoptar un rol basado en un requerimiento. Un dispositivo puede actuar como servidor, cuando recibe solicitudes del cliente, procesa y retorna los datos hacia el cliente. Por otro lado, el cliente es quien inicia la solicitud, mediante un proceso de descubrimiento de servicio para conocer los atributos del servidor. Una vez que los servicios se hayan identificado, se pueden leer y escribir los atributos. Los atributos son únicos, identificados por un Identificador Universal Único (UUID) de 128 bits (Aftab, 2017).

Un UUID es un identificador único de 128 bits, de los cuales usualmente 16 o 32 bits son utilizados para valores preasignados con propósito de registro como lo describe *Bluetooth SIG*. Los UUID son provistos en base a servicios o perfiles con un propósito específico (Aftab, 2017).

2.3.1. Jerarquía de perfiles Bluetooth basados en GATT

El Perfil Genérico de Atributos (GATT) describe la estructura mediante la cual un perfil intercambia datos entre dispositivos. Generalmente, un perfil está asociado a una tarea específica, mientras que los componentes de un perfil, que podrían considerarse como subtareas son los siguientes:

- **Servicio:** es una colección de características diseñadas para cumplir una tarea en particular.
- **Característica:** contiene valores, propiedades e información de configuración. Las propiedades de una característica definen el nivel de acceso que este otorga. Los descriptores son utilizados para definir el valor de las características (Bluetooth SIG, s.f.).

Como se puede observar en la figura 6, el nivel más alto de la jerarquía es el perfil, que está compuesto por uno o más servicios necesarios para cumplir con un caso de uso. Un servicio está conformado por características o referencias a otros servicios. Una característica consiste de: un valor, un conjunto de propiedades indicando las operaciones soportadas por la característica y un

conjunto de permisos relacionados a seguridad. Puede también incluir uno o más descriptores (metadatos o banderas de configuración relacionadas con la característica propietaria) (Bluetooth SIG, s.f.).

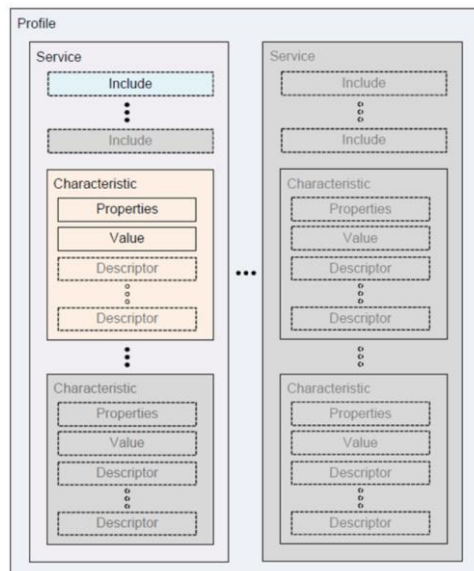


Figura 6. Jerarquía del perfil GATT.
Tomado de Bluetooth SIG., s.f.

2.3.2. Perfil de Acceso Genérico (GAP)

Bluetooth Low Energy especifica un perfil de acceso genérico que permite establecer la comunicación entre dispositivos *Bluetooth* de bajo consumo de energía. Este perfil es utilizado para unir cada una de las capas e identificar las capas inferiores desde la capa física hasta L2CAP.

El perfil genérico de acceso define también los roles utilizados en BLE, estos roles son importantes para comprender las bases sobre las cuales se basa esta tecnología. Desde el punto de vista de una implementación, un dispositivo BLE puede operar bajo un tipo de rol a la vez, aunque un escenario de múltiples roles solamente sería posible si los controladores subyacentes lo soportan (Aftab, 2017). A continuación, se describe cada rol que conforma este perfil:

- **Emisor:** es un rol que permite al dispositivo emitir de forma periódica o constante información. Un buen ejemplo de ello son los *beacons Bluetooth*, cuando todo lo que se necesita es emitir información. En teoría, el rol de emisor está dado a dispositivos solo de transmisión,

aunque en la práctica pueden ser otorgados a cualquier dispositivo *Bluetooth* que transmita y reciba. El emisor siempre transmite datos independientemente de quien esté escuchando. Los paquetes transmitidos son paquetes especiales de publicidad y no deben ser confundidos con paquetes de conexión del tipo cliente y servidor GATT.

- **Observador:** este rol está orientado a la escucha de paquetes enviados desde el emisor. Generalmente, están orientados a aplicaciones solo de recepción. El observador es capaz de leer los paquetes publicitarios y no soportan conexiones entrantes.
- **Central:** este rol está orientado hacia una arquitectura maestro-esclavo. Es capaz de manejar múltiples conexiones P2P al mismo tiempo. Un ejemplo de este rol puede ser un *smartphone*, que es capaz de conectarse a auriculares *Bluetooth* y al mismo tiempo a un reloj *Bluetooth*.
- **Periférico:** este rol hace las veces de esclavo en la arquitectura maestro-esclavo. Emite paquetes de publicidad para que la central pueda encontrarlo y responder a la solicitud de conexión.

2.3.3. Advertising y escaneo

BLE tiene solo un formato de paquete y dos tipos de paquetes (paquetes de *advertising* y datos), lo que simplifica inmensamente su implementación. Los paquetes de *advertising* tienen dos propósitos:

- Transmitir datos para aplicaciones que no necesitan establecer una conexión completa.
- Descubrir esclavos y conectarse a ellos.

Cada paquete de *advertising* puede llevar hasta 31 bytes de carga de datos de *advertising*, además de la información básica de cabecera (que incluye la dirección del dispositivo *Bluetooth*). Dichos paquetes están simplemente transmitiendo sin el conocimiento previo de la presencia de cualquier dispositivo de escaneo. Son enviados a una tasa fija definida por el intervalo de *advertising*, que va desde 20 ms a 10.24 segundos. Mientras más corto sea el intervalo, mayor será la frecuencia con la cual son transmitidos los paquetes, lo

que conduce a una mayor probabilidad de que estos paquetes sean recibidos por un escáner, pero también grandes cantidades de paquetes transmitidos significa mayor consumo de energía (Davidson, Akiba, Cufí, & Townsend 2014).

Debido a que *advertising* utiliza un máximo de tres canales de frecuencia y tanto el dispositivo anunciante, como el dispositivo escáner no están sincronizados de ningún modo, un paquete de *advertising* será recibido exitosamente por el escáner solo cuando se traslapen de manera aleatoria, como se muestra en la figura 7.

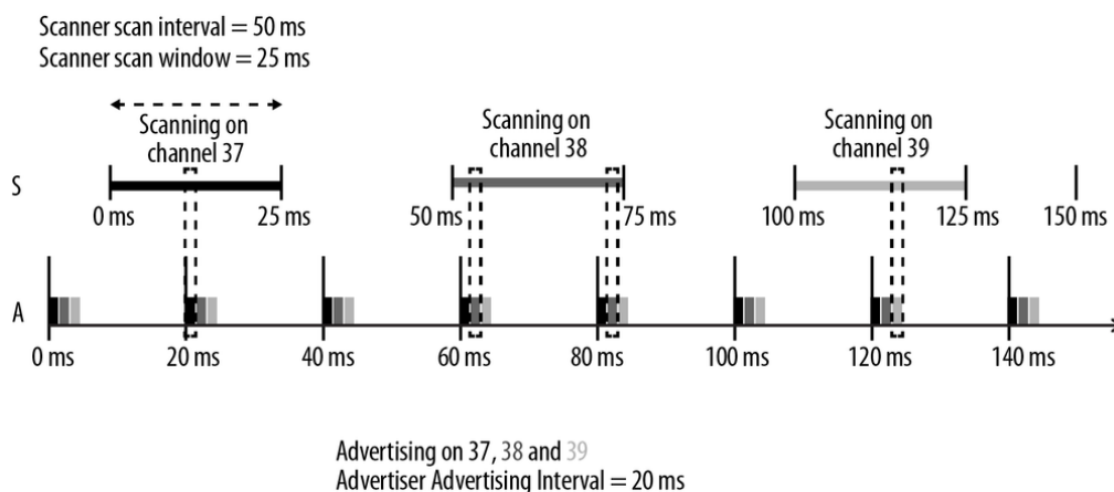


Figura 7. Proceso de *advertising* y escaneo en BLE.
Tomado de Davidson, Akiba, Cufí, & Townsend, 2014.

El intervalo de escaneo y la ventana de escaneo definen que tan frecuente y por cuanto tiempo un dispositivo de escaneo escuchará en búsqueda de potenciales paquetes de *advertising*.

La especificación define dos tipos básicos de procedimientos de escaneo:

- **Escaneo pasivo:** el escáner simplemente está a la escucha de paquetes de *advertising*, y el anunciante nunca está al tanto del hecho de que uno o más de un paquete fueron realmente recibidos por el escáner.
- **Escaneo activo:** el escáner emite un paquete de solicitud de escaneo (*Scan Request*) tras recibir un paquete publicitario. El anunciante lo

recibe y responde con un paquete de respuesta de escaneo (*Scan Response*). Este paquete adicional duplica la efectividad de la carga que el anunciante es capaz de enviar hacia el escáner, es importante tomar en cuenta que esto no significa que el escáner envíe algún tipo de información del usuario hacia el anunciante (Davidson, Akiba, Cufí, & Townsend, 2014).

En la figura 8 se puede apreciar la diferencia entre el escaneo pasivo y activo, tomando en consideración los intervalos de tiempo que toma escuchar en búsqueda de paquetes y cuando existen solicitudes y respuestas al escaneo.

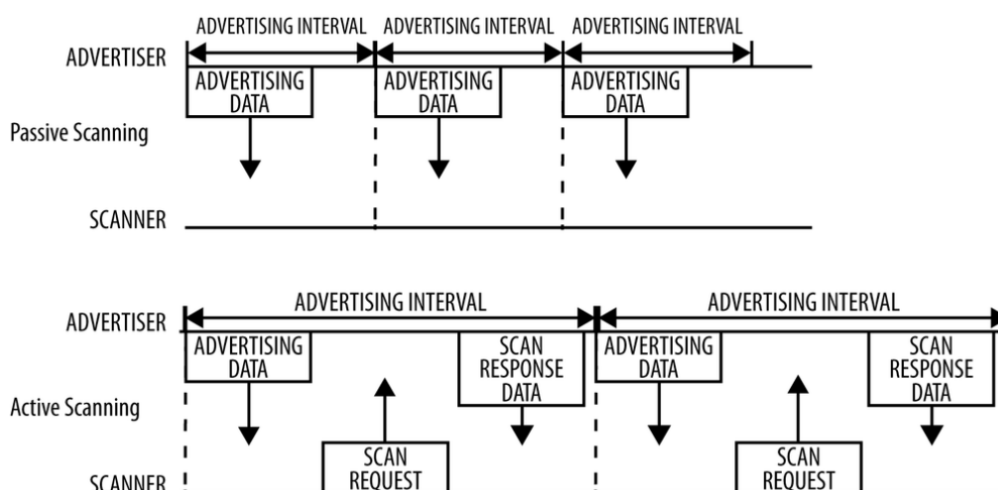


Figura 8. Escaneo activo y pasivo en BLE.

Tomado de Davidson, Akiba, Cufí, & Townsend, 2014.

Los tipos de paquetes de *advertising* pueden ser clasificados de acuerdo con tres diferentes propiedades, que se describen en la siguiente tabla:

Tabla 3.

Propiedades de los paquetes de advertising

| | | |
|--------------------------|---------------|---|
| Conectabilidad | Conectable | Un escáner puede iniciar una conexión tras recibir un paquete de <i>advertising</i> |
| | No conectable | Un escáner no puede iniciar una conexión (el paquete está destinado solo a transmisión) |
| Escaneabilidad | Escaneable | Un escáner puede emitir una solicitud de escaneo al recibir un paquete de <i>advertising</i> |
| | No escaneable | Un escáner no puede emitir una solicitud de escaneo al recibir un paquete de <i>advertising</i> |
| Direccionabilidad | Dirigido | Un paquete de este tipo solo contiene las direcciones Bluetooth del escáner de destino y del anunciante en su carga útil. No se permite datos de usuario. |

| | | |
|--|-------------|---|
| | No dirigido | Un paquete de este tipo no está dirigido a ningún escáner en particular, y puede contener datos de usuario en su carga útil |
|--|-------------|---|

Adaptado de Davidson, Akiba, Cufí, & Townsend, 2014.

2.3.4. Conexión

Para establecer una conexión, el dispositivo maestro comienza el escaneo en búsqueda de anunciantes que se encuentren aceptando solicitudes de conexión. Cuando se detecta un dispositivo esclavo adecuado, el maestro envía un paquete de solicitud de conexión y, una vez provista la respuesta del esclavo, se establece la conexión.

El paquete de solicitud de conexión incluye el incremento en el salto de frecuencia, que determina la secuencia de saltos que tanto el maestro como el esclavo deben seguir mientras dure la conexión (Davidson, Akiba, Cufí, & Townsend, 2014).

Como se puede observar en la figura 9, una conexión se resume en una secuencia de intercambio de datos entre el esclavo y el maestro en tiempos predefinidos. Cada intercambio se denomina evento de conexión.

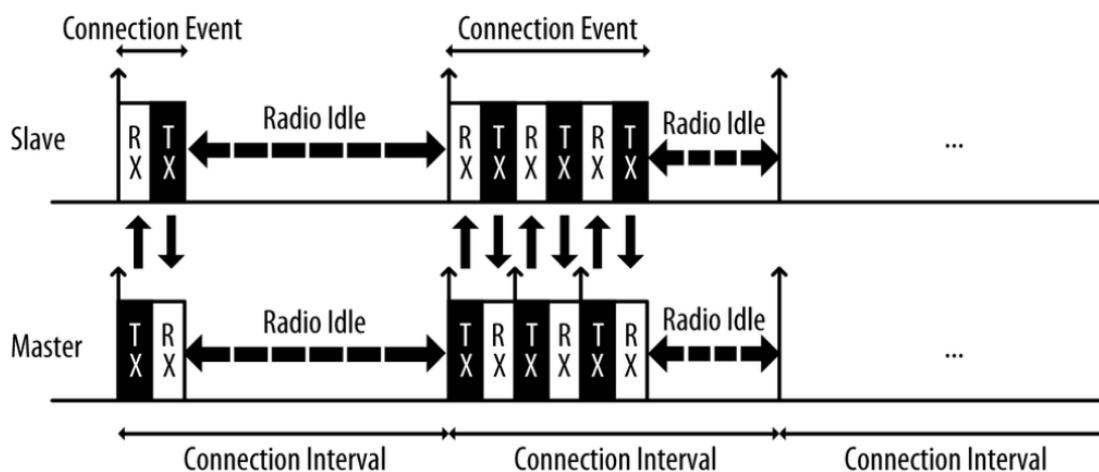


Figura 9. Eventos durante los intervalos de conexión (Maestro-Esclavo).

Tomado de Davidson, Akiba, Cufí, & Townsend, 2014.

2.3.5. Seguridad

Bluetooth Low Energy provee las siguientes características de seguridad que satisfacen los tres pilares de seguridad, que son: autenticación,

confidencialidad y autorización:

- Emparejamiento, proceso que genera llaves compartidas en ambos extremos conocidas como llaves de corto plazo (STK).
- Vinculación, es el proceso subsecuente del emparejamiento, donde la llave STK es almacenada para formar un par de confianza.
- Autenticación, cuando dos dispositivos identifican que poseen las mismas llaves.
- Encriptación, para confidencialidad de los mensajes.
- Integridad del mensaje, para evitar que los datos sean falsificados en caso de ataque.

2.3.6. Modelos de asociación en el emparejamiento

Los siguientes modelos describen las negociaciones que toman lugar en los dispositivos para generar las llaves de corto plazo (STK). *Bluetooth* LE provee cuatro modelos de asociación:

- **Comparación numérica:** este modelo está diseñado para dispositivos que poseen una pantalla, la cual es utilizada para mostrar un número de seis dígitos y permita corroborar con un sí o un no si los números mostrados en ambas pantallas son los mismos para establecer el emparejamiento (Aftab, 2017).
- **Just Works:** este modelo está orientado a dispositivos que no poseen una pantalla, por lo tanto, al usuario no se le muestra un código numérico. Sin embargo, sucede una comparación numérica y al usuario solamente se le solicita que acepte la conexión. Por ejemplo, al intentar emparejar un celular con un dispositivo manos libres *Bluetooth* (Aftab, 2017).
- **Fuera de banda (OOB):** este modelo es aplicado para escenarios donde los dispositivos son capaces de utilizar otro tipo de tecnología como NFC. Un ejemplo práctico del modelo se da al intentar enlazar un par de audífonos *Bluetooth* a un celular, el usuario debe presionar un botón en los audífonos para establecer el emparejamiento (Aftab, 2017).

- **Entrada clave:** este modelo se aplica en escenarios donde un dispositivo posee la funcionalidad de entrada de datos y otro la funcionalidad de pantalla. De esta forma al usuario se le muestra un número de 6 dígitos en el dispositivo con pantalla y se solicita ingresar dicho número en el otro dispositivo. Si el número ingresado es correcto, se establece el emparejamiento (Aftab, 2017).

2.4. Definición de las redes WPAN

Las redes inalámbricas de área personal (WPAN) son parte del grupo de trabajo IEEE 802.15 y engloba comunicación entre dispositivos personales como: teléfonos celulares, *tablets*, parlantes, impresoras, computadores de escritorio, entre otros; que comúnmente cubren distancias de hasta 10 metros y no requieren de altas tasas de transmisión de datos (Ramírez, 2006).

Las tecnologías inalámbricas poseen aplicaciones en varios escenarios que envuelven tasas de transferencia de datos desde algunos Kbps hasta varios Gbps, cada tecnología inalámbrica depende de una distancia específica para brindar cobertura como se puede observar en la figura 10.

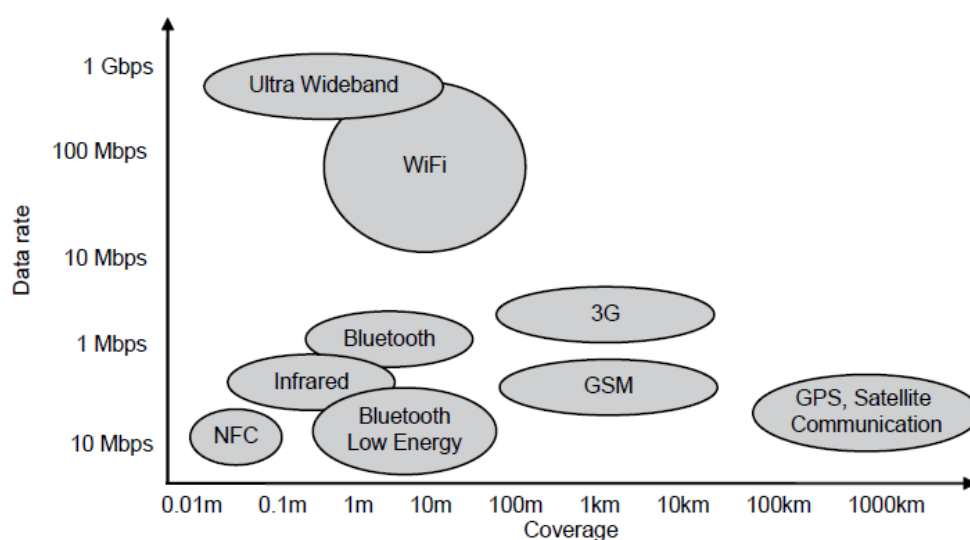


Figura 10. Cobertura y tasas de transferencia de varias tecnologías inalámbricas.

Tomado de Gupta, N., 2013, p. 3.

Como afirma Olivares (2009, p. 31): “La principal característica de este tipo de redes es que enfocan sus sistemas de comunicaciones a un área típica de 10

metros a la redonda que envuelve a una persona o a algún dispositivo, ya sea que esté en movimiento o no”. En comparación a una red inalámbrica de área local WLAN, una WPAN requiere de poca o casi nula infraestructura para conectarse directamente hacia el mundo exterior. Esta tecnología pretende utilizar de manera eficiente los recursos, por lo que existen protocolos simples diseñados con enfoque a satisfacer la necesidad de diferentes aplicaciones. En la red inalámbrica de área personal, el usuario guarda un vínculo directo con los dispositivos electrónicos en su poder o los que se encuentran en la proximidad de su entorno. El término red de área personal (PAN) nace para describir estas variantes en la conexión en red y de ahí su variante inalámbrica WPAN. La IEEE divide al grupo de estudio 802.15 en 4 grupos de trabajo que se encargan de desarrollar los estándares para satisfacer los requerimientos específicos de comunicación (Ramírez, 2006).

El grupo de trabajo 802.15.1 desarrolla el estándar en base a las especificaciones de *Bluetooth* SIG. El grupo de trabajo 802.15.2 realizó un modelo para la coexistencia entre WLAN y WPAN, así como aquellos dispositivos que las conforman. El grupo de trabajo 802.15.3 busca establecer un nuevo estándar de mayor velocidad para redes WPAN, de 20 Mbps o superior. Además, está desarrollado para ofrecer bajo consumo de energía, aplicaciones multimedia y soluciones a bajo costo. Finalmente, el grupo de trabajo 802.15.4 se encarga de la investigación y desarrollo de soluciones con baja transmisión de datos y por ende duración prolongada de las baterías en los dispositivos (Olivares, 2009).

2.4.1. Tipos de WPAN

Existen 3 grupos definidos por el grupo de trabajo IEEE 802.15 diferenciados por su tasa de transferencia de datos, consumo de energía y calidad de servicio (QoS).

- El grupo A utiliza la banda del espectro no licenciada de 2.4GHz, está dirigido a aplicaciones de baja transmisión de datos LR-WPAN. El consumo de energía del dispositivo es bajo, con capacidad de funcionar

varios meses o años sin carga (Al Agha, Pujolle, & Ali-Yahiya, 2015).

- El grupo B muestra un incremento en el desempeño, así como en el alcance, de hasta 100 Mbps en lo que respecta al control de acceso al medio (MAC) y de 10 m entre el transmisor y el receptor respectivamente. Está en capacidad de manejar una variedad de tareas que incluyen comunicación entre teléfonos celulares y QoS destinado a aplicaciones de voz (Al Agha, et al., 2015).
- El grupo C abarca las WPAN con tasas de velocidad elevada, para aplicaciones multimedia que precisen de altos niveles de QoS.

En una red WPAN un dispositivo instituye una conexión con una duración finita, es decir, si el elemento que hace las veces de maestro no participa, la red no funciona. Por ejemplo, cuando se establece la comunicación entre dos dispositivos para la transferencia de archivos se mantiene vigente la conexión mientras se lleve a cabo este propósito. Una vez finalizada, cualquiera de los dispositivos puede dar por terminada la conexión. La tecnología WPAN debe estar en capacidad de soportar una o más de una conexión, sea esta simultánea o individual de manera rápida y eficiente sin prescindir de la existencia de algún tipo de despliegue previo (Olivares, 2009).

Las redes WPAN por su capacidad de transmisión de datos, se dividen en dos grupos, orientados específicamente a aplicaciones que por un lado precisan de altas velocidades (HR-WPAN) o que por otro lado requieren bajo consumo de energía, pero no alta velocidad (LR-WPAN).

2.5. High Rate Wireless Personal Networks (HR-WPAN)

El estándar desarrollado por el grupo de trabajo IEEE 802.15.3 es sencillo, debido a la simplicidad que presentan los protocolos, el formato de las tramas, la modulación, entre otros; lo que hace la transmisión de datos más eficiente y por ende más rápida.

La seguridad de este estándar se maneja con encriptación compartida basada en el estándar AES. Su implementación se realiza de forma sencilla, gracias a un dispositivo coordinador que realiza el *handover* de manera dinámica.

2.5.1. Características de acceso al medio

Este estándar posee una topología tipo ad-hoc centralizada. El dispositivo controlador *piconet* (PNC) mantiene el control del ingreso de nuevos dispositivos, la sincronía y la asignación de tiempo para conexiones entre los dispositivos de la red (Olivares, 2009).

La comunicación entre dispositivos es de tipo P2P, con una arquitectura TDMA de super tramas. Como se muestra en la figura 11, las super tramas están compuestas por *Guaranteed Time Slots* (GTS), que consisten en 3 secciones de tiempo:

- **Beacon:** transmite información de control a la *piconet*, localización de recursos por trama y sincronización en tiempo.
- **Periodo de acceso de contención (CAP):** utilizado en los comandos de la trama durante el proceso de autenticación, solicitud y respuesta de emparejamiento, parámetros de flujo y negociación.
- **Periodo libre de contención (CFP):** cuenta con la opción de *Management Time Slots* (MTS) en reemplazo del CAP para tramas de comandos, está formado por ranuras GTS unidireccionales, asignadas por el dispositivo que hace las veces de maestro para la transmisión de datos de forma síncrona o asíncrona (Olivares, 2009).

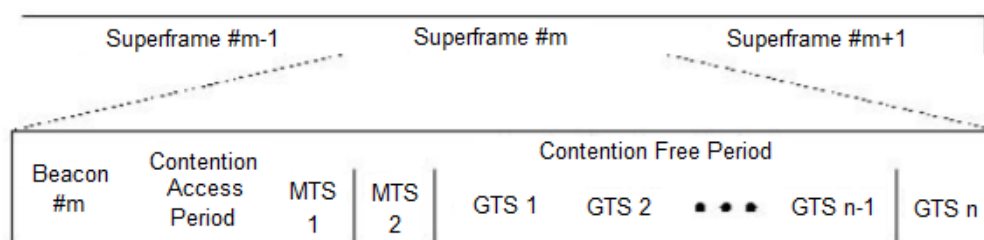


Figura 11. Estructura de las super tramas.
Tomado de Olivares, J. L., 2009, p. 47.

Soporta QoS multimedia y las negociaciones, así como el control de flujo se hacen en la capa de red; PNC solo se encarga de solicitudes de tiempo en el canal.

2.6. Low Rate Wireless Personal Networks (LR-WPAN)

El estándar IEEE 802.15.4 permite una baja transmisión de datos, además proporciona un tiempo de vida prolongado para fuentes limitadas de energía, como las baterías alcalinas. Existen dos tipos de dispositivos que participan en la red; *Full Function Device* (FFD), como su nombre lo indica, es dispositivo que cuenta con todas las funciones y *Reduced Function Device* (RFD), en cambio un dispositivo con funciones reducidas (Olivares, 2009).

Como se muestra en la figura 12, de acuerdo con la necesidad de la aplicación el estándar hace posible operar en una de las dos topologías que se describen a continuación:

- Tipo estrella, donde la comunicación se establece entre los dispositivos bajo un único coordinador de red, denominado coordinador PAN.
- *Peer-to-peer*, cualquier dispositivo está en capacidad de comunicarse con otro, siempre y cuando la distancia que existe entre estos se encuentre dentro del rango de comunicación. Esta topología facilita formaciones más complejas de red, como las redes tipo malla (*mesh*).

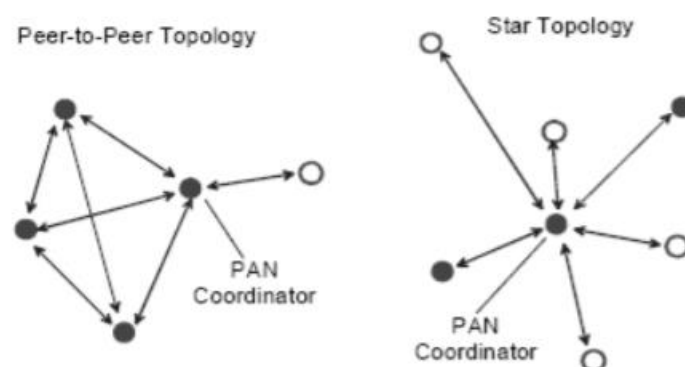


Figura 12. Topologías en LR-WPAN.
Tomado de Olivares, J. L., 2009, p. 52.

Dentro del estándar se plantean parámetros para la implementación, donde el funcionamiento de la red está ligado a su configuración, y en una red LR-WPAN es posible utilizar cualquiera de los dos mecanismos existentes de acceso al canal: sin *beacon* y con *beacon*.

2.6.1. Red sin Beacon

En una red sin *beacon*, se utiliza el mecanismo CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*), orientado a redes de bajo nivel que permite el uso de un mismo medio de transmisión a varios dispositivos. Previamente cada dispositivo anuncia de manera opcional su intención de transmitir, con el fin de evitar colisiones entre los paquetes de datos (Olivares, 2009).

Cuando un dispositivo busca transmitir en la red, verifica si existe una transmisión vigente sobre el canal por parte de otro dispositivo. Si este es el caso, intentará acceder nuevamente al canal, o en su defecto indicará una falla de conexión tras algunos intentos no exitosos. Como se indica en la figura 13, el proceso se realiza mediante una trama ACK (*Acknowledge*) que es encargada de confirmar una transmisión exitosa y enviada de forma inmediata como acuse de recibido tras la llegada de cada paquete de información (Olivares, 2009).

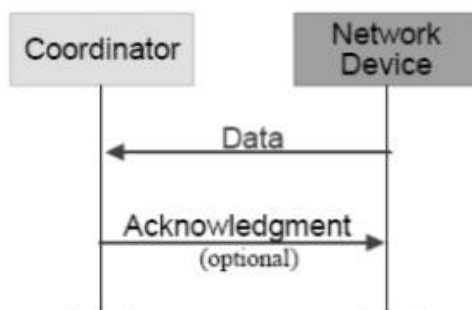


Figura 13. Proceso de transferencia de datos sin beacon.
Tomado de Olivares, J. L., 2009, p. 52.

2.6.2. Red con Beacon

En una red con *beacon*, se utiliza super tramas, la cual se divide en varios intervalos de tiempo, que permite múltiples accesos, acompañado del mecanismo CSMA/CA para evitar colisiones.

Como se observa en la figura 14, el proceso es similar a una red sin *beacon*, a diferencia que cualquier dispositivo que busque transmitir debe hacerlo mientras dure el periodo de acceso de contención (CAP), debe esperar al inicio

del siguiente time slot, tomando en consideración que no exista otro dispositivo transmitiendo en ese time slot. Si esto sucede, el dispositivo que está esperando para transmitir es replegado a un número aleatorio en los time slots (Olivares, 2009).

Cuando existen determinadas aplicaciones con requerimientos de ancho de banda específico o baja latencia, el coordinador PAN está en la potestad de dedicar porciones de la super trama a dichas necesidades. Las porciones dedicadas se denominan ranuras de garantía de tiempo (GTS) (Olivares, 2009). Como se puede observar en la figura 14, entre el coordinador y el dispositivo de red existe un tercer elemento (*beacon*) que influye en la comunicación que divide intervalos de tiempo para transmitir los datos.

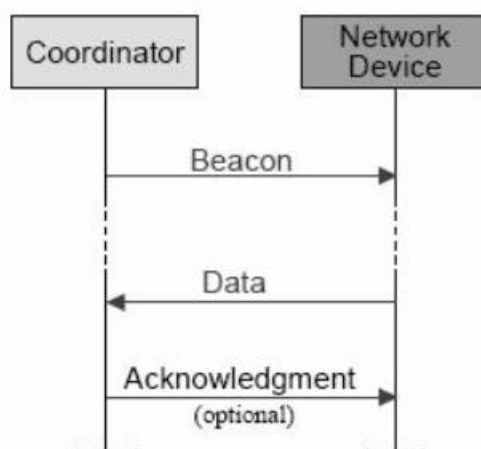


Figura 14. Proceso de transferencia de datos con beacon. Tomado de Olivares, J. L., 2009, p. 53.

2.7. Topología de red WPAN para BLE

Bluetooth SIG en su versión 4.0 define dos tipos de topología para *Bluetooth Low Energy*, y estos son:

- **Topología de conexión:** cuando se establece la conexión, dos dispositivos BLE intercambian paquetes de forma permanente y periódica. Se utilizan dos roles, maestro (central) y esclavo (periférico). Un dispositivo bajo el rol de maestro puede conectarse con diferentes dispositivos esclavos, formando una red en topología estrella. De acuerdo con la especificación BLE, el número de esclavos soportados

de manera simultánea por un maestro es ilimitada. Sin embargo, en dispositivos reales, las limitaciones de memoria hacen que este número se reduzca entre 4 a 8 dispositivos simultáneos. Tal como indica la figura 15, este modo de operación permite el intercambio de datos en ambas direcciones (Hortelano, Olivares, Ruiz, Garrido-Hidalgo, & López, 2017).

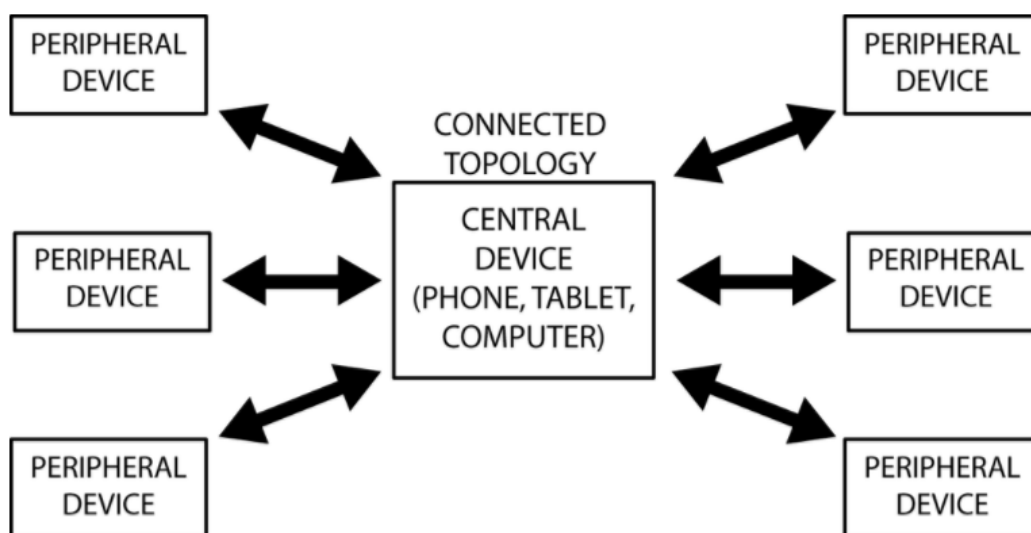


Figura 15. Topología de conexión.
Tomado de Davidson, Akiba, Cufí, & Townsend, 2017.

- **Topología broadcast:** la transmisión de datos de un dispositivo BLE utiliza el modo de *advertising* de BLE hacia cualquier dispositivo BLE que se encuentre en el rango de escucha, el cual utiliza el modo de escaneo de BLE. Como se muestra en la figura 16, esta topología posee dos roles definidos: *broadcaster* (emisor), encargado de transmitir los datos y observador, encargado de recibir e interpretar los datos. Para este caso el intercambio de datos se realiza de manera unidireccional, desde el *broadcaster* hacia uno o más observadores (Hortelano et al., 2017).

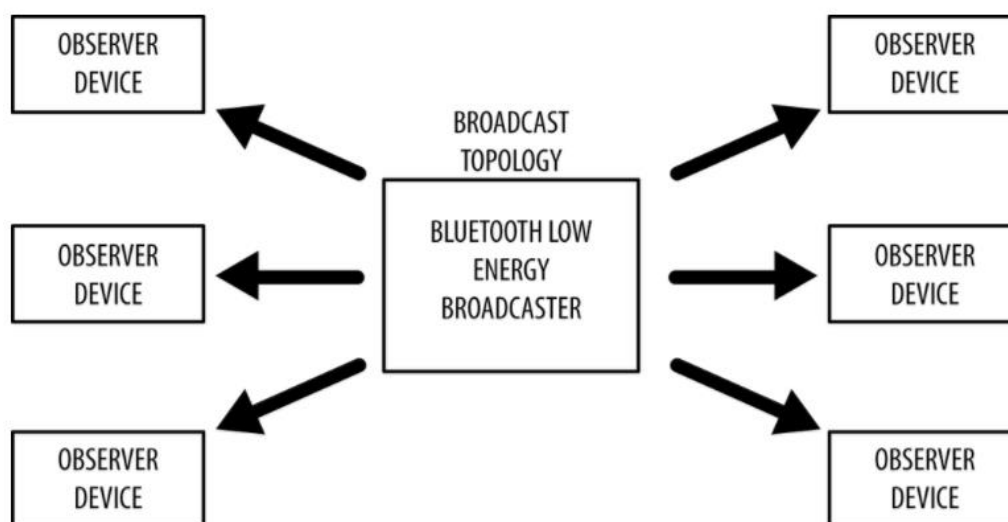


Figura 16. Topología Broadcast.

Tomado de Davidson, Akiba, Cufí, & Townsend 2017.

3. DISEÑO Y CONFIGURACIÓN DE LOS BEACONS

3.1. Diseño de la red

Ya que se ha abordado las diferentes configuraciones en las que puede funcionar una red WPAN, es necesario definir la topología que será utilizada en este proyecto, tomando en consideración que se cuenta con tres beacons BLE, los cuales estarán asociados a las carreras de Telecomunicaciones, Software y Tecnologías de la Información.

Como se puede observar en la figura 17, los beacons van a trabajar en topología *broadcast*, por lo que los beacons actuarán como dispositivos *broadcaster* (emisores) utilizando el modo de *advertising* de *Bluetooth Low Energy*. Los *smartphones* harán el papel de dispositivos observadores, tomando en consideración que precisan de *Bluetooth* habilitado para entrar en modo escaneo y cuando estén en el rango de alcance de un *beacon* podrán recibir el paquete de datos que está siendo transmitido. Los rangos configurados en los *beacons* se definen de la siguiente manera: cercano (1 metro de distancia) y lejano (5 metros). Mediante la aplicación móvil se programará el comportamiento en base a las configuraciones de alcance mencionadas que permitan mostrar notificaciones y contenido basados en la cercanía o lejanía del *smartphone* con el beacon.

Es necesario que los *smartphones* tengan conexión a Internet mediante red inalámbrica o red celular para que el contenido *web* pueda mostrarse, además de esta manera los dispositivos envían información estadística al portal *web* de *Estimote Cloud*. Esta plataforma permite gestionar la configuración de los beacons de la marca Estimote, utilizados en este proyecto, cuya descripción será ampliada posteriormente en el capítulo 2.3. Con estos datos es posible obtener información analítica sobre el número de visitantes que han permanecido dentro del alcance de cada beacon.

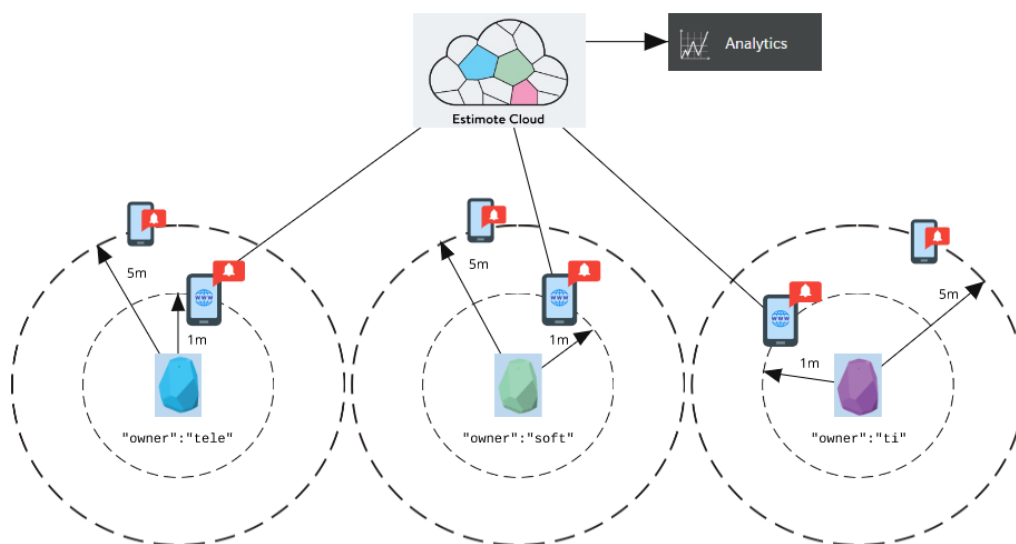


Figura 17. Diseño de la red WPAN en topología broadcast.

3.2. Introducción a los beacons Bluetooth Low Energy

Los *beacons* BLE son dispositivos *Bluetooth* que transmiten sus identificadores universales únicos (UUID) en proximidad. Son dispositivos de bajo consumo de energía ligados a un objeto físico (como un aparato electrónico o un muro), el cual detecta un dispositivo cercano, generalmente un *smartphone* y provoca cierto comportamiento. Estos comportamientos normalmente contienen una tarea en base a su ubicación o una transmisión única que puede ser preprogramada en los beacons antes de su despliegue. Los *beacons* asumen que cada persona posee un dispositivo BLE que ya se encuentra escuchando en busca de señales *Bluetooth* emitidas y que serán recibidas por estos (Aftab,

2017). Para este proyecto se utilizará *beacons Bluetooth* de la marca Estimote, que pueden apreciarse en la figura 18.



Figura 18. Beacons Bluetooth de Estimote.

3.3. Estimote Beacons

Estimote Inc. es una compañía fundada por Jakub Krzych y Lukasz Kostka en 2012, y su enfoque está orientado a la fabricación de dispositivos IoT de bajo consumo de energía. Utilizan múltiples tecnologías como *Bluetooth Low Energy*, Wi-Fi, HDMI y redes tipo malla. Los dispositivos que comercializa la marca se utilizan para proximidad, localización y localización con UWB (*Ultra Wide Band*) y generalmente incluyen diversos sensores; entre ellos, movimiento, temperatura, luz ambiente, presión y telemetría. Se especializan en *beacons* BLE con soporte completo a las plataformas Eddystone y iBeacon.

3.3.1. Características técnicas

Los *beacons* BLE de Estimote son compatibles con dispositivos *Android* y *iOS*, poseen baterías que generalmente duran entre 1 y 5 años, dependiendo del tipo de *beacon* que se utilice. Además, cuenta con las siguientes características resumidas en la siguiente tabla:

Tabla 4.

Características técnicas de beacons de proximidad Estimote

| | |
|--------------------------------------|--|
| MCU | <i>Bluetooth</i> SoC Procesador ARM Cortex M4 de 32 bits con FPU Velocidad del núcleo: 64 MHz Memoria flash: 512 kB Memoria RAM: 64 kB |
| Radio: transceptor de 2.4 GHz | Estándar <i>Bluetooth</i> 4.2 LE Alcance: hasta 70 metros Potencia de salida: -20 a +4 dB en pasos de 4dB, "modo silencioso" -40 dBm Sensibilidad: -96dBm Rango de frecuencia: 2400 MHz a 2483.5 MHz No. de canales: 40 Separación de canal adyacente: 2 MHz Modulación GFSK (FHSS) Antena: PCB, monopolo Ganancia de la antena: 0 dBi Tasa de transferencia de datos por-el-aire: 1 Mbps (soporta hasta 2 Mbps) |
| Sensores | Sensor de movimiento Sensor de temperatura |
| Características adicionales | NFC |
| Fuente de poder | Pila de botón 3.0V 1xCR2477 (reemplazable) |
| Tamaño y peso | Largo: 55 mm Ancho: 38 mm Alto: 18 mm Peso: 30 g |

Adaptado de Estimote, s.f.

3.3.2. Interfaz Estimote Cloud

Estimote provee una interfaz *web*, donde se pueden registrar los *beacons*, así como administrar su configuración. Además, es posible acceder a recursos para desarrolladores, foros, realizar análisis de los *beacons*, verificar a que aplicaciones están vinculados, etc.

Como se puede observar en la figura 19, la interfaz muestra los 3 *beacons* que de momento se encuentran registrados en la plataforma, así como los detalles de cada uno, con su nombre, identificador, el tipo de paquete que está transmitiendo y su ubicación.

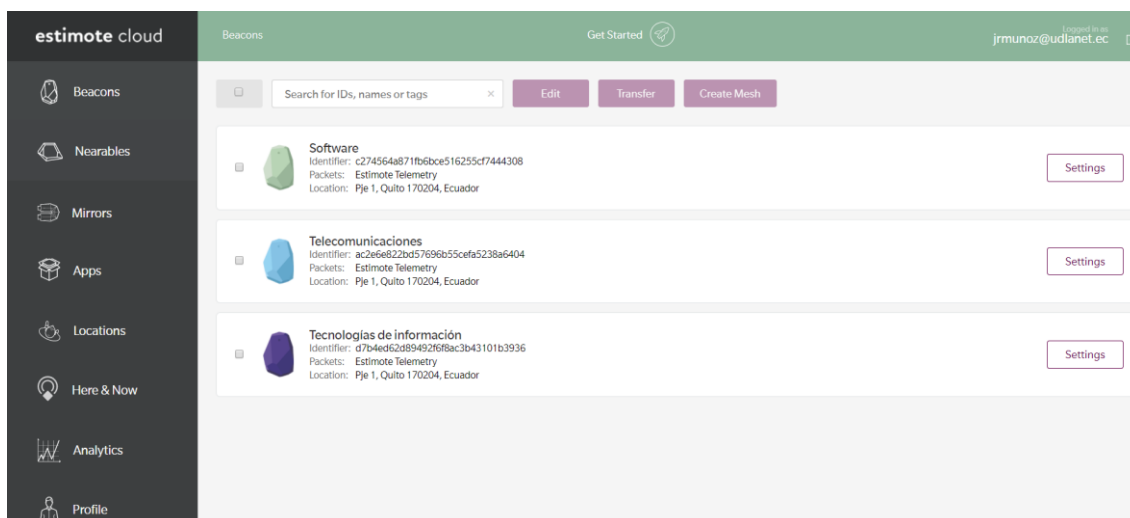


Figura 19. Visualización de beacons en la plataforma Estimote Cloud.

Tal como muestra la figura 20, al acceder a la configuración de cada *beacon* se puede obtener información adicional detallada, como el tipo de paquete que está transmitiendo, características adicionales habilitadas, tiempo restante de la batería, temperatura, entre otras.

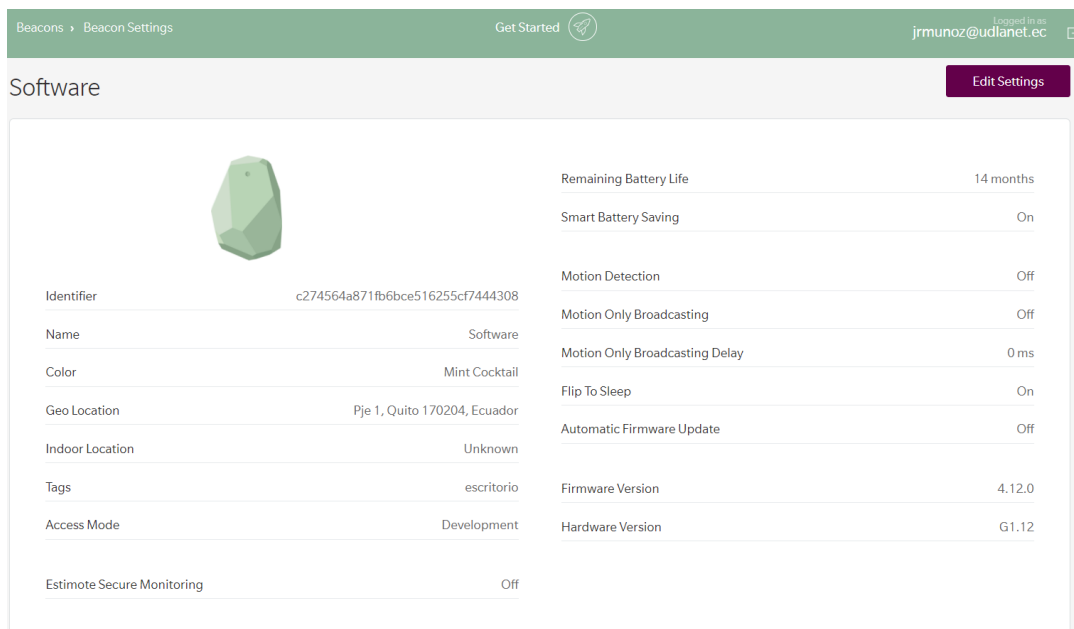


Figura 20. Visualización de información adicional del beacon Software en la plataforma Estimote Cloud.

Si bien se puede acceder a editar ciertas configuraciones para cada *beacon* desde la plataforma *web*, es necesario utilizar la aplicación móvil de Estimote para que los cambios realizados desde el portal de Estimote Cloud surtan efecto en los *beacons*.

3.4. Protocolos de los beacons

Como en cualquier otra tecnología, los *beacons* BLE utilizan un conjunto de protocolos, que no deben ser confundidos con el estándar de protocolos de *Bluetooth Low Energy*. Estos protocolos le dan una definición a como es transmitida la información, generalmente, desde un *beacon* hacia un *smartphone* que maneja el mismo protocolo. Ambos extremos deben conocer la definición de los datos transmitidos para conseguir una transmisión exitosa. Por esta razón, el tipo de dato de transmisión es definido por el tipo de protocolo que utiliza el *beacon* (Aftab, 2017). Estos protocolos están presentes de forma general en todo tipo de beacons.

3.4.1. Google Eddystone

Google Eddystone es un protocolo *open source* creado por Google, compatible con *Android* y *iOS*. Posee tres tipos de paquetes:

- Eddystone-UID
- Eddystone-URL
- Eddystone-TLM

Una vez que el objeto o el lugar es identificado para el despliegue de los *beacons*, el siguiente paso es escoger el tipo de datos que el *beacon* transmitirá, pudiendo ser cualquiera de los 3 tipos de paquetes disponibles para Eddystone.

3.4.1.1. Eddystone-UID

Eddystone-UID es un tipo de dato soportado por los *beacons* bajo el protocolo Eddystone. Es una trama de transmisión que consiste de un ID de 16 bytes, de los cuales 10 bytes se utilizan para el espacio de nombre (*namespace*) y 6 bytes para la instancia. El *namespace* se utiliza para identificar un grupo de *beacons* Eddystone, mientras que la instancia es utilizada para identificar un *beacon* en particular. Ambas partes del UID asegura la unicidad del *beacon* o del grupo de *beacons* (Aftab, 2017). La estructura de la trama Eddystone-UID se muestra en la figura 21.

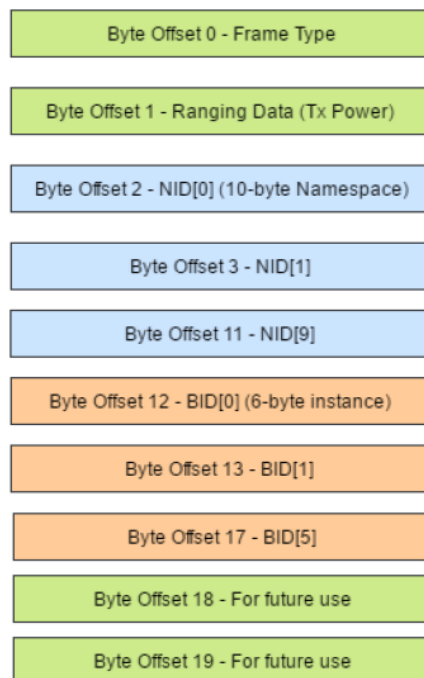


Figura 21. Estructura de trama Eddystone-UID.
Tomado de Aftab, M., 2017, p. 115.

3.4.1.2. Eddystone-EID

Eddystone-EID es otro tipo de dato soportado por los *beacons* bajo el protocolo Eddystone. La trama transmite un identificador encriptado efímero (EID) que cambia periódicamente. Esta característica es añadida al *beacon* para aplicaciones que requieran que los datos sean transmitidos de manera segura. En caso de un ataque hombre en el medio, la trama de datos interceptada cambiará aleatoriamente (Aftab, 2017). La estructura de la trama Eddystone-UID se muestra en la figura 22.

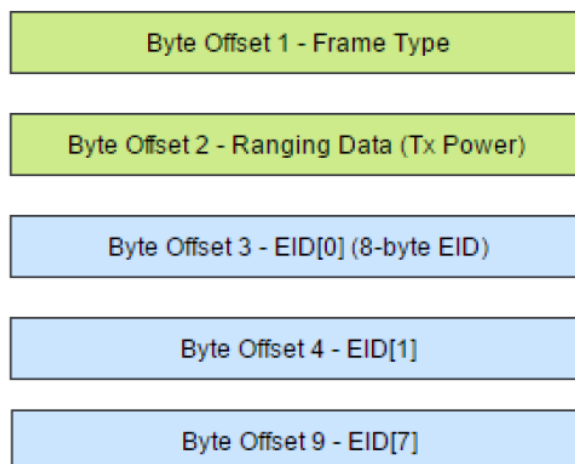


Figura 22. Estructura de trama Eddystone-EID.
Tomado de Aftab, M., 2017, p. 116.

3.4.1.3. Eddystone-URL

El formato de dato Eddystone-URL es la columna vertebral de la *web* física de Google. La trama transmite un URL codificado por Google (URL corto), lo que permite transmitir más información utilizando menos caracteres. Cualquier dispositivo con acceso a internet será capaz de decodificar el URL y abrir la página web codificada en el URL, sin necesidad de una aplicación en los dispositivos de Google y con aplicaciones como *Physical Web* en dispositivos con *iOS* (Aftab, 2017). La estructura de la trama Eddystone-URL se muestra en la figura 23.

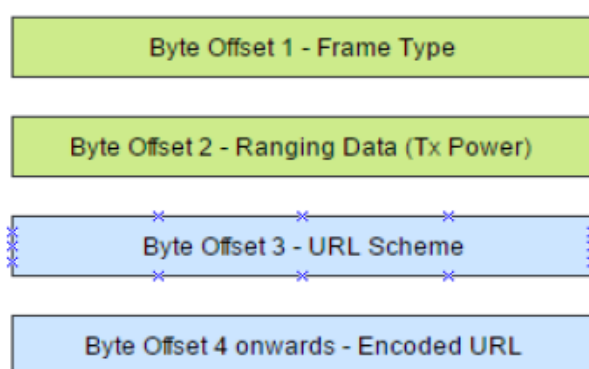


Figura 23. Estructura de la trama Eddystone-URL.
Tomado de Aftab, M., 2017, p. 117.

3.4.2. Apple iBeacon

Apple iBeacon es una plataforma para beacons utilizado para aplicaciones *Bluetooth* de proximidad. Los dispositivos *iOS* pueden identificar el momento en

el que están en el rango de alcance de un *beacon* emitiendo paquetes en este formato (Apple, 2014). A diferencia de Eddystone, iBeacon es un protocolo nativo de Apple, por lo que es exclusivo para dispositivos *iOS*. El paquete de *advertising* en iBeacon consiste de tres componentes:

- **UUID:** identificador universal único de 16 bytes
- **Major ID:** cadena de 2 bytes para identificar a un grupo de *beacons*; similar al espacio de nombres en Eddystone.
- **Minor ID:** cadena de 2 bytes para identificar *beacons* individuales; similar a la instancia en Eddystone.

Antes de *iOS 7*, la ubicación utilizaba regiones definidas por una ubicación geográfica (latitud + longitud), conocido como geolocalización. iBeacon brinda un nuevo nivel de flexibilidad al definir regiones con un identificador. Esto permite a los *beacons* fijarse a objetos que no están atados a una sola ubicación (Apple, 2014).

3.4.2.1. Rango de alcance

iOS 7 introduce un nuevo grupo de API's para determinar la cercanía aproximada a un dispositivo utilizando tecnología iBeacon, proceso conocido como rango de alcance. En escenarios de uso común, *iOS* aplica filtros para realizar una estimación de precisión con el fin de determinar la proximidad a un *beacon* (Apple, 2014). Esta estimación se obtiene utilizando cuatro estados de proximidad que se muestran en la siguiente tabla:

Tabla 5.

Estados de proximidad iBeacon.

| Estado de proximidad | Descripción |
|--------------------------------|--|
| Inmediato (<i>Immediate</i>) | Representa un alto nivel de certeza de que el dispositivo se encuentra físicamente muy cerca al <i>beacon</i> . |
| Cerca (<i>Near</i>) | Indica una proximidad de 1 a 3 metros. En caso de que existieran obstrucciones entre el dispositivo y el beacon que puedan causar atenuaciones en la señal, el estado Cerca podría no ser detectado, aunque el dispositivo esté en rango. |
| Lejos (<i>Far</i>) | Este estado indica que el beacon puede ser detectado pero la certeza en su precisión es muy baja para determinar si está cerca o inmediato. Una consideración importante es que el estado "Lejos" no necesariamente implica "no físicamente cerca" |

| | |
|--------------------------------|---|
| | al <i>beacon</i> . Se debe confiar en la propiedad de precisión (<i>accuracy</i>) para determinar la proximidad potencial al <i>beacon</i> . |
| Desconocido (<i>Unknown</i>) | La proximidad al <i>beacon</i> no puede ser determinada. Este estado podría indicar que no existen medidas suficientes para determinar su estado. |

Adaptado de Apple, 2014.

3.4.3. Apple iBeacon vs Google Eddystone

En la tabla 5 se muestra un resumen con comparaciones y diferencias entre los protocolos que rigen a los beacons *Bluetooth Low Energy*, en base a su compatibilidad, facilidad de uso, seguridad y tipos de datos soportados.

Tabla 6.

Comparación de protocolos iBeacon y Eddystone.

| | Compatibilidad | Perfil | Tipo de dato | Facilidad de uso | Seguridad |
|-------------------------|--|------------------------------|--------------------------------------|---|---|
| Apple iBeacon | Compatible con <i>Android</i> y <i>iOS</i> , pero solo nativo de <i>iOS</i> | Protocolo propietario | UUID con <i>Major</i> y <i>Minor</i> | Simple de implementar | No posee un tipo de dato seguro como EID de Eddystone |
| Google Eddystone | Compatible con <i>Android</i> y <i>iOS</i> , soportado por cualquier <i>beacon</i> BLE | Protocolo <i>open source</i> | UID, EID, URL y TLM | Simple de implementar, pero requiere programación en escenarios complejos | Seguridad de datos usando EID |

Adaptado de Aftab, M., 2017, p. 118.

3.5. Configuración de beacons desde la aplicación móvil de Estimote

La aplicación móvil de Estimote es un complemento de *Estimote Cloud*, que permite administrar desde el *smartphone* las características más relevantes para la configuración de un *beacon*, como el tipo de paquete que va a transmitir, el intervalo de *advertising* del paquete y la potencia de transmisión.

Como se observa en la figura 24, es posible administrar la configuración del paquete que está transmitiéndose desde la aplicación móvil de Estimote. Entre las opciones encontramos tres configuraciones propietarias del fabricante:

- **Monitoring:** es una propiedad que permite mejorar el monitoreo al contabilizar la cantidad de dispositivos que se conectan al *beacon*, y así alimentar información al portal de *Estimote Cloud*.
- **Indoor Location:** es una propiedad que está presente en los *beacons*

de Estimote de la gama UWB y de largo alcance para posicionamiento en interiores.

- **Telemetry:** esta propiedad permite enviar hacia *Estimote Cloud*, datos de los sensores incluidos en el *beacon*, ya sea para conocer el estado de la batería o las fluctuaciones de temperatura.

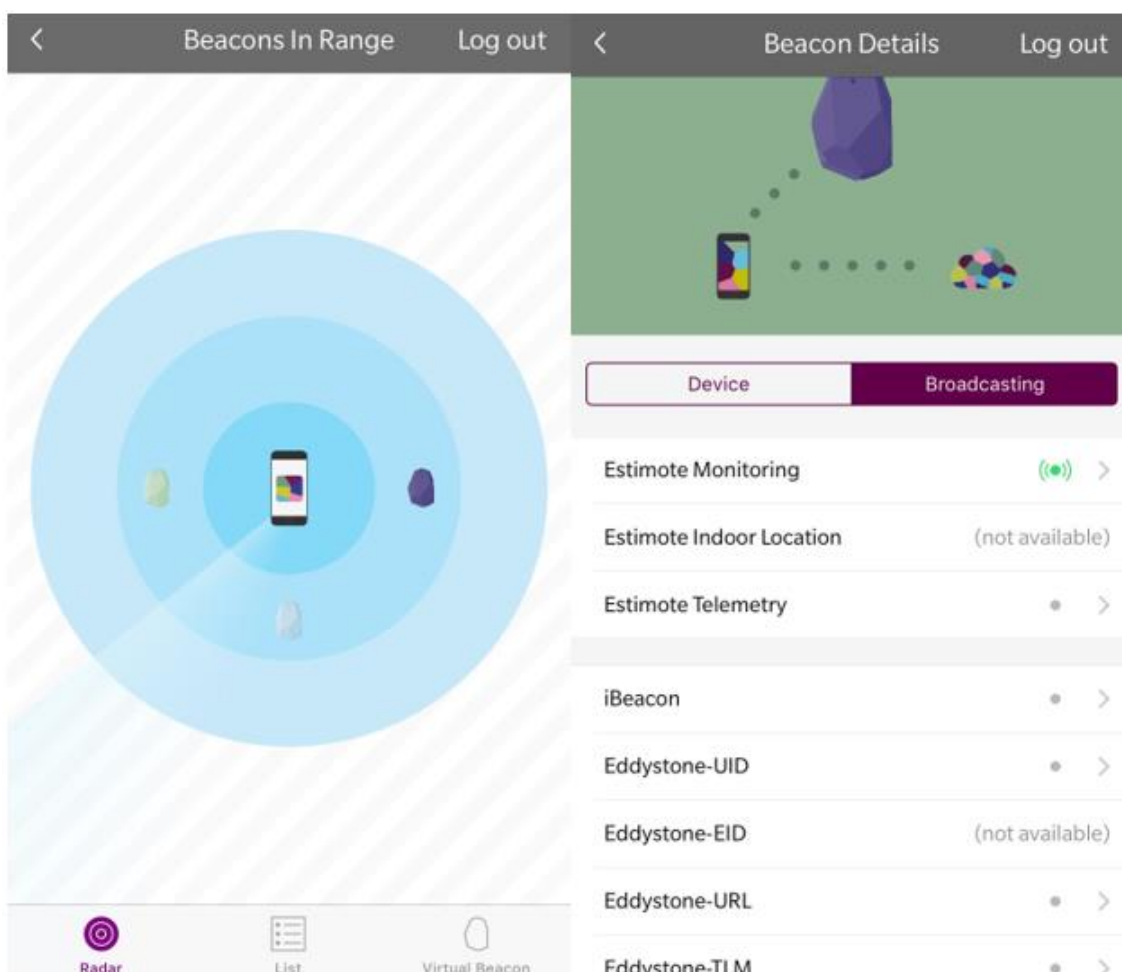


Figura 24. Visualización de *beacons* y sus configuraciones.

Además de estas opciones, también es posible configurar adicionalmente los paquetes de iBeacon y Eddystone, su intervalo de *advertising* y la potencia de transmisión. Un intervalo de *advertising* menor representa en un consumo mayor de la batería del *beacon*, pero mejora la precisión y respuesta. Con la potencia de transmisión sucede lo contrario, una mayor potencia de transmisión se traduce en un mejor alcance, pero también afecta a la duración de la batería.

Como ya se conoce, iBeacon maneja 3 componentes: UUID, *Major ID* y *Minor ID*, estos datos pueden ser editados para ser utilizados posteriormente al momento de realizar la aplicación móvil. También es posible habilitar las opciones de seguridad de UUID, intervalos de *advertising* y potencia de transmisión del *beacon*. En la figura 25 se puede visualizar la información del paquete iBeacon y sus opciones editables desde la aplicación de Estimote para dispositivos móviles.

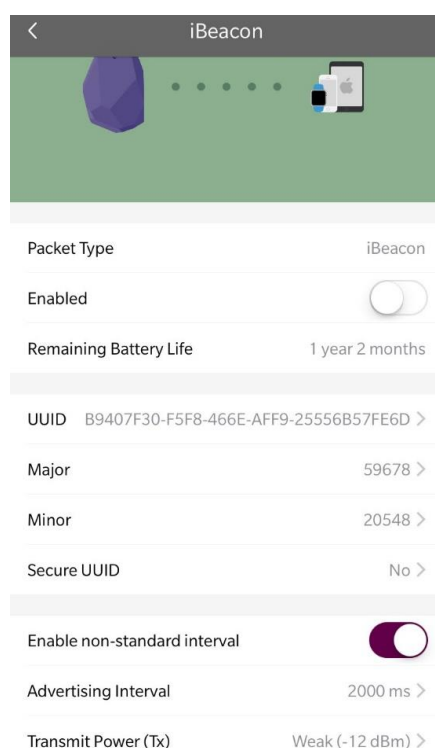


Figura 25. Visualización de las opciones disponibles para iBeacon en Estimote App.

En el caso de Eddystone se manejan tres tipos de paquetes: Eddystone-UID, Eddystone-URL y Eddystone-TLM, cada uno con sus características específicas. Para Eddystone-UID es posible editar el dato de *namespace*, la instancia, además del intervalo de *advertising* y la potencia de transmisión. Para Eddystone-URL están habilitadas las opciones de edición del URL en el que se puede ingresar un URL corto validado por Google para que el *beacon* transmita una dirección *web* específica sin necesidad de aplicaciones de terceros para dispositivos *Android* y necesariamente con una aplicación para dispositivos *iOS*, además del intervalo y potencia de transmisión. En la figura

26 se puede apreciar las opciones configurables para cada tipo de paquete Eddystone descrito anteriormente.

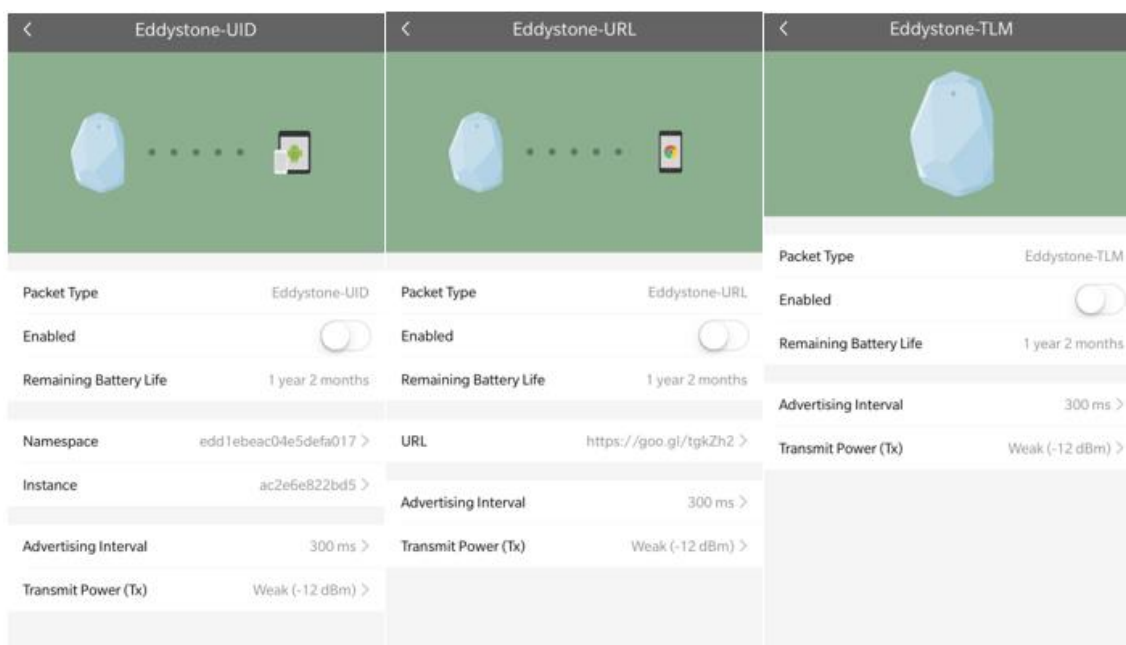


Figura 26. Visualización de opciones disponibles para Eddystone UID, URL y TLM.

Los *beacons* Estimote permiten utilizar solo una de las dos opciones de paquetes de transmisión, sean estos iBeacon o Eddystone. Todos los cambios realizados desde la aplicación se sincronizan con *Estimote Cloud* y viceversa. Esta es la razón fundamental por la cual ambas herramientas son necesarias, ya que ciertas computadoras antiguas pueden no disponer de *Bluetooth* integrado y la aplicación móvil hace las veces de intermediario para aplicar las configuraciones a los *beacons*. Además, acceder desde una interfaz web simplifica el proceso de configuración de los beacons sin necesidad de instalar software adicional en el computador.

Debido a que la aplicación va a ser utilizada en las plataformas tanto *Android* como *iOS*, la mejor opción será habilitar los paquetes Eddystone-UID que son *open source* y pueden ser utilizados en ambos dispositivos. La siguiente configuración será el intervalo de *advertising* que puede establecerse en 300 milisegundos para optimizar el descubrimiento del *beacon* por parte de otros dispositivos. Finalmente, la potencia de transmisión será establecida en -4dBm, que permite un alcance de hasta 50 metros.

4. DESARROLLO DE LA APLICACIÓN

4.1. Diagrama de flujo de la aplicación

Parte fundamental para comprender la forma en que trabaja la aplicación es apoyar su funcionamiento en un diagrama de flujo que permita describir el proceso que va a desarrollarse internamente en el dispositivo para cumplir con los requisitos previos para que trabaje la aplicación.

El diagrama de flujo que se muestra en la figura 27 describe el proceso que se llevara a cabo cuando la aplicación móvil se ejecute en el *smartphone* tanto en *Android*, como en *iOS*. Al iniciarse por primera vez solicitará al usuario el permiso para encender *Bluetooth*, ya que esta tecnología es el principal medio de transmisión de paquetes de *advertising* hacia los *smartphones* que hacen las veces de observadores.

Una vez activado el *Bluetooth*, la aplicación escanea en búsqueda del *beacon* más cercano (1 metro de distancia) para enviar un mensaje de bienvenida al *smartphone* y mostrar el contenido obtenido del portal web de la Universidad referente a la carrera asociada al *beacon*, permitiendo al usuario acceder a esta información. Finalmente, cuando el dispositivo se aleja del rango de cobertura del *beacon* recibe un mensaje de despedida.

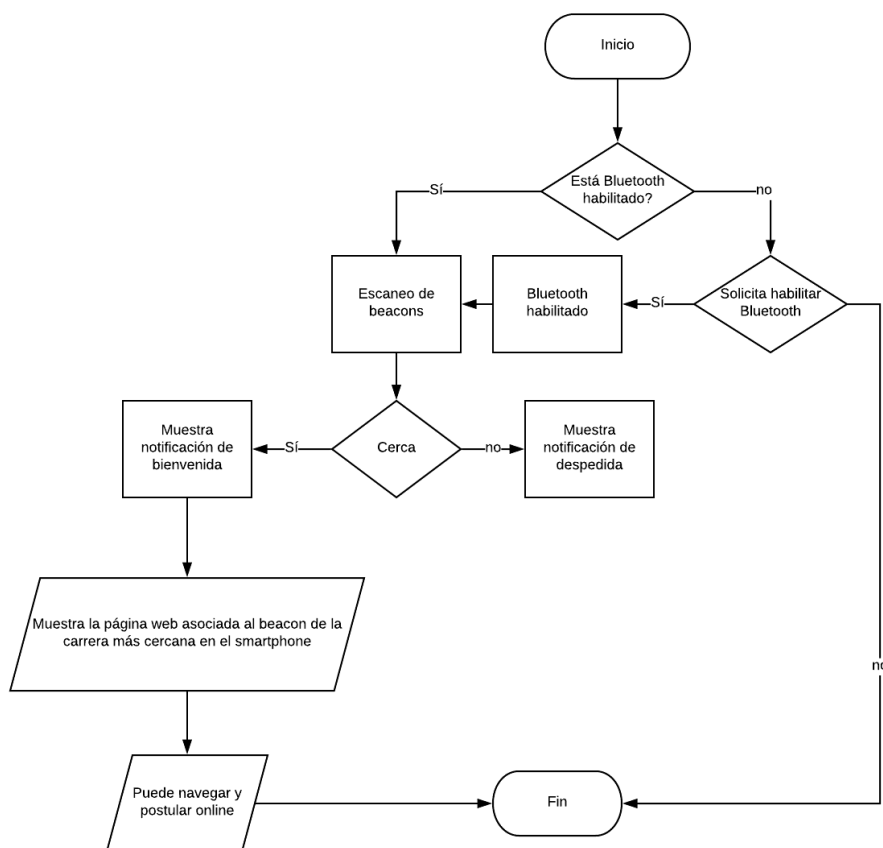


Figura 27. Diagrama de flujo de la aplicación móvil.

4.2. Android Studio

“*Android Studio* es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para *Android* y se basa en IntelliJ IDEA” (Android Developers, s.f.). Es un potente editor de códigos cuyo lenguaje de programación nativo es Java y desde la versión 3.0 de *Android Studio* viene integrado con el lenguaje de programación Kotlin, que es interoperable al 100% con Java (Rodríguez, 2017). Además cuenta con funciones adicionales que aumentan la productividad al compilar apps para *Android*, como las siguientes:

- Sistema de compilación basado en *Gradle* flexible.
- Emulador de dispositivos con funciones variadas.
- Entorno unificado para el desarrollo para todos los dispositivos *Android*.
- *Instant Run*, permite aplicar cambios al código mientras se ejecuta la aplicación sin necesidad de compilar de nuevo el APK.

- Compatibilidad con C++ y NDK.
- Soporte incorporado para *Google Cloud Platform*, facilitando la integración de *Google Firebase Cloud Messaging* y *App Engine*.

4.2.1. Estructura de los proyectos en Android Studio

Los proyectos de *Android Studio* contienen uno o más módulos con archivos de código fuente, así como archivos de recursos. Como se indica en la figura 28, *Android Studio* muestra de forma predeterminada los archivos del proyecto en la vista de proyectos de *Android*, esta vista se organiza en módulos para permitir un acceso rápido a los archivos de origen del proyecto (Android Developers, s.f.).

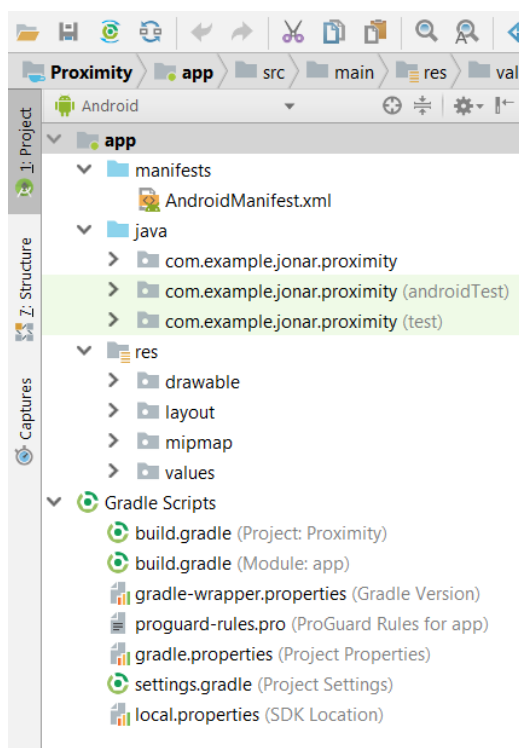


Figura 28. Archivos del proyecto en la vista de proyectos de *Android*.

Los *Gradle Scripts* contienen archivos de compilación, en los que se detalla de forma explícita las configuraciones de versiones mínimas de sistema operativo para ejecutar la aplicación y dependencias para integrar kits de desarrollo de terceros (SDK). Además, cada módulo de la aplicación contiene las siguientes carpetas:

- **Manifests:** contiene el archivo `AndroidManifest.xml`.
- **Java:** contiene los archivos de código fuente de la aplicación.
- **Res:** contiene todos los recursos que se utilizan en el proyecto, como diseños XML, estilos e imágenes *bitmap*.

4.2.2. Estimote Proximity SDK para Android

Estimote posee su propio kit para desarrollo de software (SDK), que permite a los desarrolladores implementar sus propios proyectos. El SDK funciona desde *Android* 4.3 en adelante y requiere que los dispositivos soporten *Bluetooth Low Energy*.

Agregar el SDK para eventos de proximidad al proyecto de *Android* es sencillo. Es necesario agregar las siguientes dependencias en la sección de *Gradle Scripts* `“build.gradle(Module:app)”`: `“com.estimote:proximity-sdk:0.4.4”`. Además de una librería de apoyo que será utilizada para permisos que requieran ubicación y acceso a *Bluetooth*: `“com.estimote:mustard:0.2.1”`.

Para realizar escaneo *Bluetooth*, *Android* requiere que la aplicación obtenga los siguientes permisos explícitos: `“ACCESS_COARSE_LOCATION”` o `“ACCESS_FINE_LOCATION”` en tiempo de ejecución. Para facilitar la tarea de solicitar estos permisos por parte del usuario se usará la librería `“com.estimote:mustard”`, que permite con unas líneas de código adicionales simplificar, verificar y solicitar los permisos necesarios de acceso y disponibilidad de *Bluetooth*.

4.3. Aplicación en Android

Una vez incluidas las dependencias para el SDK en nuestro proyecto, Estimote sugiere agregar credenciales de *Estimote Cloud* que ayudarán a recolectar información analítica como: el número de visitantes, los beacons más visitados y el tiempo que los visitantes permanecieron frente a un *beacon*. Todos estos datos estadísticos pueden ser visualizados en el portal *web*.

Para agregar las credenciales es necesario obtenerlas desde el portal de *Estimote Cloud*, y en la sección Apps seleccionar `“Your Own App”`, posteriormente se le asigna un nombre que identifique a la aplicación. Las

credenciales para su integración se generan automáticamente. Como muestra la figura 29, obtendremos un App ID y un App *Token* que se incluirá dentro del método “*onCreate*” de la actividad del proyecto que maneja el escaneo de beacons con la siguiente instrucción: “*cloudCredentials (“app-id”, “token”)*”

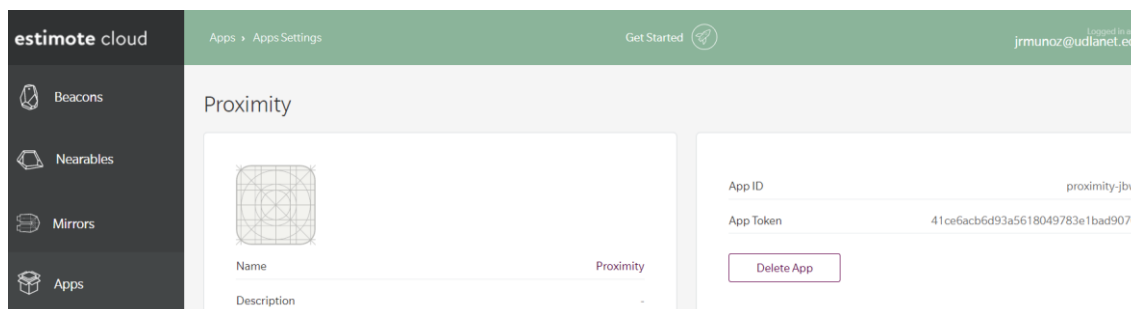


Figura 29. Visualización de las credenciales obtenidas del portal *Estimote Cloud*.

Una vez incluidas las credenciales, se crea un objeto “*ProximityObserver*” para el manejo de eventos. Se definen 3 zonas de proximidad (Telecomunicaciones, Software y TI), utilizando el método “*ProximityZone*”. Para facilitar la tarea de identificación de los *beacons* (diferente al ID de 32 dígitos con el que vienen desde fábrica), se adjuntará un identificador compuesto por un par clave-valor a cada *beacon* desde el portal de *Estimote Cloud*. Como se muestra en la figura 30, la clave para asignar a todos los beacons será denominada “*owner*”, el valor será asignado en base a la carrera propietaria de cada *beacon*; por ejemplo: el valor del *beacon* de la carrera Software será “*soft*”, de Telecomunicaciones, “*tele*” y de Tecnologías de Información, “*ti*”.

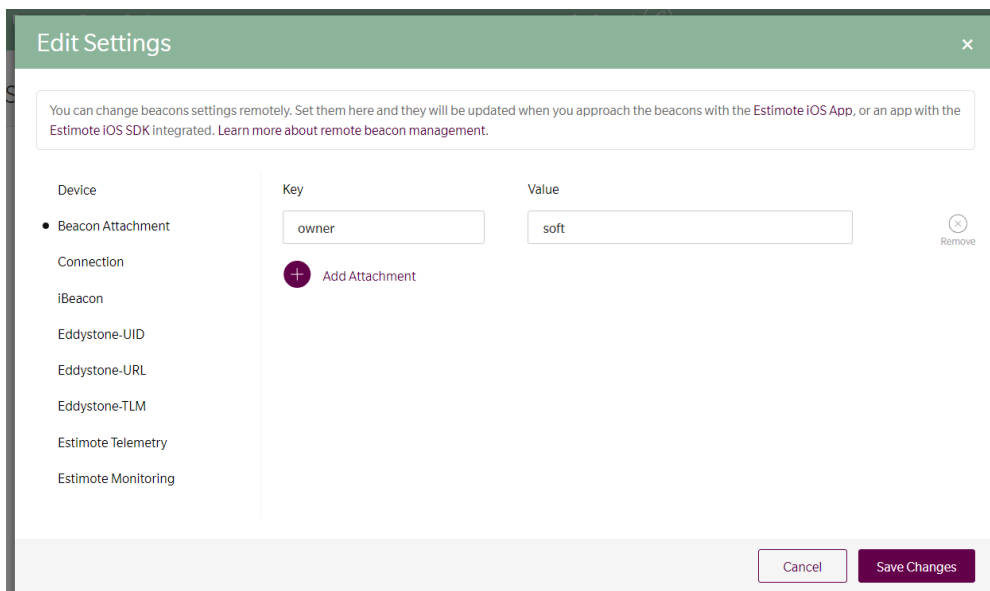


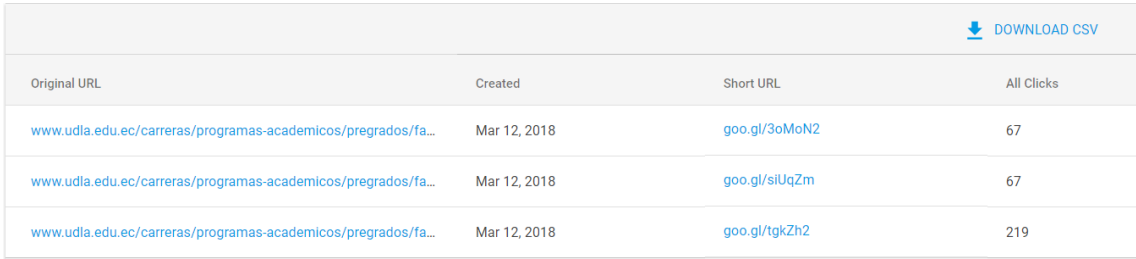
Figura 30. Asignando identificadores adjuntos a un beacon desde el portal Estimote Cloud.

De esa forma en el código se puede utilizar el método “*forAttachmentKeyAndValue*” y así indicar a que *beacon* programar determinado comportamiento al entrar, con el método “*withOnEnterAction*”, o al salir “*withOnExitAction*” del rango de alcance del *beacon*. Para los rangos de alcance existen tres métodos: Lejos (5 metros) “*inFarRange*”, cerca (1 metro) “*inNearRange*” y un método para un rango personalizado “*inCustomRange*”.

Debido a que los *beacons* solo pueden soportar un tipo de paquete a la vez (iBeacon, Eddystone o *Estimote Monitoring*), y considerando que la aplicación va a ser programada tanto en *Android* como en *iOS*, se utilizará el paquete *Estimote Monitoring*, ya que funciona en ambos sistemas operativos y permite cumplir con la meta de este proyecto que es mostrar los portales *web* de las carreras de Telecomunicaciones, Software y Tecnologías de Información de forma dinámica a medida que el *smartphone* se posicione en la zona de cobertura de un *beacon* específico.

La función principal de la aplicación es mostrar directamente la página *web* de la carrera asociada al beacon más cercano, por lo que es necesario incluir en la interfaz gráfica un *WebView* y declararlo posteriormente dentro del método “*withOnEnterAction*” de cada zona creada previamente. Como se puede observar en la figura 31, los 3 URL de las carreras antes mencionadas serán

acortados utilizando URL *Shortener* de Google y declararlas individualmente como una variable global tipo *String* que posteriormente serán utilizadas dentro del método del *WebView* “*loadUrl*”.



| Original URL | Created | Short URL | All Clicks |
|--|--------------|--|------------|
| www.udla.edu.ec/carreras/programas-academicos/pregrados/fa... | Mar 12, 2018 | goo.gl/3oMoN2 | 67 |
| www.udla.edu.ec/carreras/programas-academicos/pregrados/fa... | Mar 12, 2018 | goo.gl/sIUqZm | 67 |
| www.udla.edu.ec/carreras/programas-academicos/pregrados/fa... | Mar 12, 2018 | goo.gl/tgkZh2 | 219 |

Figura 31. Visualización de URL acortados mediante la página Google URL *Shortener*.

4.3.1. Diseño de la interfaz gráfica en Android

Android Studio permite escoger entre varios tipos de actividades prediseñadas para darle forma a la interfaz gráfica o en su defecto puede ser declarada como actividad vacía y personalizar su contenido a conveniencia. El diseño depende del enfoque que vaya a tener la aplicación, ya que cada plantilla incluye características específicas para casos de uso donde se necesite: interactuar con mapas, actividades que necesitan utilizar toda la pantalla, inicio de sesión, navegación por pestañas o botones, entre otras; como se muestra en la figura 32.

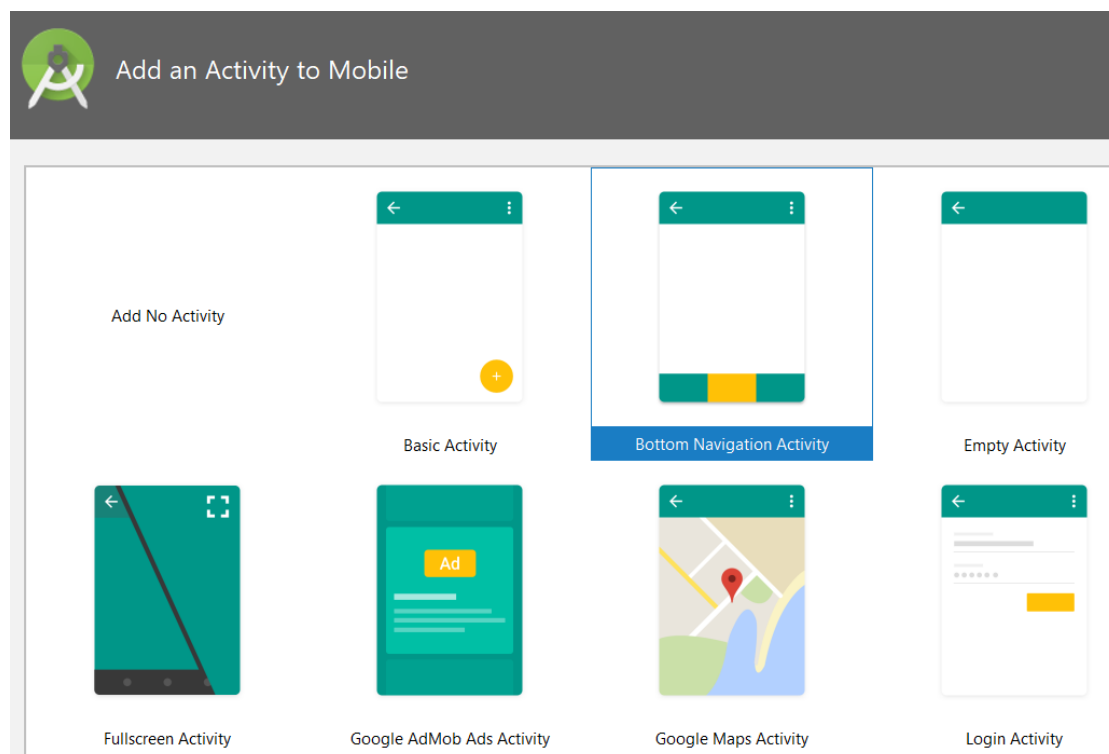


Figura 32. Actividades prediseñadas disponibles al crear una nueva aplicación en *Android Studio*.

De acuerdo con la especificación técnica provista en el portal para desarrolladores de Google, se puede utilizar un *bottom navigation activity* cuando se necesite de 2 hasta 5 actividades en la parte inferior de la aplicación. En caso de que sean necesarias más de 5 actividades se recomienda utilizar una interfaz gráfica tipo *tab navigation* que está compuesta por pestañas en la parte superior de la barra de navegación.

Para la aplicación se utilizará una interfaz *bottom navigation activity* que permita la navegación por botones ubicados en la parte inferior del entorno de la actividad y muestre el contenido en base a la selección de los ítems que conforman la barra de navegación. La barra de navegación estará conformada por dos actividades: la primera actividad está orientada a la visualización del contenido de la sección de noticias obtenido del portal web de la Universidad en un *WebView*. En la figura 33 se puede observar el plano de pantalla de esta actividad.

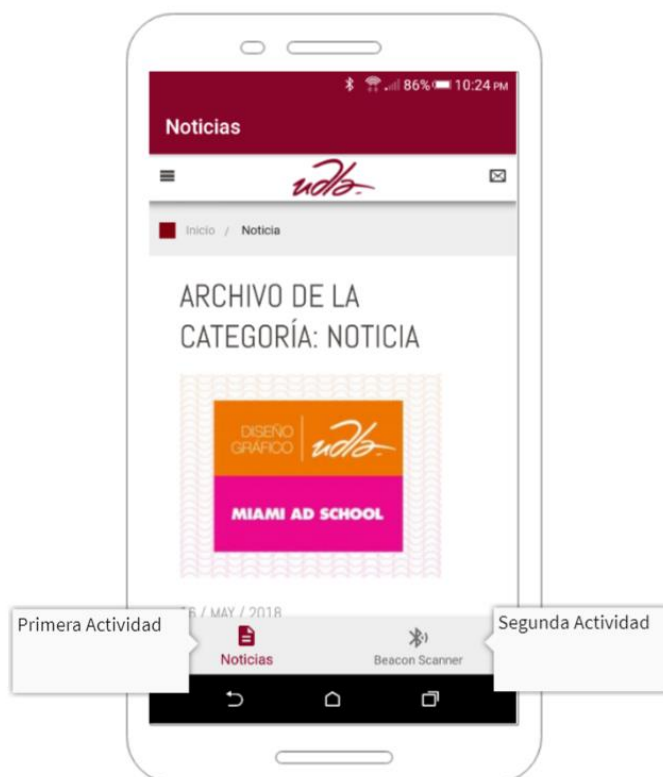


Figura 33. Plano de pantalla de la actividad “Noticias”.

La segunda actividad contiene un botón con la leyenda “Escanear Beacons”, que ejecuta el proceso de solicitud de permisos de acceso a *Bluetooth* y acceso a servicios de localización para establecer con mayor exactitud la proximidad del *smartphone* con respecto a los beacons. Una vez realizadas estas validaciones muestra una nueva actividad (*WebActivity.java*) que contiene un *WebView* y reacciona en base a la proximidad de los beacons mostrando la información de cada carrera obtenida de la página web de la Universidad. En la figura 34 se muestra el plano de pantalla del proceso de navegación antes descrito

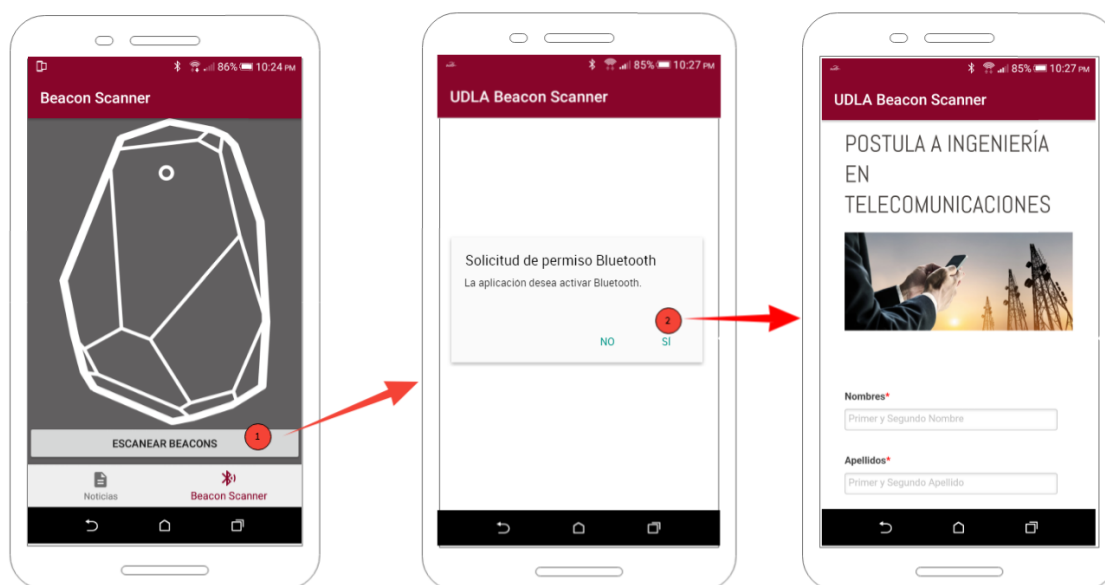


Figura 34. Plano de pantalla y navegación de la actividad “Beacon Scanner”

Cuando se utiliza una interfaz gráfica tipo *bottom navigation activity* se deben declarar las actividades que componen la barra de navegación inferior como *fragment activity*, esto permite que *Android Studio* las diferencie de una *activity* tradicional y puedan ser especificadas en el *MainActivity* para reconocerlas como un elemento más de la barra de navegación.

4.3.2. Notificaciones locales en Android

La aplicación está en capacidad de generar notificaciones locales cuando el *smartphone* se encuentra en el rango de alcance de un *beacon*, con el objetivo de informar al usuario de la presencia de estos dispositivos en el entorno y para permitir que accedan al contenido informativo de la carrera directamente. Para programar este comportamiento se hará uso de los métodos “*withOnEnterAction*” y “*withOnExitAction*” descritos anteriormente en este capítulo durante la especificación de las zonas de proximidad de cada beacon.

Para utilizar la función de notificaciones locales es necesario importar las librerías *NotificationCompat* y *NotificationManager*, posteriormente se debe declarar el objeto *NotificationCompat* como variable global, que permita generar las notificaciones en cada una de las zonas de proximidad y no necesite ser declarada varias veces.

Cuando el *smartphone* ingrese en el rango de cobertura de un *beacon* la aplicación usará el método “*withOnEnterAction*” y generará automáticamente una notificación de bienvenida acompañada del nombre de la carrera asociada al *beacon* más próximo, permitiendo que cuando el usuario seleccione la notificación cargue la información de la carrera para mostrarla en un *WebView*, caso contrario, cuando el *smartphone* se aleje de la zona de proximidad, la aplicación reconocerá este comportamiento y entrará al método “*withOnExitAction*” y generará una notificación de despedida. En la figura 35 se muestra una captura de pantalla de este comportamiento, cuando el *smartphone* entra y sale de la zona de proximidad del beacon asociado a la carrera de Ingeniería en Telecomunicaciones.

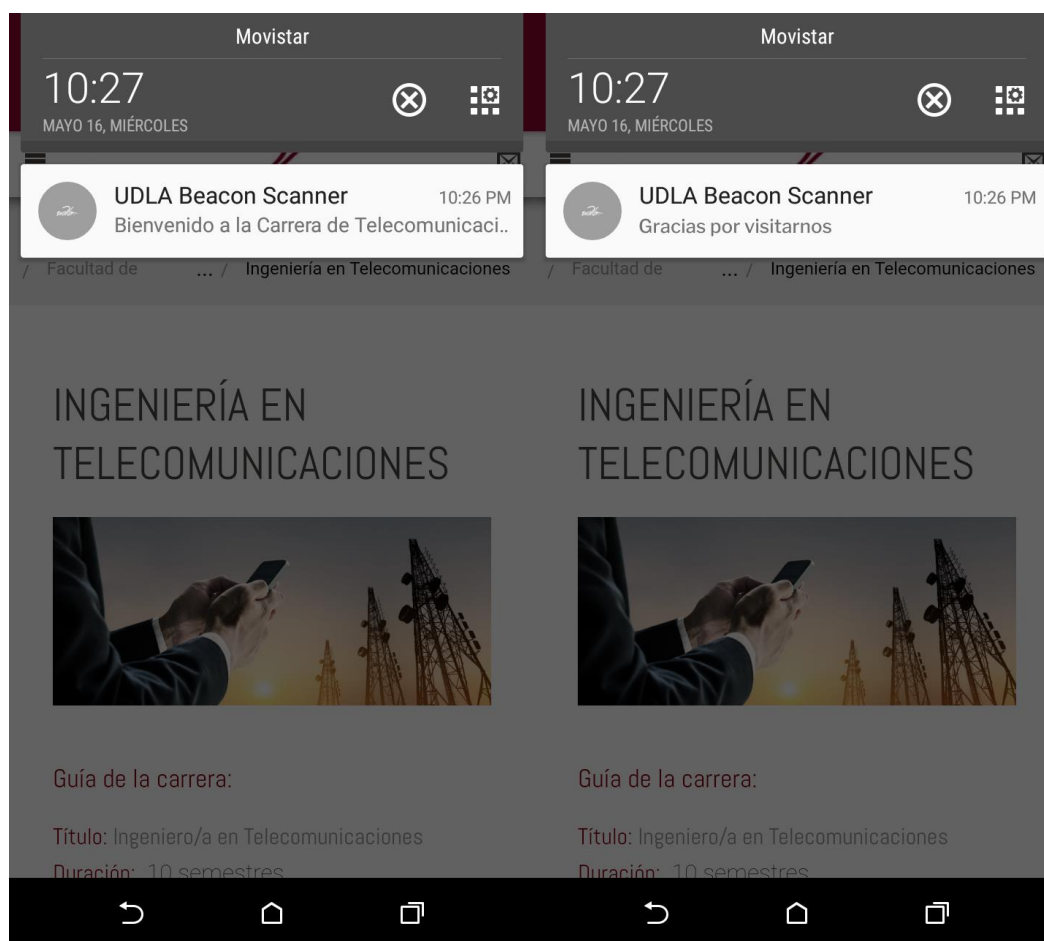


Figura 35. Captura de pantalla con las notificaciones de proximidad recibidas en la bandeja de notificaciones del sistema de *Android* al entrar y salir del rango de alcance de un beacon.

En la figura 36 se muestra un diagrama con el resumen de métodos, clases y *frameworks* utilizados para el desarrollo de la aplicación para *Android*.

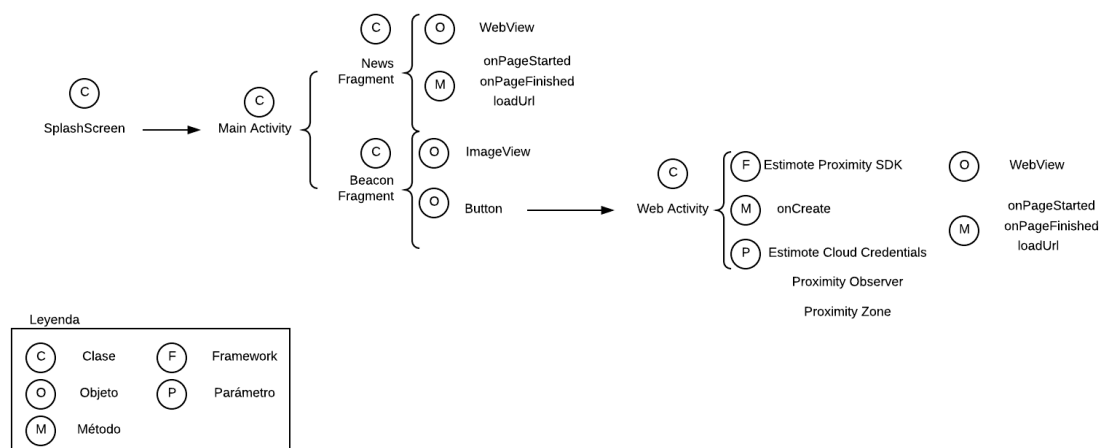


Figura 36. Diagrama de distribución de métodos, clases y *frameworks* para el desarrollo de la aplicación en Android.

Finalmente es necesario importar las librerías *Intent* y *PendingIntent* y declarar en su constructor el contexto de la actividad de origen y destino para que al momento de seleccionar la notificación nos lleve a la actividad que contiene la información de la carrera.

4.4. Xcode

Xcode es el entorno de desarrollo integrado (IDE) de Apple para programar software para cualquier plataforma de Apple (MacOS, iOS, watchOS y tvOS). Utiliza el formato de archivo *.xcodeproj* para administrar todos los archivos de código, así como archivos de configuración creados o importados. También cuenta con un editor de código fuente optimizado para los lenguajes de programación *Objective-C* y *Swift*, incluyendo archivos con ajustes preestablecidos para el desarrollo de aplicaciones para las distintas plataformas de Apple (Tiano, 2016).

4.4.1. Estructura de los proyectos en Xcode

Como se puede observar en la figura 37 los proyectos de *Xcode* contienen varios ficheros, entre ellos archivos de código fuente, así como de recursos.

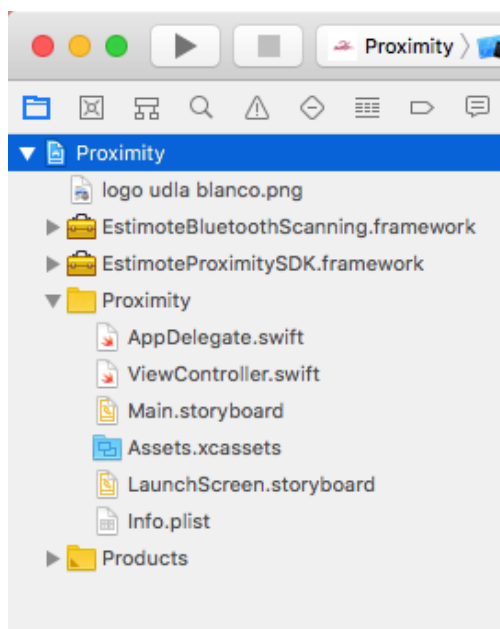


Figura 37. Archivos que conforman la estructura de un proyecto en Xcode.

Los archivos que conforman un proyecto por defecto son los siguientes:

- **AppDelegate:** se encarga del comportamiento básico de la aplicación, en este archivo se encuentran las funciones que se ejecutan cuando arranca la aplicación, cuando entra en segundo plano o vuelve de este estado.
- **Main.storyboard:** aquí se encuentran todos los archivos para gestionar y acceder a las herramientas necesarias para construir o editar la interfaz gráfica de la aplicación. El storyboard permite crear la estructura y la secuencia de navegación en la aplicación.
- **ViewController:** este archivo está directamente relacionado con el *storyboard*, aquí se declaran los elementos colocados en la interfaz gráfica para programar su comportamiento.
- **Assets.xcassets:** incluye los archivos gráficos utilizados para generar los íconos de la aplicación en sus diferentes resoluciones.
- **Launchscreen.storyboard:** en este archivo se diseña la pantalla de bienvenida cuando se ejecuta la aplicación.
- **Info.plist:** en este archivo se definen ciertos comportamientos básicos de la aplicación, como tipos de letra, orientación del dispositivo, nombre de la aplicación, permisos para acceder a recursos del *smartphone*, etc.

4.4.2. Estimote Proximity SDK para iOS

El kit de desarrollo de software (SDK) puede integrarse de dos maneras al proyecto de *Xcode*:

- Mediante *CocoaPods* utilizando el terminal de comandos, o.
- Manualmente descargando del repositorio los *framework* de *EstimoteProximitySDK* y *EstimoteBluetoothScanning* e incluyéndolos en los binarios embebidos del proyecto como se muestra en la figura 38.

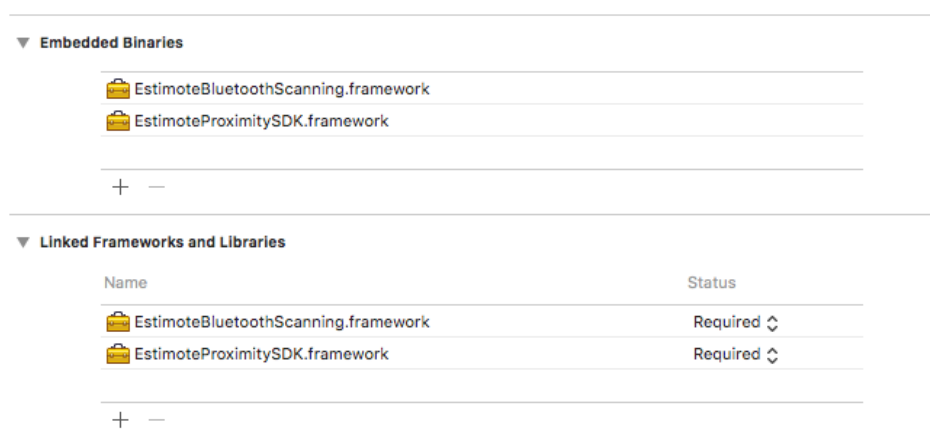


Figura 38. Visualización de *frameworks* de *Estimote Proximity SDK* en la interfaz de *Xcode*.

Las librerías son compatibles tanto con *Objective-C* como con *Swift*. Las clases públicas están escritas en *Objective-C*, mientras que la interfaz de programación de aplicaciones (API), está optimizada para *Swift* (Estimote, s.f.). El siguiente paso es importar el SDK recientemente agregado al archivo controlador de la vista con el comando.

4.5. Aplicación en iOS

Como se describió anteriormente en el capítulo 3, iBeacon es parte del servicio *Core Location* de *iOS*; por lo que es necesario preguntar explícitamente al usuario por su permiso para acceder a sus datos de localización. Esto puede ser realizado al agregar las siguientes líneas de tipo *string* en el archivo *Info.plist* para establecer la descripción de uso de los servicios de ubicación:

- ***NSLocationWhenInUseUsageDescription***: describe como la aplicación utiliza los servicios de localización cuando está en uso o en segundo

plano (Estimote, s.f.).

- ***NSLocationAlwaysUsageDescription***: describe como la aplicación utiliza los servicios de localización en ambas circunstancias, cuando está en uso y cuando está en segundo plano. Esta descripción aplica solamente a usuarios que utilizan la aplicación con *iOS* 10 o posterior (Estimote, s.f.).
- ***NSLocationAlwaysAndWhenInUsageDescription***: esta descripción se utiliza en circunstancias similares, solo que orientada a aplicaciones ejecutándose *iOS* 11.

Una vez incluidas estas líneas, lo que provocarán en la aplicación cuando se ejecute por primera vez será preguntarle al usuario si desea darle permiso a la aplicación para acceder a sus datos de localización con 3 opciones: siempre, cuando la aplicación esté en uso o nunca.

Para permitir que la aplicación se ejecute en segundo plano cuando se encuentre en el rango de alcance de un *beacon*, es necesario habilitar la opción *Background Mode* de la pestaña *Capabilities* en el proyecto de *Xcode*. En la figura 39 se muestra la lista de opciones disponibles dentro de esta sección, de la cual habilitaremos solamente la opción “*Uses Bluetooth LE accessories*”.

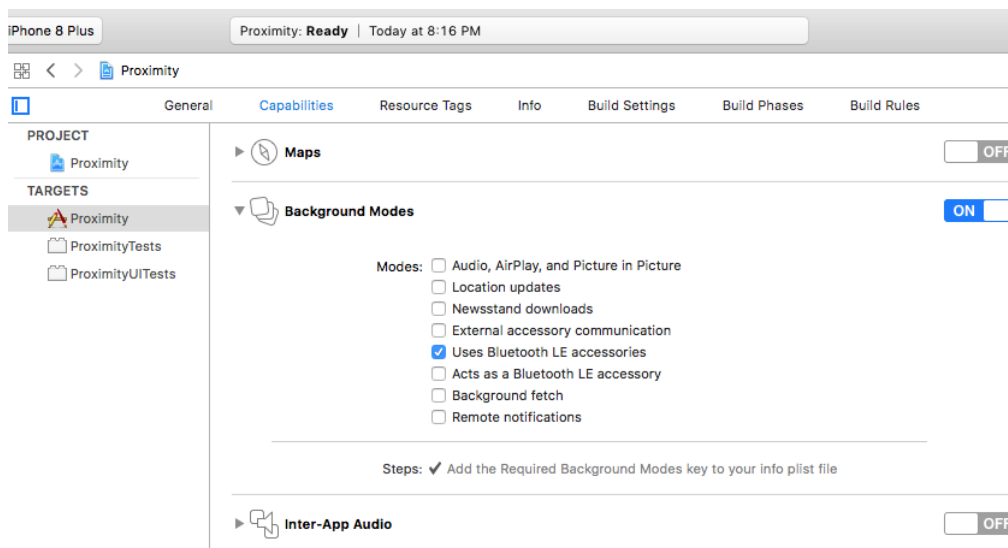


Figura 39. Habilitación de opción Bluetooth para modo de ejecución en segundo plano.

El siguiente paso es establecer las credenciales de *Estimote Cloud*, que son generadas desde el portal *web* de Estimote. Como se mencionó en la descripción de la aplicación en *Android*, el proceso es similar, se obtendrá un ID y un *Token* que debe ser incluido en el proyecto con el comando “`cloudCredentials = EPXCloudCredentials(appID:,appToken:)`” dentro del método “`viewDidLoad`” del archivo controlador de la vista.

Una vez establecidas las credenciales, creamos el *Proximity Observer* declarándolo como una variable global. Mantenemos los identificadores de tipo llave y valor, que fueron asignados a los *beacons* para diferenciarlos de cada carrera en la aplicación de *Android*. En base a esto creamos las variables para crear las zonas de proximidad a los *beacons* con el siguiente comando: “`let zone1 = EPXProximityZone`” y a continuación se especifica el rango, que puede ser: “`.far`” (5 metros), “`.near`” (1 metro) o “`.desiredMeanTriggerDistance`” para distancias personalizadas. Aquí también se asociará la llave y valor que se asignó a los *beacons* desde el portal de *Estimote Cloud* y se lo especifica en: “`attachmentKey`” y “`attachmentValue`”.

Para mostrar el portal *web* de cada carrera de forma dinámica se incluirá un *WebView* dentro de los tres métodos definidos para las zonas de proximidad de los *beacons* asociados a cada carrera, tomando en cuenta que antes deben ser

declaradas las URL acortadas correspondientes.

La documentación obtenida del portal de desarrolladores de Apple sugiere que para implementar una aplicación que utilice vistas web en versiones de iOS 8 y superior, es necesario utilizar un objeto *WKWebView* que es parte del *framework WebKit*, que incluye varias mejoras con relación a la capacidad de ejecutar *javascript* y velocidad al mostrar el contenido en comparación al anterior *WebView*.

Finalmente, para comenzar con el escaneo de *beacons* utilizaremos el comando “*self.proximityObserver.startObserving*” y se incluirán las 3 zonas de proximidad definidas anteriormente.

4.5.1. Diseño de la interfaz gráfica en iOS

Xcode posee un editor de interfaz incorporado en el IDE, que hace más simple diseñar completamente la interfaz de usuario sin necesidad de escribir código. Con tan solo arrastrar y soltar objetos de la biblioteca de recursos como: ventanas, botones, campos de texto, entre otros en el lienzo de diseño es posible crear una interfaz de usuario funcional (Apple, s.f.).

Debido a que *Cocoa* y *CocoaTouch* están construidas utilizando el patrón Modelo-Vista-Controlador (MVC), resulta más sencillo diseñar las interfaces independientemente de su implementación. De hecho, las interfaces de usuario están archivadas como objetos *Cocoa* o *CocoaTouch* (almacenados como archivos con extensión *nib*), *iOS* dinámicamente creará la conexión entre la interfaz de usuario y el código una vez que la aplicación se ejecute (Apple, s.f.).

Una aplicación está compuesta por múltiples vistas por las que el usuario navega. Las relaciones entre estas vistas se componen de un guion gráfico o *Storyboard*, que se encarga totalmente del diseño de la interfaz gráfica, la navegación y del flujo de las vistas que tendrá la aplicación, para posteriormente agregar el código necesario que controle el comportamiento personalizado que tendrá la aplicación.

Para la aplicación se utilizará una interfaz *Tab Bar Controller* que contendrá dos vistas: una vista denominada Noticias, que mostrará en un *WebView* la sección de noticias obtenidas del portal de la Universidad, como se muestra en la figura 40.

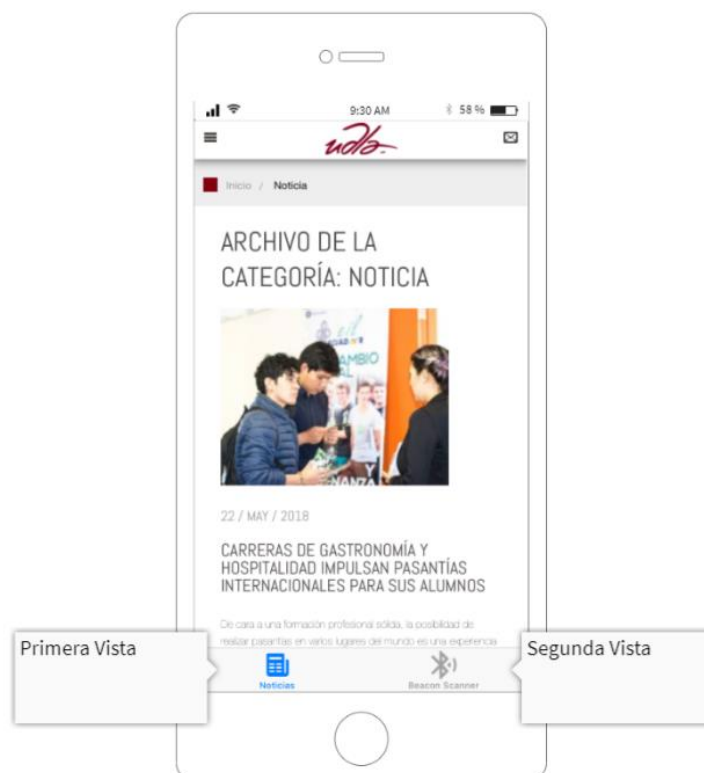


Figura 40. Plano de pantalla de la vista “Noticias”

La segunda vista denominada *Beacon Scanner* contendrá un botón con la leyenda “Escanear Beacons” que automáticamente verifica si se encuentra habilitado *Bluetooth*, caso contrario muestra una alerta para dirigir al usuario a las configuraciones de su dispositivo para habilitar *Bluetooth*. Una vez realizadas estas validaciones muestra una nueva vista con un *WebView* y reacciona en base a la proximidad de los *beacons* mostrando la información de cada carrera obtenida de la página web de la Universidad. En la figura 41 se muestra el flujo de la aplicación al presionar el botón en búsqueda de *beacons*.

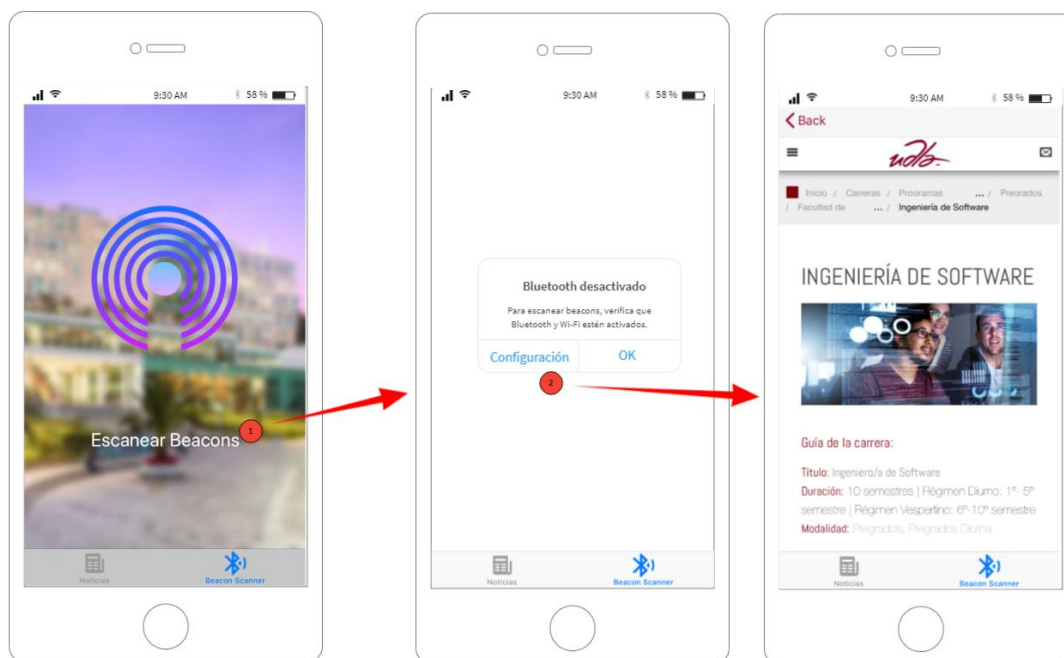


Figura 41. Plano de pantalla y navegación de la vista “Beacon Scanner”.

Para programar este comportamiento se utilizará el archivo controlador de esta vista: `WebViewController.swift`, en el han sido definidas previamente las zonas de proximidad e inicializado el *Proximity Observer*, proceso descrito anteriormente en este capítulo. Se hará uso del método “*onEnterAction*” de cada zona de proximidad, declarando como parámetro al objeto *webView* que cargará el contenido *web* con el método “*load(URLRequest: url)*”.

Con el fin de hacerle saber al usuario de forma gráfica que existe una tarea en proceso, en este caso cargar el contenido *web* de la carrera, se agregará un indicador de actividad (representado por una rueda de proceso), mientras se ejecuta esta tarea. Para lograr esto utilizaremos el delegado de navegación (*WKNavigationDelegate*) de *WKWebView*, y definiremos dos funciones: una cuando comience la carga (*didCommit*) del contenido *web* y otra cuando finalice (*didFinish*), llamando dentro de cada función al objeto indicador de actividad con sus métodos *startAnimating* y *stopAnimating* respectivamente.

4.5.2. Notificaciones locales en iOS

La aplicación también puede generar notificaciones locales, sobre todo cuando el *smartphone* se encuentra en el rango de alcance de un *beacon*. Estas notificaciones tienen el objetivo de informar al usuario sobre la presencia de

estos dispositivos en el entorno, de manera que una vez presionadas las notificaciones sea posible acceder directamente a la aplicación. Para programar este comportamiento será necesario importar el *framework UserNotifications* al archivo *AppDelegate.swift*, este archivo se encarga de manejar todos los eventos que se producen fuera de los controladores de cada vista. Dentro de la función *“didFinishLaunchingWithOptions”*, que se ejecuta cuando la aplicación es abierta por primera vez, se declara el método para solicitar los permisos que permitan a la aplicación recibir notificaciones.

En el controlador *WebViewController.swift* también se hará uso de los métodos *“onEnterAction”* y *“onExitAction”* de las zonas de proximidad de cada *beacon* para enviar notificaciones de bienvenida o despedida respectivamente. Dentro de estos métodos se declararán los parámetros para construir el contenido de la notificación utilizando el método *“UNMutableNotificationContent”* para posteriormente designar el título, cuerpo e insignia de la notificación. La figura 42 muestra una captura de pantalla obtenida al momento de recibir una notificación cuando el smartphone entra y sale del rango de alcance de un *beacon*.

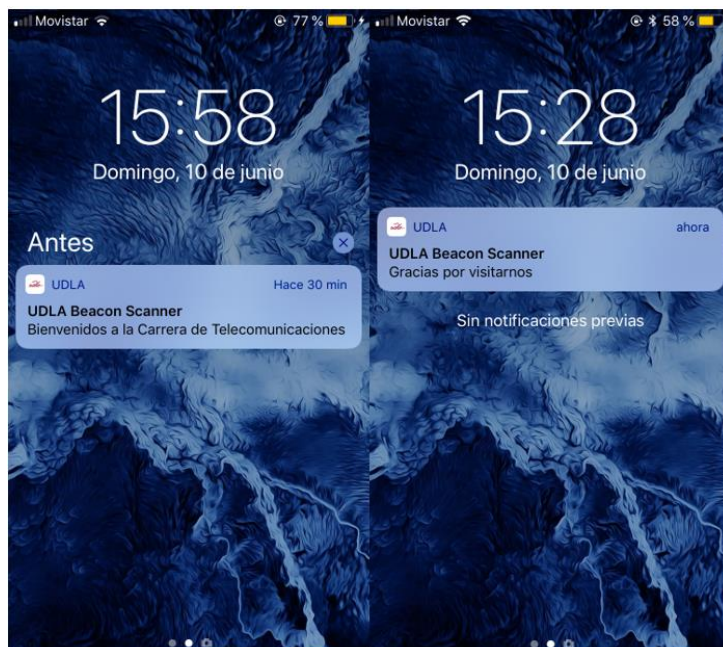


Figura 42. Captura de pantalla con la notificación de proximidad recibida en la bandeja de notificaciones del sistema iOS al entrar y salir del rango de alcance del beacon.

En la figura 43 se muestra un diagrama con el resumen de métodos, clases y *frameworks* utilizados para el desarrollo de la aplicación para iOS.

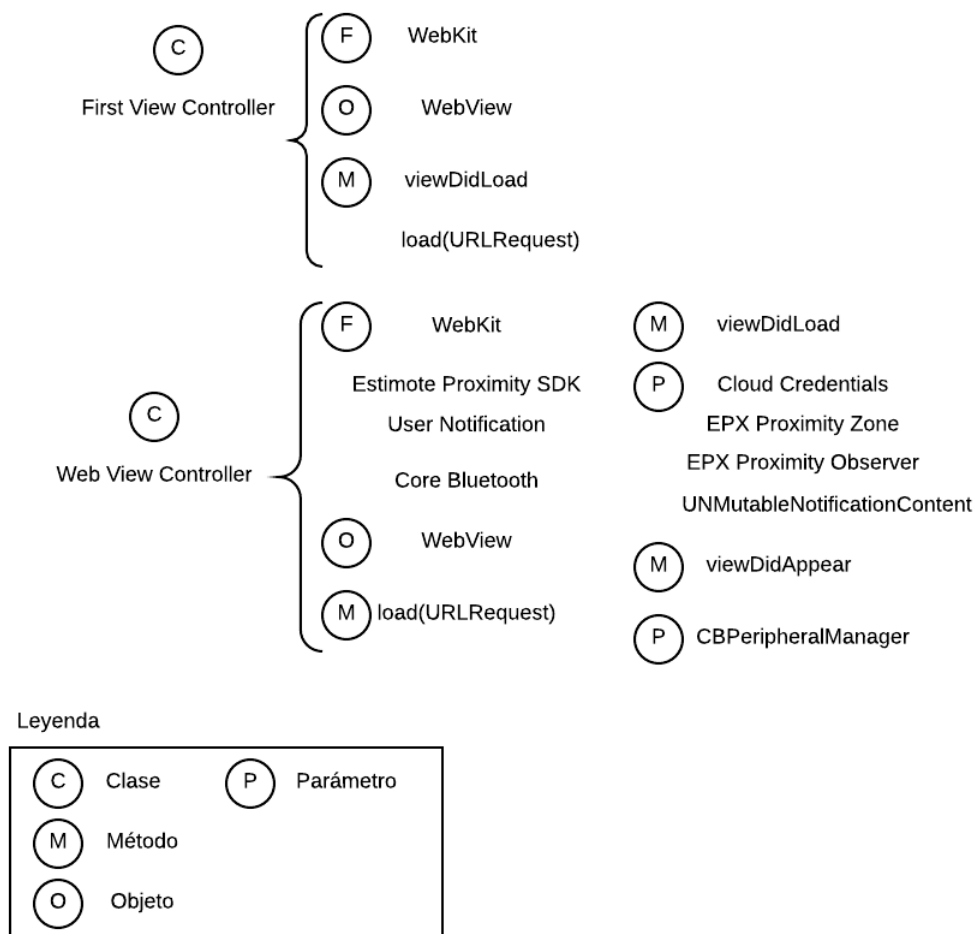


Figura 43. Diagrama de distribución de métodos, clases y *frameworks* para el desarrollo de la aplicación en iOS.

5. INTEGRACIÓN DE LA APLICACIÓN CON GOOGLE FIREBASE CLOUD MESSAGING

5.1. Google Firebase Cloud Messaging

De acuerdo con la definición del autor “*Firebase Cloud Messaging (FCM)* es una solución de mensajería multiplataforma que permite enviar mensajes de forma segura y gratuita” (Google, 2018). Esta herramienta hace posible enviar mensajes de notificación o notificaciones *push* desde una aplicación servidor hacia una aplicación cliente sin necesidad de que exista una petición previa,

con el objetivo de informar a sus usuarios sobre temas como:

- Comunicar sobre eventos.
- Recordatorios de servicios.
- Promocionar productos o servicios.
- Envío de alertas informativas basadas en la ubicación del usuario.
- Chat entre usuarios.

FCM permite distribuir mensajes a las aplicaciones cliente de tres formas:

- Dispositivos individuales.
- Dispositivos suscritos a temas.
- Grupos de dispositivos.

Estas opciones pueden ser seleccionadas al componer mensajes desde la consola de *Firebase* y brinda la posibilidad de segmentar las notificaciones para dispositivos específicos.

5.1.1. Tipos de mensajes

FCM soporta dos tipos de mensaje que pueden ser enviados a los aplicativos clientes:

- Mensajes de notificación: denominados también mensajes de pantalla, manejados por el SDK de FCM de forma automática.
- Mensajes de datos: cuyo comportamiento es manejado en la aplicación cliente.

La estructura de los mensajes de notificación contiene un conjunto preestablecido de claves (título y contenido del mensaje), mientras que los mensajes de datos contienen pares clave-valor que son definidos por el usuario. Sin embargo, los mensajes de notificación pueden contener adicionalmente una carga de datos opcional, el peso máximo de ambos mensajes es de 4 KB, a excepción de cuando son enviados desde la consola de *Firebase*, con un límite de 1024 caracteres (Google, 2018).

El comportamiento de la aplicación al recibir mensajes que incluyen cargas tanto de notificación como de datos dependerá del estado en el que se encuentre la aplicación, estos son:

- Primer plano. Cuando la aplicación está activa, recibe el mensaje con ambas cargas disponibles.
- En segundo plano. La aplicación recibe la carga de notificaciones en la bandeja de notificaciones del sistema y se encarga del manejo de datos cuando el usuario presiona la notificación.

5.1.2. Arquitectura de FCM

La arquitectura de FCM está conformada por los siguientes componentes para enviar y recibir datos:

- Entorno de confianza o un servidor de aplicaciones para generar, orientar y enviar mensajes.
- Servidor de FCM.
- Aplicación cliente que reciba los mensajes (Android, iOS o Web).

Como se muestra en la figura 44 mediante la consola gráfica para composición de mensajes de *Firebase* o mediante una aplicación servidor de terceros se realiza la petición al servidor de FCM y esta a su vez se encarga de entregar la notificación a la aplicación cliente (Google, 2018).

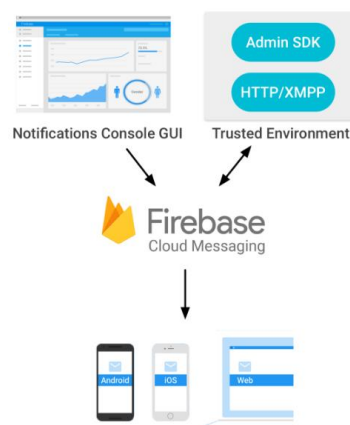


Figura 44. Flujo de componentes que influyen en el proceso de envío de mensajes mediante FCM.
Tomado de Google, 2018.

FCM utiliza el puerto 5228 y en ciertas ocasiones 5229 y 5230 para conexiones entrantes. Para conexiones salientes, FCM no proporciona una IP específica, ya que el rango de IP cambia constantemente (Google, 2018).

5.2. Envío de mensajes con Firebase Console

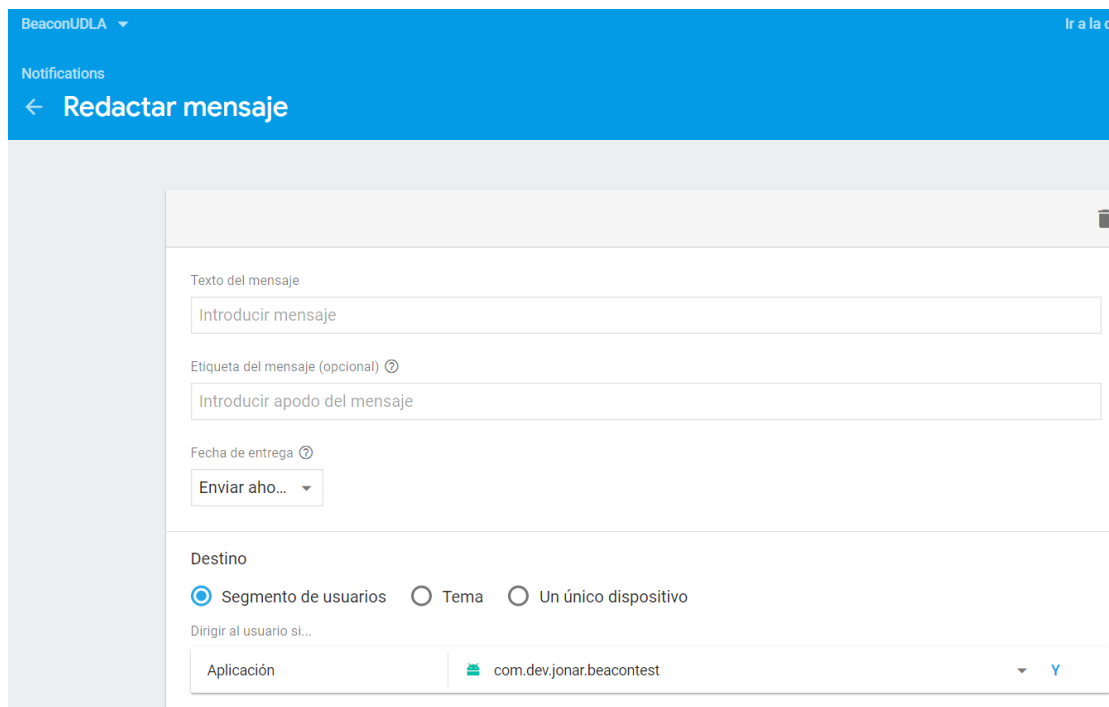
Enviar mensajes desde la consola de composición de notificaciones de *Firebase* permite enviar mensajes hacia dispositivos con *iOS* y *Android*. Sin embargo, esta acción tiene ciertas limitaciones con relación al envío de mensajes utilizando protocolos HTTP y XMPP, pero resulta muy útil para realizar pruebas o *marketing* orientado a la participación de usuarios (Google, 2018).

Cuando se envía una notificación desde el compositor de notificaciones, los campos ingresados en el compositor se utilizan de dos maneras:

- Los campos segmento de usuario y vencimiento determinan las opciones para el destino y entrega del mensaje (Google, 2018).
- Los campos texto del mensaje y datos personalizados son enviados al cliente contenidos en una carga de pares clave-valor (Google, 2018).

En la figura 45 se muestra una captura de pantalla obtenida del compositor de notificaciones de *Firebase*, donde figuran los campos de texto del mensaje, fecha de entrega, donde se puede escoger entre el envío inmediato de la notificación o calendarizar la entrega del mensaje para una fecha y hora posterior. Además, se puede especificar el destino de la notificación con las siguientes opciones:

- Segmento de usuarios, seleccionando la aplicación o aplicaciones que estén vinculadas al proyecto de *Firebase*.
- Bajo suscripción a un tema.
- Un dispositivo individual, para lo cual es necesario conocer el *token* del dispositivo que ejecuta la aplicación cliente.



The screenshot shows the 'Redactar mensaje' (Compose message) interface in the Firebase Cloud Messaging console. The interface is in Spanish and includes the following fields and options:

- Texto del mensaje:** A text input field with the placeholder 'Introducir mensaje'.
- Etiqueta del mensaje (opcional):** A text input field with the placeholder 'Introducir apodo del mensaje'.
- Fecha de entrega:** A dropdown menu currently set to 'Enviar aho...'.
- Destino:** Three radio button options: 'Segmento de usuarios' (selected), 'Tema', and 'Un único dispositivo'.
- Dirigir al usuario si...:** A dropdown menu currently set to 'Aplicación' with the value 'com.dev.jonar.beacontest' displayed.

Figura 45. Campos disponibles para personalizar un mensaje desde la consola de FCM.

Las opciones avanzadas del compositor de notificaciones permiten llenar los campos con datos personalizados para enviar una carga de pares clave-valor, dando la apertura para agregar más información al contenido del mensaje que debe ser interpretado como extras en la aplicación. Además de esta opción, se puede establecer la prioridad de la notificación entre alta y media, habilitar o deshabilitar el sonido de la notificación (aplica solo para iOS) y la fecha de caducidad de la notificación que es el tiempo que debe guardarse para ser reenviado. En la figura 46 se muestra una captura de pantalla de lo antes descrito.

Figura 46. Campos opcionales para enviar datos personalizados clave-valor y opciones adicionales del compositor de notificaciones.

En la tabla 7 se especifican las claves que la consola de *Firestore* envía a los clientes y la relación que tienen con las etiquetas del campo de la consola.

Tabla 7.

Resumen de claves enviadas por la consola de Firestore y su descripción.

| Clave | Etiqueta del campo de la consola | Descripción |
|----------------------------------|----------------------------------|---|
| <i>notification.title</i> | Título del mensaje | Indica el título de la notificación. |
| <i>notification.body</i> | Texto del mensaje | Indica el texto del cuerpo de la notificación. |
| <i>data</i> | Datos personalizados | Pares clave-valor definidas por el compositor del mensaje. Se envían como carga útil de datos y son administrados por la aplicación |
| <i>priority</i> | Prioridad | Establece la prioridad del mensaje. |
| <i>sound</i> | Sonido | Indica un sonido para reproducir cuando el dispositivo recibe la notificación. |
| <i>time_to_live</i> | Vencimiento | Este parámetro especifica el tiempo (en segundos) que el mensaje se debe conservar en el almacenamiento de FCM si el dispositivo se encuentra sin conexión. |

Adaptado de Google, 2018.

5.3. Integración de FCM con la aplicación para Android

Para agregar los servicios de *Firebase Cloud Messaging* con la aplicación para *Android* es necesario asociar los datos de la aplicación a un nuevo proyecto desde el portal de *Firebase*. En este espacio será necesario definir un nombre para identificar al proyecto, además del país desde el cual se van a generar las notificaciones, a continuación se especificará el nombre del paquete de *Android* para registrar la aplicación, que tiene la siguiente estructura: “com.company.appname”.

Una vez registrados los datos principales, la plataforma de *Firebase* generará automáticamente el archivo “google-services.json”, que contiene las credenciales asociadas al propietario del proyecto y el nombre de la aplicación cliente asociada. Este archivo debe ser descargado y colocado en el directorio raíz del módulo de la aplicación para *Android*. En la figura 47 se indica el procedimiento a seguir para incluir correctamente el archivo JSON al proyecto.



Figura 47. Procedimiento para inclusión del archivo google-services.json al proyecto.

Adaptado de Google, 2018.

Para usar el complemento de *Google Services* para *Gradle* del proyecto es necesario agregar las siguientes dependencias a los archivos *Build Gradle* tanto a nivel de aplicación, como de proyecto. En el archivo *Gradle* a nivel de proyecto se debe añadir la siguiente dependencia: “*classpath* ‘com.google.gms:google-services:3.2.0’”, para el archivo a nivel de aplicación se debe añadir la siguiente línea en la sección de dependencias:

“*com.google.firebase:firebase-messaging:11.8.0*”, con esto se está integrando la biblioteca de FCM al proyecto. En la parte inferior de la declaración de dependencias con el siguiente comando se habilita el complemento de Google Services: “*apply plugin: ‘com.google.gms.google-services’*”.

Una vez que se ha integrado el SDK de FCM y el complemento de Google Services, es necesario crear dos clases java que extiendan los siguientes servicios:

- ***FirebaseInstanceIdService***, para administrar la creación y actualización de los *tokens* de registro. Esta instancia es necesaria para enviar mensajes a dispositivos específicos o para agrupar dispositivos por tipo de mensaje (Google, 2018).
- ***FirebaseMessagingService***, para tener administración de la carga de datos de mensajes entrantes y además manejar la recepción de notificaciones en aplicaciones ejecutándose en primer y segundo plano (Google, 2018).

Estas instancias deben ser declaradas en el archivo *AndroidManifest.xml* incluyendo el código que se muestra a continuación:

```
<!-- Servicio Google FCM -->
<service android:name=".fcm.MyFirebaseMessagingService">
  <intent-filter>
    <action android:name="com.google.firebase.MESSAGING_EVENT"/>
  </intent-filter>
</service>
<service android:name=".fcm.MyFirebaseInstanceIdService">
  <intent-filter>
    <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
  </intent-filter>
</service>
```

La clase java *myFirebaseMessagingService* que extiende el servicio *FirebaseMessagingService* debe manejar dentro de su método “*onMessageReceived*” el contenido del mensaje, ya sea que este posea carga de datos y notificación o solo una de las dos. En la clase *myFirebaseInstanceIdService* que extiende el servicio *FirebaseInstanceIdService* se puede manejar la recuperación del *token* dentro

del método “*onTokenRefresh*”, el *token* puede ser utilizado para enviar notificaciones a dispositivos individuales o crear grupos de dispositivos (Google, 2018).

El *token* de registro puede cambiar bajo las siguientes situaciones:

- La aplicación borra un identificador de instancia.
- La aplicación se restablece en un nuevo dispositivo.
- El usuario desinstala y vuelve a instalar la aplicación.
- El usuario borra los datos de la aplicación.

Para manejar la recepción del contenido de los mensajes enviados desde la consola de composición de mensajes de *Firebase* es necesario importar las librerías *NotificationCompat* y *NotificationManager*, además se creará el método “*showNotification*” dentro de la clase *myFirebaseMessagingService* que recibirá como parámetro “*remoteMessage.getData*” que contiene el cuerpo del mensaje, ya sea que es enviado como mensaje de notificación o mensaje de datos con el objetivo de que la notificación recibida se pueda mostrar tanto en primer plano como en segundo.

En la figura 48 se muestra un diagrama con el resumen de métodos y clases que conforman la implementación de FCM en *Android*.

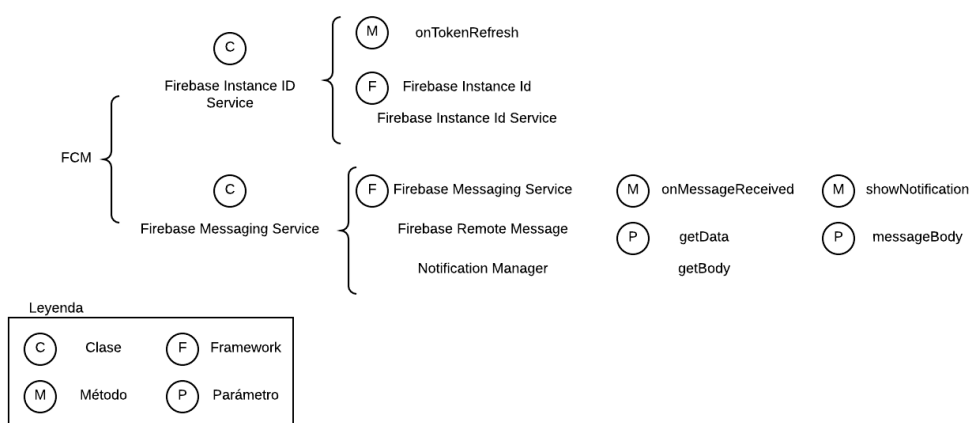


Figura 48. Diagrama de distribución de métodos y clases para implementar Google Firebase Cloud Messaging.

5.4. Integración de FCM con la aplicación para iOS

El proceso para integrar los servicios de *Firebase Cloud Messaging* con la aplicación para *iOS* es similar al seguido para integrarlo con *Android*. *Firebase* requiere asociar los datos de la aplicación a un proyecto, en este caso se lo añadirá como parte del proyecto existente previamente creado para la aplicación de *Android*. Se le indicará el nombre del paquete de *Xcode* para registrar la aplicación, que tiene la siguiente estructura: “com.company.appname”.

Una vez registrados los datos principales, la plataforma de *Firebase* generará automáticamente el archivo `GoogleService-Info.plist`, este archivo debe ser descargado y colocado en el directorio raíz del proyecto de *Xcode*, como se muestra en la figura 49.

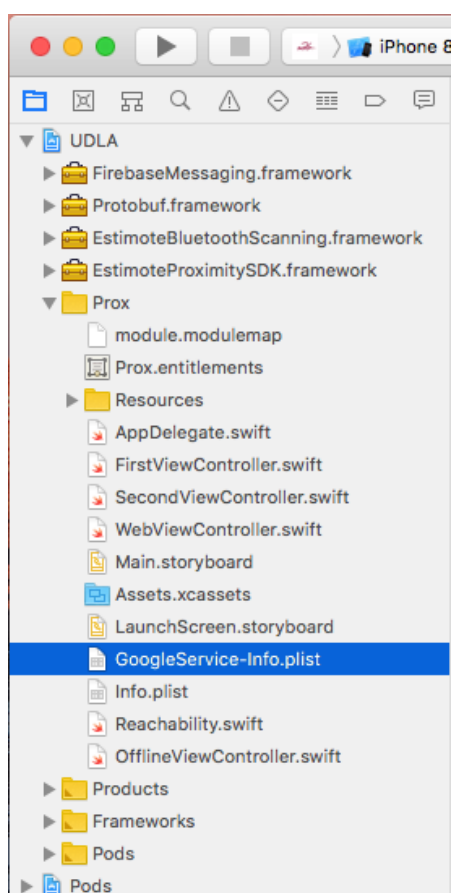


Figura 49. Archivo `GoogleService-Info.plist` en el directorio raíz del proyecto de *Xcode*.

El siguiente paso es incluir el SDK de *Firestore* y *Firestore Messaging* al proyecto, puede hacerse de dos maneras:

- Mediante *cocoapods* utilizando el terminal de comandos de Mac, o.
- Descargando del repositorio los *frameworks* de *Firestore* para incluirlos manualmente en el proyecto de *Xcode*.

Incluir los *frameworks* de *Firestore* mediante *cocoapods*, puede resultar redundante, pero es necesario contar con *cocoapods* instalado. Posteriormente hay que ubicarse en el directorio del proyecto mediante terminal de comandos y ejecutar la instrucción “pod init” para crear un archivo *Podfile* en la carpeta del proyecto. En este archivo se debe agregar la línea “pod ‘Firestore” y nuevamente en el terminal de comandos ejecutar “pod install”. Una vez concluida la instalación se generará un nuevo archivo con extensión *xcworkspace* que deberá ser utilizado en adelante para continuar configurando la aplicación. El directorio del proyecto debería contar con los siguientes archivos y carpetas, como se muestra en la figura 50.

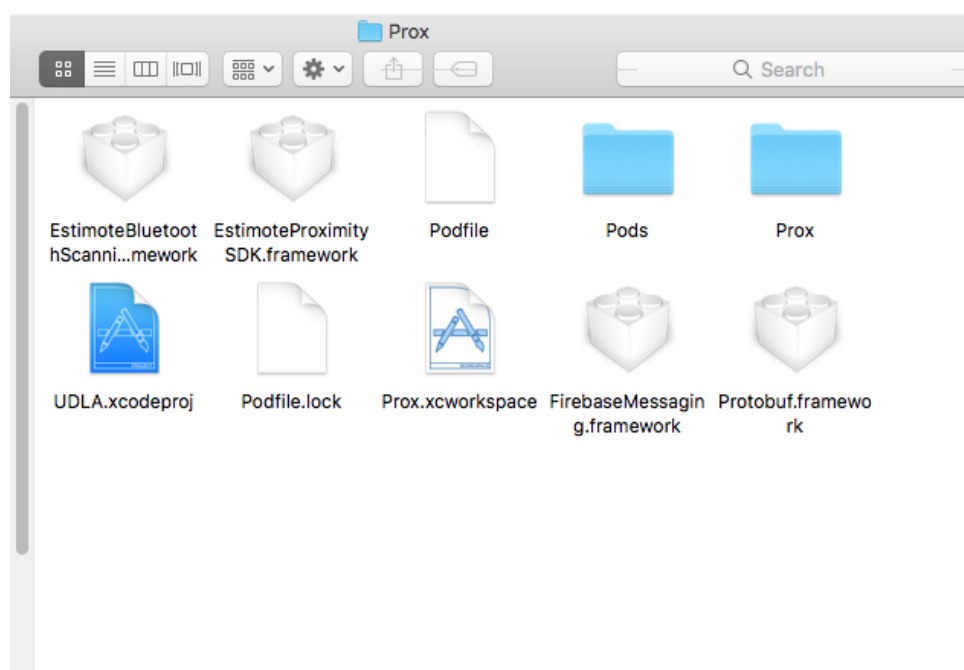


Figura 50. Directorio del proyecto en Xcode al agregar el *framework* de *Firestore* mediante *cocoapods*.

De igual manera se puede observar en la figura 50 los *frameworks* de *FirestoreMessaging* y *Protobuf*, que fueron agregados manualmente al proyecto

debido a una falla en el proceso del administrador de dependencias al incluir el *framework* de FCM mediante *cocoapods*.

Concluido este proceso, es necesario importar los *frameworks* de *Firebase* y *FirebaseMessaging* en el *AppDelegate* de la aplicación y dentro de la función *didFinishedLaunchingWithOptions* agregar el método “*FirebaseApp.configure*” para configurar la instancia de FCM. Se creará una función personalizada llamada *FirebaseHandler* que tendrá como parámetro la llamada *shouldEstablishDirectChannel* en verdadero que establece la conexión con el servidor de FCM.

Adicionalmente dentro del método *applicationDidEnterBackground*, que controla cuando la aplicación entra a segundo plano, se agrega el método *shouldEstablishDirectChannel* a falso, en el método *applicationDidBecomeActive*, que permite a la aplicación saber cuándo cambio de estado inactivo a activo, se agrega el método *FirebaseHandler* que establece nuevamente la conexión con el servidor de FCM.

Apple Push Notification service (APNs) es la pieza central de las notificaciones remotas para dispositivos Apple. Es un servicio robusto, seguro y eficiente para propagar información. En el lanzamiento inicial de la aplicación, el sistema automáticamente establece una conexión IP acreditada, encriptada y persistente entre la aplicación y el servicio APN. Esta conexión permite configurar a la aplicación para recibir notificaciones (Apple, 2017).

La otra mitad de la conexión para enviar notificaciones, que corresponde al canal seguro entre el servidor y el servicio APN requiere de configuración adicional de las características habilitadas para la aplicación en la cuenta de desarrolladores de Apple, en este caso notificaciones remotas además del uso de certificados criptográficos provistos por Apple (Apple, 2017). En la figura 51 se muestra el proceso que conlleva la entrega de una notificación remota.

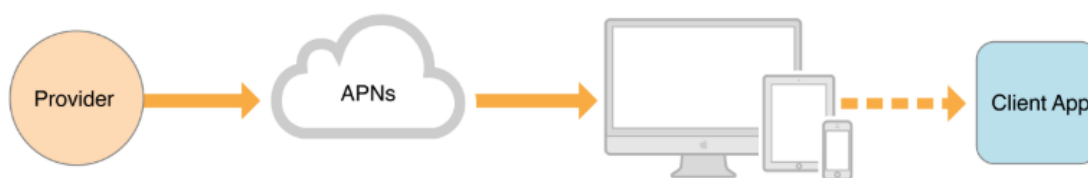


Figura 51. Proceso de entrega de notificación remota hacia una aplicación cliente.

Tomado de Apple, 2017.

Para crear un certificado criptográfico provisto por Apple será necesario acceder al portal de desarrolladores de Apple, seleccionar la opción “certificados, identificadores y perfiles” y escoger el nombre del paquete de la aplicación y habilitar la opción de notificaciones *push* de las opciones disponibles, como se muestra en la figura 52.

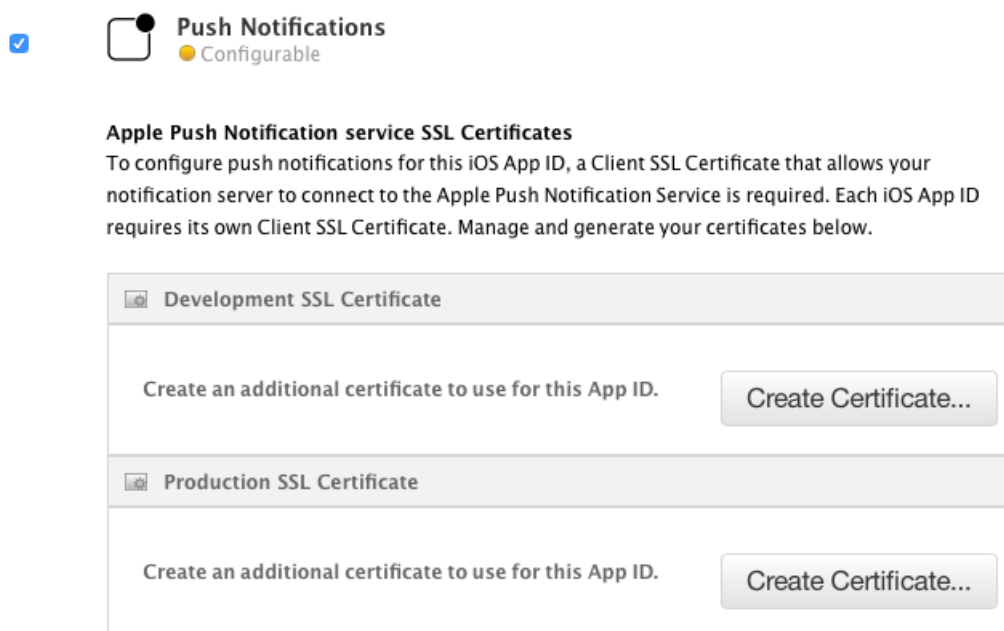


Figura 52. Habilitación de notificaciones push para crear el certificado para el servicio APN desde el portal de desarrolladores de Apple.

Para realizar una solicitud de generación del certificado, según las instrucciones provistas en el portal de desarrolladores se utilizará la opción “solicitar un certificado desde una autoridad” de la herramienta *Keychain Access* de *MacOs*. El asistente de certificados nos guiará durante el proceso, en el cual debemos especificar una cuenta de correo electrónico y un nombre para crear el certificado. Este certificado debe cargarse nuevamente al portal

de desarrolladores para que sea firmado por la Apple y sea descargado para incluirlo en la aplicación. El certificado firmado puede apreciarse en la figura 53.

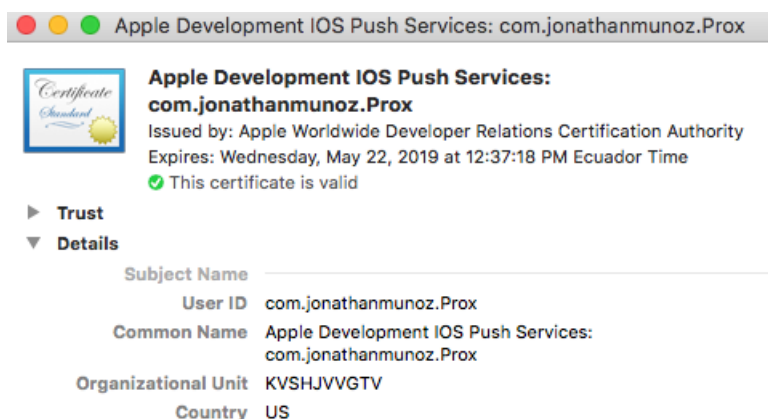


Figura 53. Certificado criptográfico firmado para servicios *push*.

El certificado obtenido debe ser subido a la consola de FCM como un archivo con extensión P12 utilizando la herramienta *Keychain Access* para generarlo; al subir el certificado se entrega al servidor de *Firebase* las credenciales de autenticación que permite al servicio APN verificar la vigencia del certificado y establecer la conexión entre la aplicación y el servicio APN.

Como se muestra en la figura 54 el servidor solicita establecer una conexión segura con el servicio APN utilizando seguridad de la capa de transporte (TLS), el servicio APN le entrega el certificado al servidor y este lo valida. En este punto, la confianza de conexión se encuentra establecida y el servidor está habilitado para realizar solicitudes de notificaciones *push* basadas en *tokens*. Cada solicitud de notificación enviada desde el servidor debe estar acompañada de un token de autenticación JSON Web (JWT). El servicio APN responde a cada solicitud con una respuesta HTTP/2 (Apple, 2017).

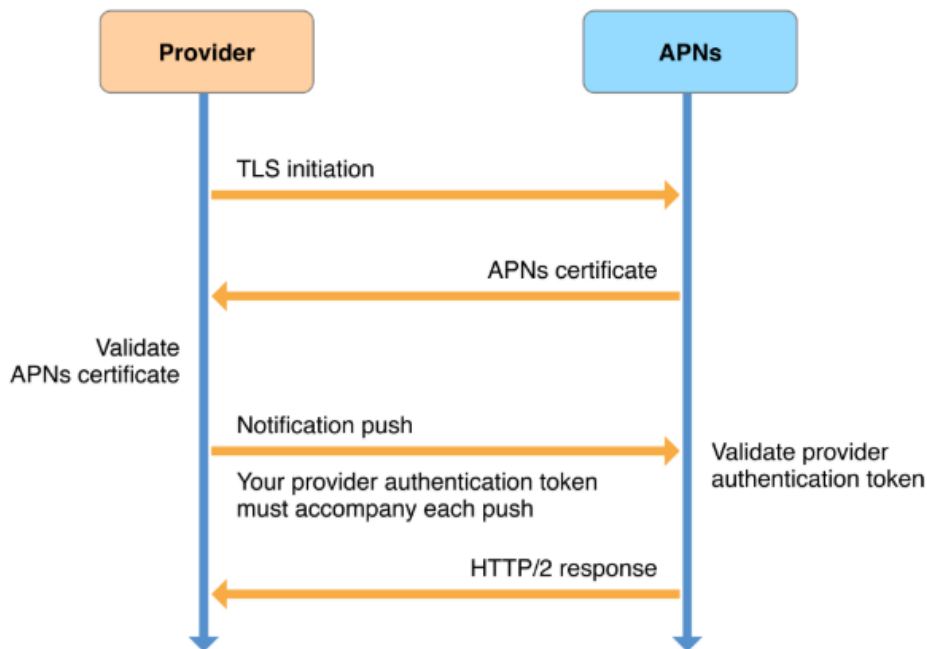


Figura 54. Proceso para establecer una conexión confiable para notificaciones push basadas en tokens.
Tomado de Apple, 2017.

Una vez concluida la configuración de notificaciones *push* tanto en FCM como en la aplicación, desde la consola de *Firebase* se puede enviar solicitudes de notificación hacia el servicio APN, el cual transporta la carga de la notificación hacia cada dispositivo. Al recibir una notificación, el sistema entrega la carga hacia la aplicación correspondiente en el dispositivo, y administra las interacciones con el usuario.

Si una notificación llega al dispositivo cuando se encuentra encendido, pero con la aplicación sin ejecutarse, el sistema aún puede mostrar la notificación. Si el dispositivo está apagado cuando el servicio APN envía la notificación, el servicio retiene la notificación y vuelve a intentarlo nuevamente más tarde.

En la figura 55 se muestra un diagrama con el resumen de métodos y clases que conforman la implementación de FCM en iOS.

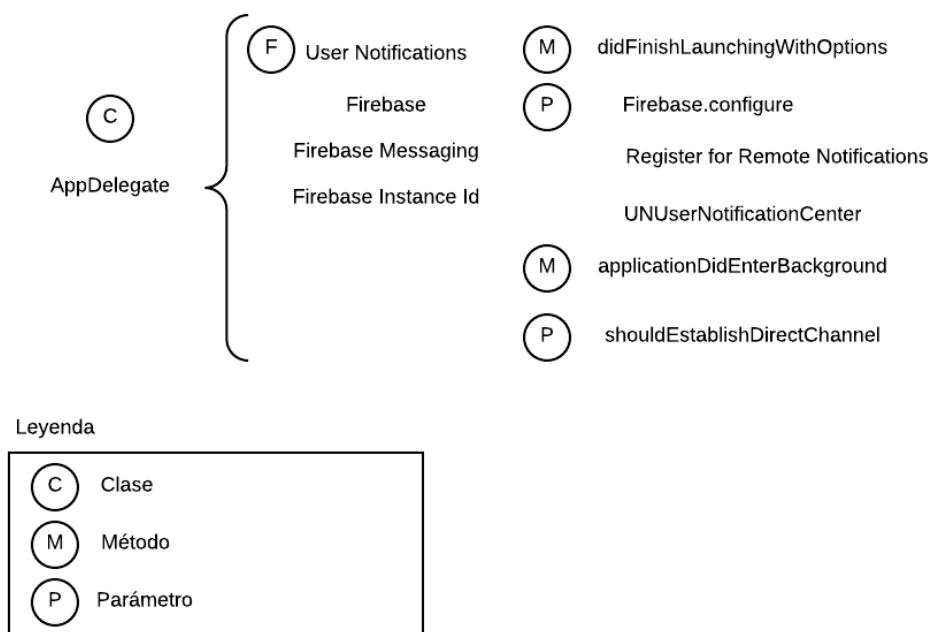


Figura 55. Diagrama de distribución de métodos y clases para implementar Google Firebase Cloud Messaging.

6. IMPLEMENTACIÓN DE LA SOLUCIÓN (BEACONS Y APLICACIÓN)

6.1. Implementación de beacons

Durante el proceso de desarrollo de la aplicación, los *beacons* se mantuvieron en modo de acceso de desarrollo o *development*, lo que permite a cualquier persona con una aplicación *sniffer* de *Bluetooth* o con la App de Estimote detectar a estos dispositivos de forma directa y hacer cambios sobre sus configuraciones sin necesariamente ser el propietario de los *beacons*.

Ahora que los *beacons* van a cambiar de la etapa de desarrollo a implementación, por motivos de seguridad el siguiente paso será cambiar el modo de acceso a los *beacons* de desarrollo a despliegue, lo que garantiza el acceso para hacer cambios a sus configuraciones solamente al propietario que registró a los *beacons*. De esta forma se restringe que terceros puedan realizar cambios sobre el tipo de paquete que está transmitiendo o su potencia de transmisión.

Desde la App de Estimote se realiza este cambio para aplicarlo directamente a

los *beacons*, de forma que cuando sean colocados en un espacio público de la Universidad los parámetros establecidos durante el proceso de desarrollo se mantengan y puedan ser modificados, en caso de ser necesario, solo por el propietario. Estos cambios del modo de acceso no afectan al proceso de escaneo y reconocimiento que realiza la aplicación para detectar *beacons* y mostrar el contenido en base a su proximidad con un *smartphone*.

La figura 56 muestra una captura de pantalla obtenida de la App de Estimote con los tres *beacons* configurados en modo de acceso *deployment* o despliegue, permitiendo así el acceso a modificaciones solo al propietario.

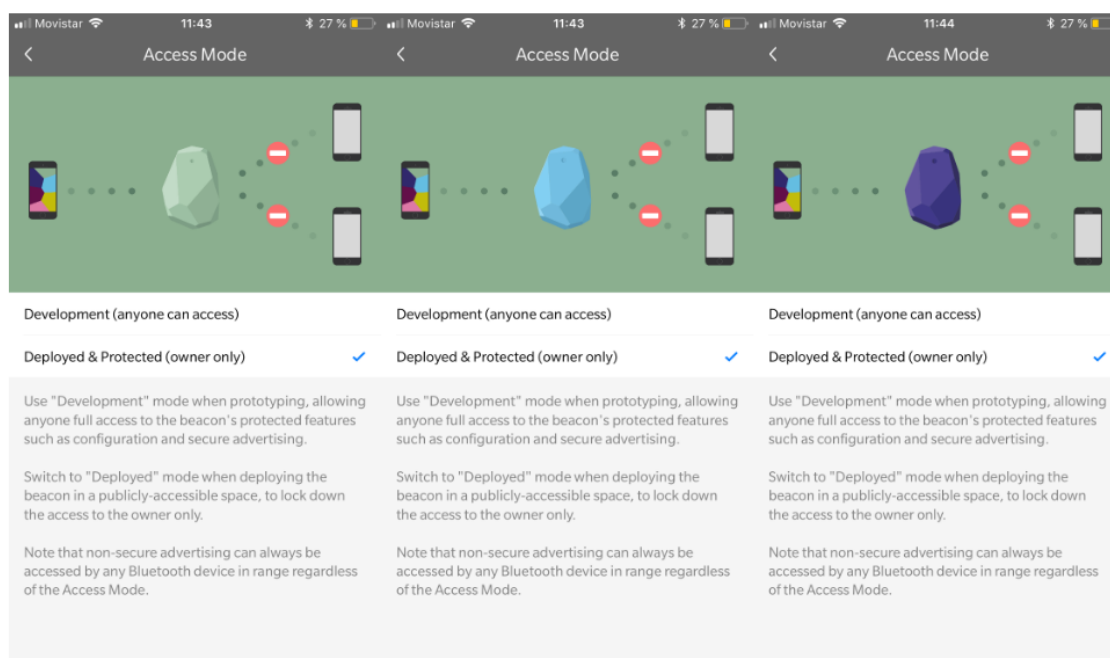


Figura 56. Captura de pantalla del modo de acceso protegido y desplegado para asegurar el acceso a los *beacons* en un entorno público.

6.2. Implementación de la aplicación

6.2.1. Preparación de la aplicación para distribución en Google Play Store

El proceso de preparación de una aplicación para su lanzamiento requiere completar las siguientes tareas principales:

- **Juntar materiales y recursos:** tarea conformada por claves criptográficas para firmar la aplicación, que consiste en un certificado firmado digitalmente que pertenece al desarrollador de la aplicación; un ícono para la aplicación y se sugiere incluir un contrato de licencia para el usuario final (CLUF) (Android Developers, 2018).
- **Configurar la aplicación para el lanzamiento:** esta tarea comprende en establecer un nombre apropiado al paquete, ya que más adelante no podrá ser cambiado. Como buena práctica es importante desactivar el registro de eventos, removiendo las llamadas a los métodos *Log* en los archivos de origen, así como inhabilitar la depuración eliminando los atributos que hacen referencia en el manifiesto de la aplicación. Además, se recomienda limpiar los directorios del proyecto en búsqueda de bibliotecas que utilicen recursos que ya no van a ser utilizados, revisar el manifiesto especificando solo aquellos permisos que sean requeridos por la aplicación, así como la configuración de los atributos de compilación de *Gradle* referentes a las versiones de la aplicación (Android Developers, 2018).
- **Compilar la aplicación para el lanzamiento:** una vez terminada la configuración de la aplicación, es posible compilarla en un archivo APK (*Android Application Package*) optimizado y firmado con la clave privada obtenida previamente (Android Developers, 2018).
- **Probar la aplicación para el lanzamiento:** el proceso de prueba de la aplicación permite garantizar que la aplicación se ejecute adecuadamente en dispositivos y redes reales, verificar el flujo y rendimiento de la interfaz de usuario (Android Developers, 2018).

Android requiere que todos los APK sean firmados digitalmente con un certificado antes de poder ser instalados, la firma de aplicaciones se utiliza para verificar la identidad del desarrollador y garantizar que futuras actualizaciones se realicen de forma segura.

Para la firma de aplicaciones es posible elegir entre dos posibles escenarios: usar la firma de aplicaciones de *Google Play* empleando la infraestructura de

Google o utilizando una clave de firma y *keystore* propios generados desde *Android Studio*.

6.2.1.1. Firma de aplicaciones usando Google Play

Al usar la firma de aplicaciones de *Google Play* se hará uso de dos claves: la firma de aplicaciones y la firma de subida. Google administra y protege la clave de firma de aplicaciones y la clave de subida debe ser almacenada por el desarrollador para firmar las aplicaciones que posteriormente serán subidas a *Google Play Store* (Android Developers, 2018).

Todo el proceso que conlleva la firma de aplicaciones con este método se indica en la figura 57, posteriormente Google verifica la identidad del certificado de subida y vuelve a firmar el APK con la clave de firma de aplicaciones para su distribución.

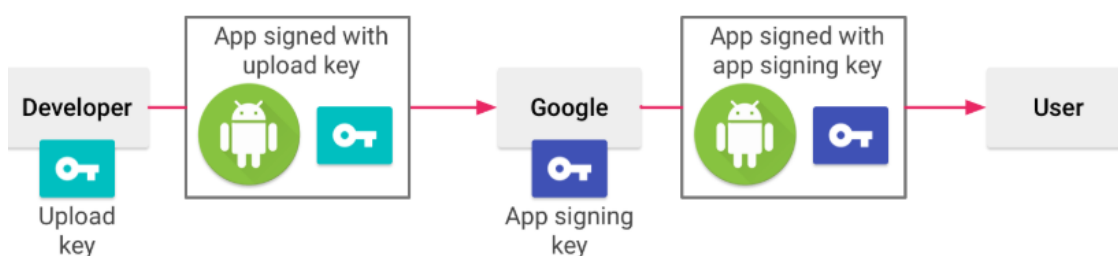


Figura 57. Proceso de firma de una aplicación utilizando la firma de aplicaciones de Google Play.

Tomado de Android Developers, 2018.

6.2.1.2. Firma de aplicaciones utilizando clave y keystore propios

La otra opción que ofrece *Google Play* para subir aplicaciones firmadas es utilizando una clave de firma de aplicaciones y *keystore* propios, que son generados desde *Android Studio* y donde el desarrollador es encargado del almacenamiento y protección de estos certificados criptográficos. Tanto la clave como el *keystore* poseen una contraseña (generalmente diferentes), que son definidas durante su creación. En caso de que el acceso o la contraseña de estos certificados se vea comprometido, Google no puede ofrecer métodos de recuperación y no será posible ofrecer nuevas versiones de la aplicación como actualizaciones de la aplicación original (Android Developers, 2018).

El proceso de compilar y generar un APK firmado desde *Android Studio* se realizará de forma local utilizando tanto la clave de firma de aplicaciones como el *keystore* para subir posteriormente el APK a *Google Play Store* para su distribución, como se muestra en la figura 59.

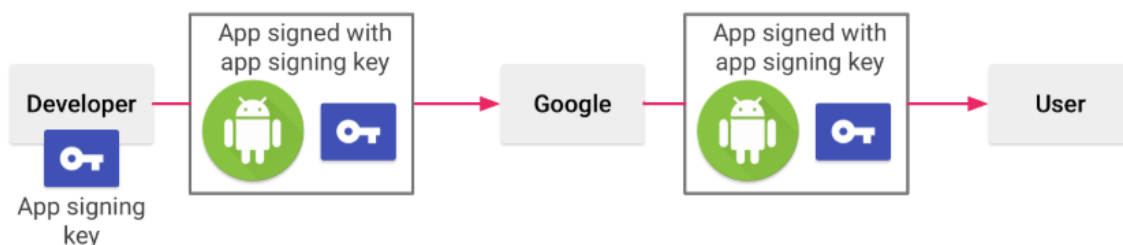


Figura 58. Proceso de firma de una aplicación al utilizar una clave de firma de aplicaciones propia.

Tomado de Android Developers, 2018.

La aplicación desarrollada se encuentra firmada bajo la firma de aplicaciones de Google Play, por las ventajas que ofrece sobre el almacenamiento en la infraestructura de Google.

6.2.2. Publicación de la aplicación en Google Play Store

Para publicar una aplicación en la tienda de *Google Play* es necesario darse de alta como desarrollador en el portal de *Google Play Console* y pagar mediante tarjeta de crédito la cantidad de \$25 dólares estadounidenses. Una vez confirmado el pago se podrá acceder a la consola de *Google Play* para subir una aplicación, tomando en consideración el proceso que conlleva preparar una aplicación para ser subida a la tienda.

El primer paso dentro de la consola es crear el perfil de desarrollador, con esta información registrada es posible comenzar a subir aplicaciones. Cuando se sube una aplicación es necesario definir el nombre comercial con el que será encontrada la aplicación en la tienda, en este caso la aplicación se llamará “UDLA *Beacon Scanner*”.

El siguiente paso en el proceso de publicación de la aplicación es definir la ficha de *Play Store*, que consiste en llenar información que describa la utilidad de la aplicación, recursos gráficos como capturas de pantalla (mínimo 2

máximo 8), un ícono en alta resolución de la aplicación y una imagen destacada, que puede considerarse como un banner; precio de la aplicación, disponibilidad en la tienda, países en los que podrá ser descargada y clasificación de contenido. En la figura 59 se muestra una captura de pantalla obtenida durante la creación de la ficha de *Play Store*.

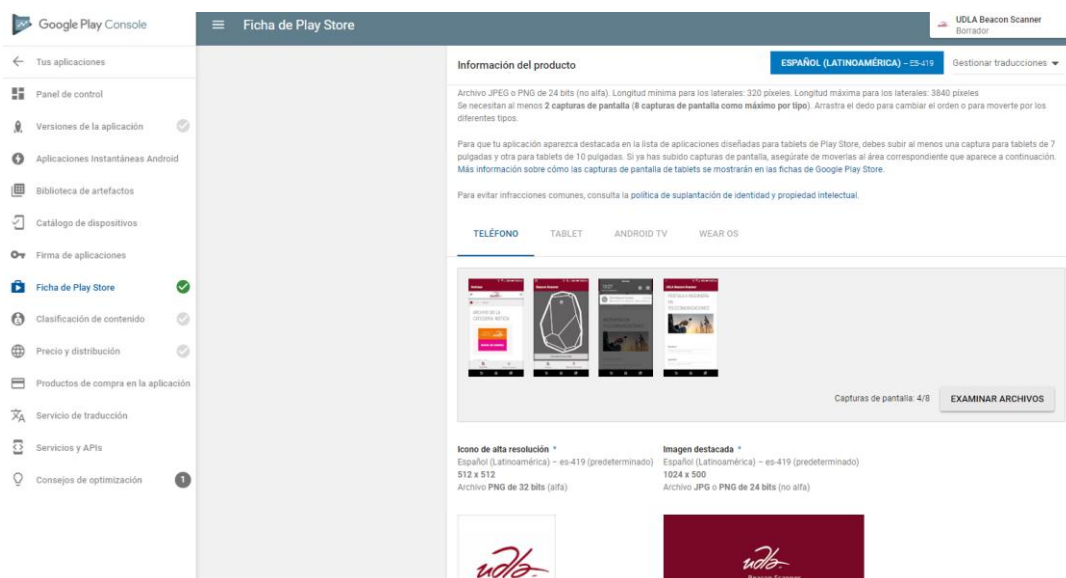


Figura 59. Captura de pantalla obtenida de la consola de Google Play durante la creación de la ficha de *Play Store* para la aplicación.

Una vez terminada la ficha de *Play Store* se define el tipo de segmento de distribución que tendrá la aplicación, las opciones disponibles son:

- **Producción:** cuando la aplicación haya superado la etapa de pruebas, sea estable y esté lista para ser utilizada con normalidad.
- **Beta:** permite que la aplicación esté disponible para todos los usuarios de la tienda, permite realizar pruebas de funcionamiento y es probable que tenga errores o sea inestable.
- **Alfa:** versión de la aplicación aún en periodo de prueba, pero restringida a un grupo específico de usuarios, que pueden acceder mediante un URL a descargar la aplicación o usando comunidades de Google+ y no aparece en la tienda de *Google Play*.
- **Prueba interna:** permite que un máximo de 100 usuarios pueda acceder a probar la aplicación, su distribución se hace exclusivamente mediante un URL.

La aplicación al estar en un periodo de prueba abierta (Beta) permitirá que los usuarios accedan a la tienda y descarguen la aplicación para realizar las pruebas correspondientes que permitan identificar errores. Una vez superadas las pruebas es posible cambiar a producción sin necesidad de cargar un nuevo APK.

Al cargar el APK de la aplicación, *Google Play Console* verifica que esté correctamente firmada y somete el APK a un proceso de evaluación en búsqueda de posibles errores, problemas de seguridad, bloqueos y revisiones de rendimiento. Este proceso puede durar un par de horas hasta que las verificaciones hayan sido superadas y la aplicación pueda estar disponible en la tienda de *Google Play*. Concluida la evaluación del APK se generará un informe previo al lanzamiento que contiene un resumen sobre los parámetros que fueron evaluados. En la figura 60 se muestra una captura de pantalla de la evaluación del rendimiento y uso de recursos de la aplicación “UDLA Beacon Scanner” en varios dispositivos.

| Modelo del dispositivo | Uso medio de CPU (porcentaje) | Promedio de datos enviados a través de la red (bytes/s) | Promedio de datos recibidos a través de la red (bytes/s) | Uso medio de memoria (bytes) | Tiempo de inicio (ms) |
|------------------------|-------------------------------|---|--|------------------------------|-----------------------|
| LG G6 | 7,66 % | 1,7 mil | 19 mil | 178 M | 443 |
| Xperia XZ Premium | 3,20 % | 1,1 mil | 9,6 mil | 157 M | 240 |
| Moto G4 Play | 9,62 % | 1,2 mil | 13 mil | 88 M | 588 |
| P8 Lite | 5,94 % | 1,7 mil | 10 mil | 86 M | 767 |
| Mate 9 | 4,65 % | 645 | 10 mil | 148 M | 212 |
| Galaxy S7 Edge | 5,31 % | 563 | 5,8 mil | 138 M | 365 |
| Galaxy J7(2016) | 4,28 % | 743 | 9,1 mil | 99 M | 428 |
| Pixel | 2,27 % | 1,1 mil | 9,5 mil | 134 M | 245 |

Figura 60. Captura de pantalla obtenida del informe previo al lanzamiento de la aplicación en la consola de Google Play.

Al publicar la aplicación en la tienda de *Google Play*, la información que previamente fue llenada en la ficha de *Play Store* aparecerá en los detalles de la aplicación. Tal como muestra la figura 61.

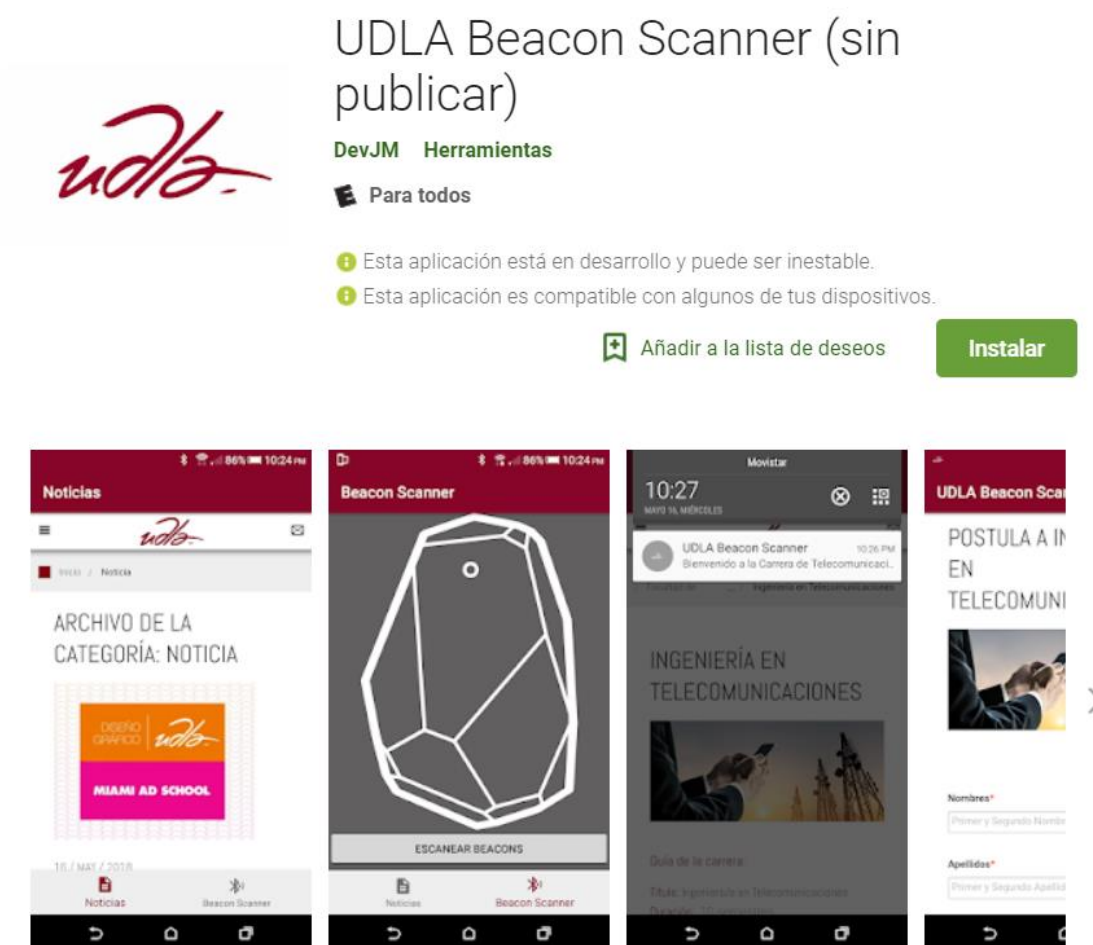


Figura 61. Captura de pantalla obtenida de la tienda Google Play que muestra a la aplicación publicada.

Transcurrido el periodo de prueba de la aplicación, se procedió a cambiar su estado de versión beta a versión de producción, esta nueva versión requirió subir un nuevo APK con la corrección de errores y ajustes para la versión definitiva. La aplicación se encuentra disponible en la tienda de Google Play Ecuador para todos los dispositivos Android ejecutando API 21 (Android 5.0 *Lollipop*) o superior.

La consola de *Google Play* además ofrece herramientas para dar seguimiento a las aplicaciones que son publicadas con información estadística sobre el

número de descargas o porcentaje de usuarios activos en el mes, reporte de errores, seguimiento de comentarios y calificación de la aplicación.

6.2.3. Preparación de la aplicación para distribución en App Store

La preparación de una aplicación previo a su lanzamiento para distribución requiere seguir una serie de lineamientos establecidos por el proveedor de servicio, los mismos que serán detallados a continuación:

- **Establecer el ID del paquete:** identifica de forma única a la aplicación a lo largo del sistema ya que una vez que se carga la primera compilación no es posible cambiar este identificador en *iTunes Connect*, plataforma para administrar la distribución de aplicaciones en *App Store* (Apple Inc., 2018).
- **Establecer el número de versión y la cadena de compilación:** antes de distribuir la aplicación permite que *iTunes Connect* utilice esta información para emparejar la compilación con la versión de la aplicación. Estos datos son utilizados para identificar la aplicación en el sistema (Apple Inc., 2018).
- **Editar la configuración de la información de implementación:** estos datos son utilizados por *iTunes Connect* para determinar las versiones de *iOS* y la configuración de orientación de pantalla y dispositivos que soporta la aplicación.
- **Preparación de recursos gráficos:** entre estos recursos se consideran los íconos y la pantalla de lanzamiento de la aplicación. Los íconos de la aplicación deben cumplir con los lineamientos de tamaño y calidad para pasar las validaciones de *iTunes Connect* (Apple Inc., 2018).
- **Crear el archivo de la aplicación:** el archivo está compuesto por un paquete que incluye la aplicación junto con información de librerías externas. La creación del archivo puede ser utilizada para distribuir una aplicación desde *iTunes Connect* (Apple Inc., 2018).

La creación del archivo compilará todos los elementos utilizados durante la fase de desarrollo de la aplicación como los SDK y los *framework* para el escaneo Bluetooth y envío de mensajes remotos. Algunos *frameworks* son compilados

como binarios embebidos (Estimote SDK) y otros solo como librerías adjuntas (*Firebase*). *Xcode* ejecuta validaciones preliminares en el archivo y podría mostrar mensajes de advertencia en caso de errores que deben ser corregidos para completar la creación del archivo.

Posteriormente el archivo debe pasar por un proceso de revisión, firmado y validación de integridad, en el cual se verifica la consistencia del código, de los binarios, se firma digitalmente la aplicación con el certificado de desarrollador y se verifica el archivo “info.plist” que administra las configuraciones de la aplicación y de los permisos para utilizar los recursos del teléfono. En caso de que existan errores, deben ser corregidos hasta que la validación sea exitosa. En la figura 62 se muestra una captura de pantalla tomada durante el proceso de la validación del archivo generado desde *Xcode*.

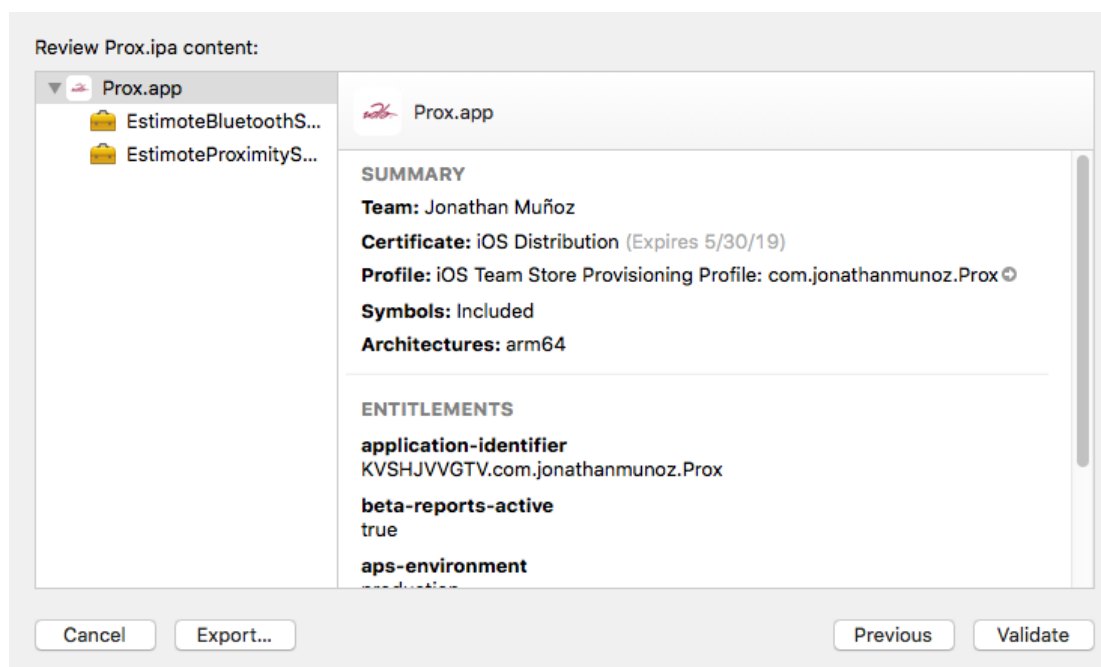


Figura 62. Captura de pantalla tomada durante la validación de integridad del archivo de la aplicación.

La primera validación se ejecuta en *Xcode*, y de ser exitosa la siguiente validación se ejecuta cuando el archivo de la aplicación es subido a las compilaciones de *iTunes Connect*.

6.2.4. Publicación de la aplicación en App Store

Para publicar aplicaciones en la tienda *App Store* de Apple es necesario darse de alta como desarrollador en el portal de desarrolladores, el costo es de \$99 dólares estadounidenses y tiene una duración de un año, que puede ser renovable. Posee dos módulos:

- **Certificados, identificadores y perfiles:** es el módulo que administra los certificados, perfiles, registro de dispositivos e identificadores necesarios para desarrollar una aplicación.
- **iTunes Connect:** es la plataforma para publicar y administrar las aplicaciones en la tienda App Store.

Todas las aplicaciones subidas a *iTunes Connect* son sometidas a pruebas de evaluación de calidad, Apple revisa manualmente todas las aplicaciones que pretenden estar disponibles en la tienda *App Store*, por lo que es posible que el proceso demore unos días mientras se revisa que la aplicación cumpla con los requisitos necesarios exigidos.

Desde *iTunes Connect* es posible publicar aplicaciones de dos formas: directamente a la tienda de *App Store* o mediante el lanzamiento de una versión beta para realizar pruebas. El proceso comienza con la creación de un registro para la aplicación en la plataforma, es requerido ingresar los datos de la plataforma en la que estará disponible la aplicación (*iOS* o *tvOS*), nombre, idioma, identificador del paquete (debe coincidir con el paquete de desarrollo) y un identificador SKU (*Stock Keeping Unit*).

Por lo tanto, para comenzar a probar la aplicación en su fase beta en otros dispositivos es necesario subir una compilación de la aplicación a *iTunes Connect*. Previo a estar disponible la compilación seleccionada será sometida a revisión por parte del equipo de Apple. Quienes sean parte del proceso de prueba necesitan utilizar la aplicación *TestFlight*, que provee de mecanismos para acceder a la descarga de la aplicación y reporte de problemas.

La forma de distribuir la versión beta de la aplicación se realiza mediante una invitación por correo electrónico para un grupo específico de personas y les

permite probar la aplicación. En esta fase la aplicación no se encuentra aún disponible, mucho menos visible, en la tienda oficial de aplicaciones *App Store*. En la figura 63 se muestra una captura de pantalla obtenida de *iTunes Connect* donde se puede observar el número de invitaciones e instalaciones de la versión beta de la aplicación.

| Compilación | Usuarios de iTunes Connect ? | Probadores externos ? | Invitaciones ? | Instalaciones ? |
|-------------|---|-----------------------------------|----------------|-----------------|
| 2 | ● Lista para las pruebas Caduca en 89 días | ● En pruebas Caduca en 89 días | 5 | 5 |

Figura 63. Captura de pantalla del número de invitaciones e instalaciones de la compilación de la versión beta de la aplicación en *TestFlight*.

Probar la aplicación en varios dispositivos permitirá probar las capacidades de la aplicación al utilizar *Bluetooth* y recibir notificaciones remotas, además de comprobar que la interfaz gráfica sea compatible con las diferentes dimensiones de pantalla que maneja cada dispositivo. Una vez que la aplicación haya superado su etapa de pruebas es posible utilizar el archivo de compilación de la versión beta para pasarla a producción desde *iTunes Connect*.

Una vez efectuadas las pruebas por un mínimo de dos semanas se procedió a cambiar el estado de la aplicación de su versión beta a producción. Este proceso consiste en subir el archivo de la aplicación junto con información referente a descripción, clasificación de categoría de uso por edades, capturas de pantalla para someterla a revisión y aprobación por parte del equipo evaluador de Apple para su distribución oficial en *App Store*. Adicional a esto fue necesario añadir instrucciones para la persona que evaluará la aplicación,

en este caso particular fue necesario enviar un video de la aplicación funcionando junto a los beacons para evitar contratiempos. La aplicación se encuentra disponible en la tienda de Ecuador y Estados Unidos para todos los usuarios que posean cuentas en las tiendas de estos países puedan descargarla siempre y cuando el dispositivo cuente con iOS 11 o superior.

7. PROTOCOLO DE PRUEBAS

Se realizó un proceso de pruebas para las dos aplicaciones, en el cual se evaluaron parámetros de proximidad a los *beacons* y recepción de notificaciones *push*, que buscan identificar posibles falencias y corregirlas en esta fase.

7.1. Pruebas de la aplicación en Android

El protocolo de pruebas de la aplicación está enfocado a evidenciar los resultados obtenidos durante la fase de desarrollo e implementación de la solución, por este motivo se escogió el periodo de tiempo comprendido entre el 17 de abril de 2018 al 30 de mayo de 2018, que considera el inicio del proceso de desarrollo de la aplicación y pruebas de funcionamiento y comunicación “*in-house*”. De igual manera abarca las pruebas integrales de la aplicación durante la versión beta, en la que se evaluaron los siguientes parámetros, que son parte fundamental de la aplicación:

- Respuesta de la aplicación móvil cuando el dispositivo se encuentra en el rango de proximidad de cada *beacon*.
- Recepción de notificaciones remotas.

Además, se identificaron y corrigieron errores y se estimó el tiempo aproximado de respuesta de la aplicación al cargar el contenido web de la carrera asociada a cada *beacon* con apoyo de las herramientas de uso de recursos de *Android Studio*.

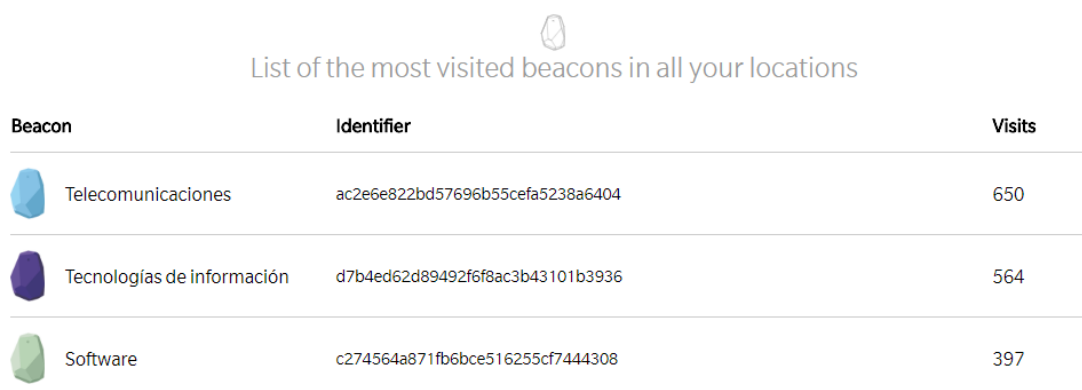
7.1.1. Evaluación de parámetros de proximidad

Como se describió en el capítulo 3 sobre el desarrollo de la aplicación, el uso de credenciales obtenidas de *Estimote Cloud* permiten diferenciar los datos

asociados a un beacon específico. De esta manera se posibilita a la aplicación enviar información de uso cuando se produce un evento de ingreso del smartphone en el rango de alcance de un beacon. Los datos enviados al portal de *Estimote Cloud* facilitan la obtención de informes analíticos sobre: número de visitantes, *beacons* más visitados y tiempo promedio de permanencia cerca de un *beacon*.

La forma de representar la proximidad que registró un dispositivo al estar cerca de un beacon se contabiliza en la plataforma de *Estimote Cloud* como visitas, los eventos considerados como visitas se generan cuando un smartphone con la aplicación móvil y *Bluetooth* habilitado entra en rango de alcance del beacon.

En la figura 64 se muestra un listado de los *beacons* más visitados y el número de visitas que ha recibido cada *beacon*, esta lista muestra un resumen del total de veces que la aplicación entró en el rango de un *beacon*, se consideran las visitas realizadas durante la etapa de desarrollo e implementación.






| Beacon | Identifier | Visits |
|--|----------------------------------|--------|
|  Telecomunicaciones | ac2e6e822bd57696b55cefa5238a6404 | 650 |
|  Tecnologías de información | d7b4ed62d89492f6f8ac3b43101b3936 | 564 |
|  Software | c274564a871fb6bce516255cf7444308 | 397 |

Figura 64. Lista de beacons más visitados durante la etapa de desarrollo e implementación.

El 17 de mayo se celebró el “Día Mundial de las Telecomunicaciones”, y la UDLA como miembro de la ITU (Unión Internacional de Telecomunicaciones) lo celebró con un evento en el Auditorio de la Sede Queri, al cual asistieron estudiantes de varios colegios y autoridades del sector de las Telecomunicaciones del país para ver los proyectos que alumnos de la Universidad están desarrollando. Durante este evento se aprovechó para probar la aplicación en aproximadamente 12 dispositivos *Android* con

estudiantes tanto de la Universidad como de los colegios visitantes. Además, se impartió una charla a un grupo de estudiantes del Colegio Alvernia sobre la funcionalidad de la aplicación. Las fotografías tomadas durante este evento se encuentran en los anexos de este documento.

En la figura 65 se muestra un cuadro de tendencia estadística, en el cual se puede apreciar el flujo de visitantes diarios, en el eje Y se encuentra el número de visitantes y en el eje X los días de la semana. El día con mayor número de visitas (8) se produjo el 17 de mayo. Para que una visita sea reportada en el portal de Estimote *Cloud* debe ser mayor a 1 minuto y menor a 8 horas.

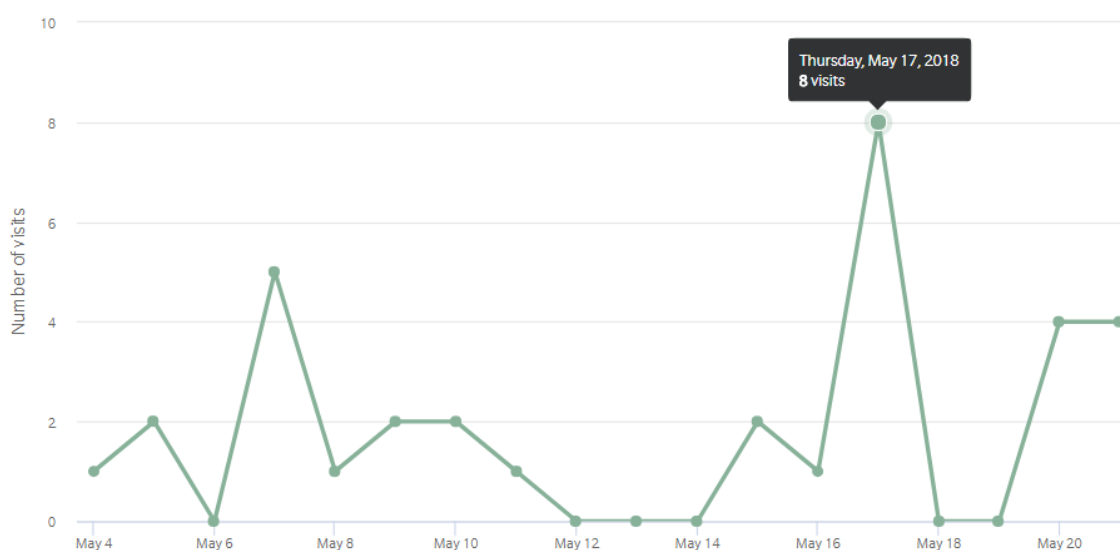


Figura 65. Cuadro de tendencia estadística del número de visitantes que ingresaron en el rango de los beacons en un periodo de tiempo.

7.1.2. Evaluación de notificaciones remotas y uso de la aplicación

Con la implementación del *framework* de *Firestore Cloud Messaging* en la aplicación fue posible probar el envío de notificaciones remotas desde la consola de FCM. Durante el evento del 17 de mayo se envió un mensaje de bienvenida a todos quienes había descargado la aplicación. Las estadísticas de la recepción del mensaje para esta aplicación se la puede obtener utilizando la consola de mensajes permite conocer el porcentaje de recepción que tuvo ese mensaje del global de dispositivos la cual registra su token al utilizar la aplicación. En la figura 66 se muestra un gráfico de barras, en el cual se

aprecia que del 100% de usuarios que pudo recibir la notificación solamente el 33% abrió la notificación, esto quiere decir 6 usuarios.

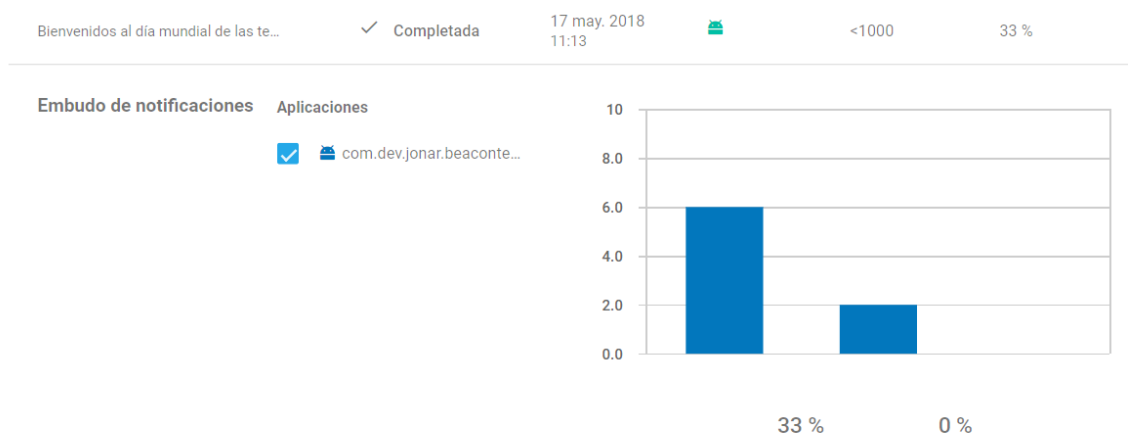


Figura 66. Gráfico de barras del porcentaje de usuarios que abrieron la notificación enviada durante el evento del 17 de mayo.

Del portal de administración de aplicaciones Google *Play Console* se pudieron obtener datos referentes al número de smartphones Android en los cuales fue probada la aplicación, así como sus versiones de sistema operativo. De este informe se sabe que la aplicación fue probada en 8 dispositivos con versiones de sistema operativo Android 6, 7, 7.1, 8 y 8.1.

Mediante la herramienta de uso de recursos, disponible en *Android Studio*, se pudo determinar que la ejecución de la aplicación toma aproximadamente 1 segundo al ser abierta cuando está en segundo plano y dependiendo de la velocidad de Internet o de los datos celulares entre 15 y 20 segundos para mostrar el contenido de la sección de noticias. El escaneo de beacons toma entre 10 y 28 segundos para identificar al beacon más cercano y mostrar la información de la carrera, tomando en consideración que el smartphone debe estar en el rango de alcance del beacon (1 metro).

7.2. Pruebas de la aplicación en iOS

Las pruebas de la aplicación en iOS están orientadas a evaluar y evidenciar los resultados obtenidos durante la fase de desarrollo e implementación de la solución, en el periodo de tiempo comprendido entre el 01 de mayo de 2018 hasta el 02 de junio de 2018. La versión beta de la aplicación tuvo demoras en

su publicación, razón por la cual no pudo ser probada durante el evento “Día Mundial de las Telecomunicaciones”. Sin embargo, la aplicación fue probada posteriormente con un grupo de 8 dispositivos Apple con diversos modelos y tamaños de pantalla.

El portal de Estimote Cloud, provee un listado del número de visitas filtrado por aplicación que permite conocer que aplicación ha generado mayor tráfico durante esta etapa como se muestra en la figura 67.



| Application | Visits |
|---------------|--------|
| Proximity iOS | 22 |
| Proximity | 18 |

Figura 67. Listado del número de visitas generadas por aplicación (Android y iOS).

7.2.1. Evaluación de notificaciones push y uso de la aplicación

Para evaluar la versión beta de la aplicación, se envió una invitación a 8 personas por correo electrónico que tenían en su posesión los siguientes modelos de iPhone con sistema operativo iOS 11 o superior:

- iPhone X (pantalla 5.8”)
- iPhone 7 Plus (pantalla 5.5”)
- iPhone 6, 6s y 6s Plus (pantallas 4.7” y 5.5” respectivamente)
- iPhone SE y 5S (pantalla 4”)

Durante las pruebas se pudo verificar que la interfaz de usuario en diferentes tamaños de pantalla no se mostraba correctamente, este error estaba siendo ocasionado porque algunos objetos de la vista tenían aplicados limitantes en el diseño de la interfaz que no eran compatibles cuando la dimensión de la pantalla del smartphone variaba. Se procedió a corregir este inconveniente de la interfaz y se lanzó una actualización que abordaba el problema para que la aplicación sea compatible con diferentes tamaños de pantalla.

Con los 8 usuarios que recibieron la invitación para probar la aplicación en su versión beta se evaluó el escaneo de *beacons* y la recepción de notificaciones *push* enviadas desde la consola de FCM. Como se muestra en la figura 68, del total de notificaciones enviadas, 6 fueron recibidas porque no todos los usuarios habilitaron el permiso que solicita la aplicación al ejecutarse por primera vez para recibir notificaciones. Solo 1 de las notificaciones fue abierta, razón por la cual el porcentaje de recepción de la notificación fue del 17%.

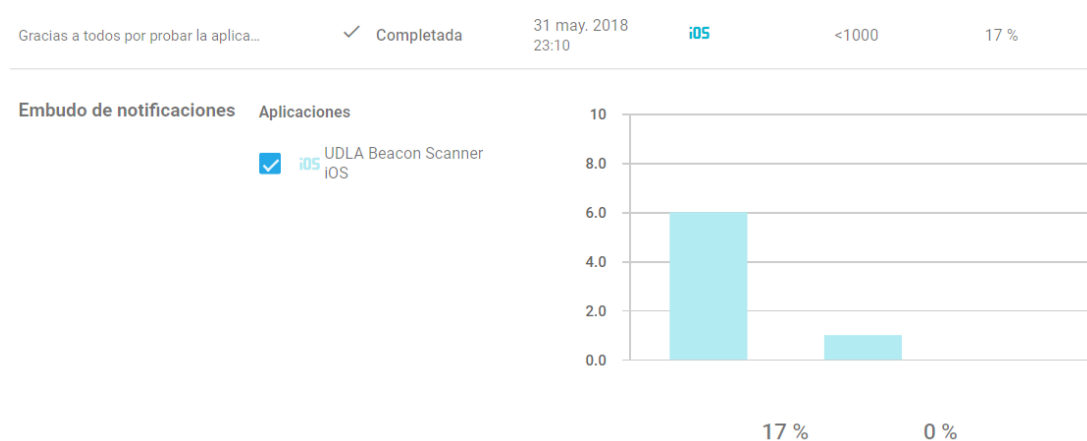


Figura 68. Gráfico de barras del porcentaje de usuarios que abrieron la notificación de prueba enviada a los usuarios de la aplicación en iOS.

A través de la herramienta de evaluación de uso de recursos de Xcode, se pudo determinar que en iOS la aplicación toma aproximadamente entre 1 a 3 segundos en ejecutarse al ser abierta por primera vez o al abrirse estando segundo plano. Dependiendo de la velocidad de Internet o de los datos celulares muestra el contenido de la sección de noticias entre 14 y 22 segundos. El escaneo de beacons toma entre 8 y 20 segundos para identificar al beacon más cercano y mostrar la información de la carrera, tomando en consideración que el smartphone debe estar en el rango de alcance del beacon (1 metro).

Todos los parámetros evaluados, así como los datos obtenidos son preliminares considerando que forman parte del prototipo de la aplicación y contienen información global del proceso de desarrollo y pruebas. Las muestras tomadas al evaluar el uso de recursos de la aplicación en ambos sistemas operativos no representan el comportamiento generalizado en todos

los smartphones y pueden variar dependiendo las condiciones de navegación, acceso a internet y la concesión de permisos que solicita la aplicación al usuario.

8. CONCLUSIONES Y RECOMENDACIONES

8.1. Conclusiones

Trabajar con *beacons Bluetooth* que ofrecen más de un paquete de *advertising* permite que sea posible escoger el tipo de paquete más adecuado para desarrollar una aplicación, si se considera que la aplicación estará disponible para dos sistemas operativos móviles diferentes, seleccionar un tipo de paquete propietario (*iBeacon* o *Eddystone*) no sería lo más adecuado por posibles incompatibilidades. Por este motivo, el paquete de *advertising* de *Estimote Monitoring* fue el adecuado para implementar la solución por su compatibilidad con ambos sistemas operativos.

La tendencia de las aplicaciones móviles está orientada a establecer un vínculo de interacción entre el usuario y el contenido, por esta razón el desarrollo de la aplicación móvil permite que las señales de radiofrecuencia enviadas por los *beacons* sean interpretadas por la aplicación y para el usuario sea transparente recibir información interactiva cuando los smartphones entran en el rango de alcance de estos dispositivos *Bluetooth*.

La implementación de notificaciones remotas mediante *Google Firebase Cloud Messaging* además de permitir al usuario ser informado sobre eventos importantes en desarrollo o que están por comenzar, permite además conocer con exactitud el porcentaje de recepción y apertura de la notificación que tuvo el mensaje desde la consola de composición de mensajes de FCM.

Si bien, para subir aplicaciones es necesario registrarse como desarrollador en ambas plataformas y pagar una cuota, el precio incluye el acceso a múltiples herramientas de desarrollo, información estadística de descargas y uso de la aplicación, además de acceso anticipado a versiones de prueba de sistemas operativos y SDKs que permiten realizar las pruebas por adelantado en las nuevas versiones antes de estar disponibles para todos, por lo que la relación costo-beneficio es adecuada.

Las tiendas de aplicaciones son más que un portal para cargar los archivos que contienen las aplicaciones, cada una posee filtros adicionales que permiten

evaluar la calidad de una aplicación en búsqueda de posibles fallos o falta de cumplimiento en los lineamientos de calidad que cada una establece. *App Store* posee controles más rigurosos en cuanto a evaluación de aplicaciones, ya que el análisis se realiza manualmente previo a estar disponible en la tienda; mientras que *Google Play Store* ejecuta un proceso automatizado que evalúa el APK en múltiples dispositivos con diversas configuraciones y versiones de sistema operativo.

La evaluación de las versiones beta de la aplicación permitió que otros usuarios probaran la aplicación en varios dispositivos y de esta manera recibir retroalimentación e identificar posibles problemas durante la ejecución o al retomar la aplicación del segundo plano. Este periodo sirvió para ajustar parámetros, corregir problemas de interfaz de usuario y optimización en el proceso de escaneo de *beacons*.

Las pruebas realizadas durante el evento “Día Mundial de las Telecomunicaciones” permitió a muchos estudiantes de los colegios visitantes tener la posibilidad de probar la aplicación de primera mano, al mismo tiempo sirvió de incentivo para dar a conocer a los visitantes una de las múltiples habilidades que están en capacidad de desarrollar al estudiar esta y otras carreras de la Facultad de Ingenierías y Ciencias Aplicadas en la Universidad de Las Américas.

8.2. Recomendaciones

Debido a que en el mercado existe una amplia variedad de *beacons Bluetooth* es recomendable conocer de antemano cual es el objetivo de la aplicación, tiempo de duración de la batería (aunque los dispositivos BLE se caracterizan por consumir poca energía), tipos de paquetes de *advertising (iBeacon, Eddystone)* únicos o múltiples, compatibilidad con dispositivos que soporten BLE y capacidad para adaptarse o reconocer nuevas versiones de *Bluetooth*.

Familiarizarse con los lenguajes de programación que manejan los IDE *Xcode* y *Android Studio* es necesario, ya que cada lenguaje posee una estructura diferente para declarar variables, métodos o constructores. Cada lenguaje

posee una biblioteca que describe la funcionalidad de cada método y *framework* en caso de ser necesario despejar alguna duda sobre su uso.

La estructura de organización de archivos es otro de los factores que influyen al momento de crear directorios dentro de un proyecto, así como nombrar y ubicar determinados archivos de recursos externos como imágenes. Por ejemplo, en *Android Studio* el nombre de archivo de una imagen no puede contener espacios, mientras que en Xcode el nombre de archivo es indiferente para importarlo al proyecto.

Se recomienda también tomar en consideración los tipos de permisos a los que requiere tener acceso la aplicación, estos deben ser declarados en el manifiesto de la aplicación de *Android* o deben ser habilitados en la pestaña *capabilities* de Xcode. Además, se debe tener en consideración que estos permisos darán apertura para que la aplicación acceda a utilizar recursos del dispositivo móvil durante su ejecución, por lo que se debe habilitar estrictamente los permisos necesarios.

Se debe tomar en consideración que existen ciertas limitaciones de acceso a determinados servicios que pueda necesitar una aplicación, sobre todo en Xcode, donde fue necesario pagar la cuota de desarrollador para desbloquear algunos servicios que no están disponibles para los desarrolladores con cuentas gratuitas. Por ejemplo, para implementar notificaciones *push* es necesario habilitar el servicio que está disponible solamente al darse de alta como desarrollador.

Es recomendable probar la aplicación mientras es desarrollada en un dispositivo físico, ya que el simulador de dispositivos integrado en los IDE Xcode y *Android Studio*, no están en capacidad de simular todas las características que posee un dispositivo real, como acceder a la cámara o encender *Bluetooth*, que es necesario para probar el funcionamiento de la aplicación.

Una buena práctica es lanzar una versión beta, antes de subir una aplicación oficialmente a la tienda de aplicaciones, esta versión permite a otros usuarios

probarla en múltiples dispositivos, con diferentes tamaños de pantalla y sistema operativo. Este proceso de evaluación permitirá identificar posibles errores, recibir retroalimentación y corregir cualquier desperfecto durante la fase de desarrollo, para posteriormente subir una aplicación de calidad cumpliendo con los estándares que demanda cada tienda.

Con el objetivo de obtener información más fiable se recomienda tomar más datos que permitan caracterizar los parámetros con mayor detalle y exactitud, debido a que fueron obtenidos mientras la aplicación estaba en desarrollo y en su fase de pruebas y no representan información real sobre su uso masivo en múltiples dispositivos con la versión definitiva de la aplicación.

Se recomienda extender el uso de la aplicación a otras carreras de la Universidad, que permita poner a disposición sus beneficios para facilitar el registro a futuros alumnos postular a la carrera de su elección.

REFERENCIAS

- Aftab, M. U. (2017). *Building Bluetooth Low Energy Systems*. Birmingham: Packt Publishing Ltd.
- Al Agha, K., Pujolle, G., & Ali-Yahiya, T. (2015). *Mobile and wireless networks*. Recuperado el 22 de marzo de 2018 de <https://ebookcentral-proquest-com.bibliotecavirtual.udla.edu.ec>
- Android Developers. (2018). Preparar el lanzamiento. Recuperado el 28 de mayo de 2018 de <https://developer.android.com/studio/publish/preparing>
- Android Developers. (s.f.). Conoce Android Studio. Recuperado el 13 de abril de 2018 de <https://developer.android.com/studio/intro/index.html?hl=es-419>
- Apple. (2014). *Getting Started with iBeacon*. Recuperado el 11 de abril de 2018 de <https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf>
- Apple. (2017). *APNs Overview*. Recuperado el 24 de mayo 24 de 2018 de https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html#//apple_ref/doc/uid/TP40008194-CH8-SW1
- Apple Inc. (2018). *Archive, Distribute, and Test*. Recuperado el 30 de mayo de 2018 de <https://help.apple.com/xcode>
- Apple Inc. (2018). *Submit apps to the App Store*. Recuperado el 30 de mayo de 2018 de <https://help.apple.com/xcode>
- Apple. (s.f.). *Xcode*. Recuperado el 23 de mayo de 2018 de <https://developer.apple.com/xcode>
- Bhargava, M. (2017). *IoT Projects with Bluetooth Low Energy*. Birmingham:

Packt Publishing Ltd.

Bluetooth SIG. (s.f.). *GATT Overview*. Recuperado el 26 de marzo de 2018 de <https://www.bluetooth.com/specifications/gatt/generic-attributes-overview>

Bluetooth SIG. (s.f.). *SIG introduces Bluetooth low energy wireless technology, the next generation of Bluetooth wireless technology*. Recuperado el 22 de marzo de 2018 de <https://www.bluetooth.com/news/pressreleases/2009/12/17/sig-introduces-bluetooth-low-energy-wireless-technologythe-next-generation-of-bluetooth-wireless-technology>

Changsu, J., Kyungjun, K., Jihun, S., Bhaya, N. S., & Kijun, H. (2017). *Topology configuration and multihop routing protocol for bluetooth low energy networks*. *IEEE Access*, 9587-9598. doi:10.1109/ACCESS.2017.2707556

Cisco. (s.f.). *Wireless, LAN (WLAN)*. Recuperado el 12 de julio de 2018 de <https://www.cisco.com/c/en/us/tech/wireless-2f-mobility/wireless-lan-wlan/index.html>

Davidson, R., Akiba, Cufí, C., & Townsend, K. (2014). *Getting Started with Bluetooth Low Energy*. Recuperado el 09 de abril de 2018 de <https://www.safaribooksonline.com/library/view/getting-started-with/9781491900550/>

Estimote. (s.f.). *Add proximity events to an iOS app*. Recuperado el 05 de mayo de 2018 de <https://developer.estimote.com/proximity/ios-tutorial/>

Google. (2018). *Firestore Cloud Messaging*. Recuperado el 17 de mayo de 2018 de <https://firebase.google.com/docs/cloud-messaging>

Gupta, N. (2013). *Inside Bluetooth Low Energy*. Retrieved from <https://ebookcentral-proquest-com.bibliotecavirtual.udla.edu.ec>

Hortelano, D., Olivares, T., Ruiz, M. C., Garrido-Hidalgo, C., & López, V. (2017).

From Sensor Networks to Internet of Things. Bluetooth Low Energy, a Standard for This Evolution. doi:10.3390/s17020372

- Mahdi Darroudi, S., & Gomez, C. (2017). *Bluetooth Low Energy Mesh Networks: A Survey.* doi:10.3390/s17071467
- Olivares, J. L. (2009). *Modelo de cobertura para redes inalámbricas de interiores.* Recuperado el 29 de marzo de 2018 de <http://bibing.us.es/proyectos/abreproy/11761/fichero/Volumen1%252F6-Cap%C3%ADtulo2+-+Redes+inal%C3%A1mbricas+de+%C3%A1rea+personal+%28WPA+N%29.pdf+>
- Ramírez, A. J. (2006). *Redes inalámbricas de área personal - WPAN.* Recuperado el 29 de marzo de 2018 de CITEI: http://www.oas.org/en/citel/infocitel/2006/enero/bluetooth_e.asp
- Revelo, J. (2016). *Android Push Notifications Con Firebase Cloud Messaging.* Recuperado el 17 de mayo de 2018 de <http://www.hermosaprogramacion.com/2016/06/android-push-notifications-firebase-cloud-messaging/>
- Rodríguez, T. (2017). *Kotlin es un lenguaje oficial en Android: ¿que implicaciones tiene y por qué es tan importante?* Recuperado el 23 de abril de 2018 de Xataka Android: <https://www.xatakandroid.com/programacion-android>
- Tiano, J. (2016). *Learning Xcode 8.* Recuperado el 24 de abril de 2018 de <https://www.safaribooksonline.com/library/view/learning-xcode-8/9781785885723/ch02s02.html>

ANEXOS

ANEXO 1

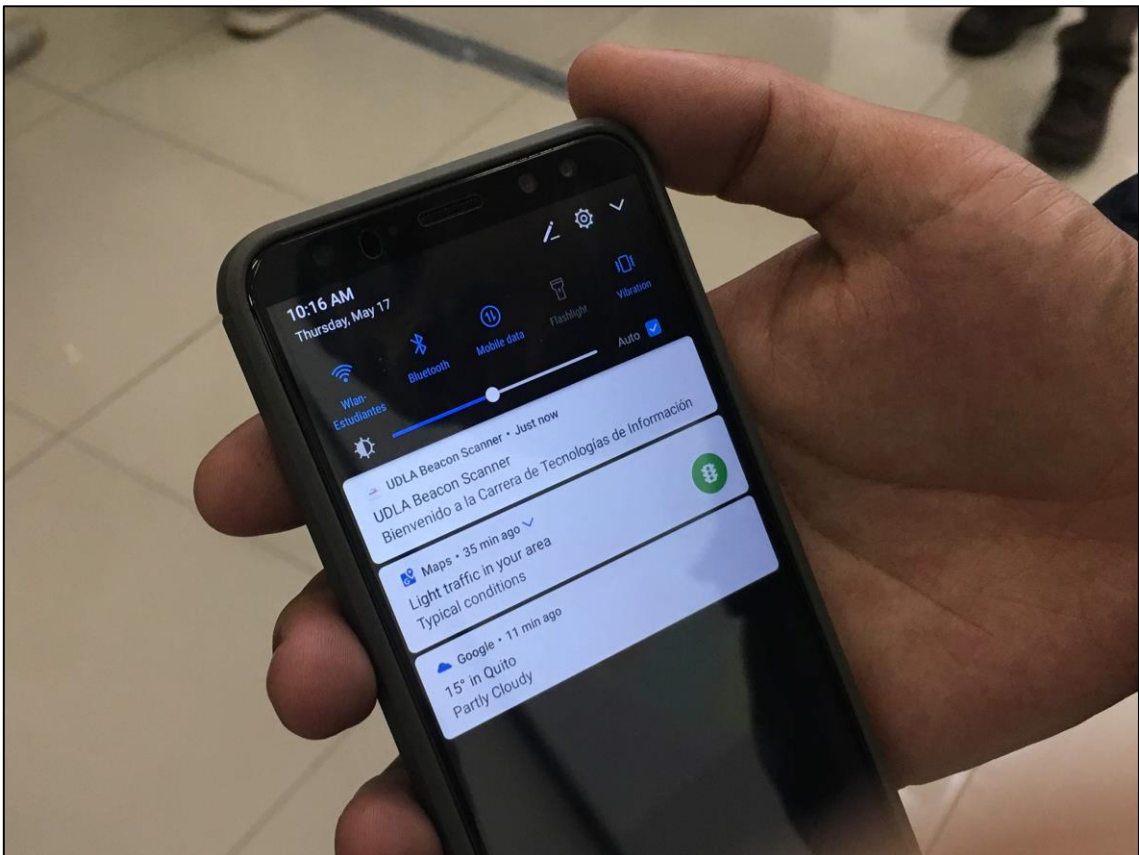
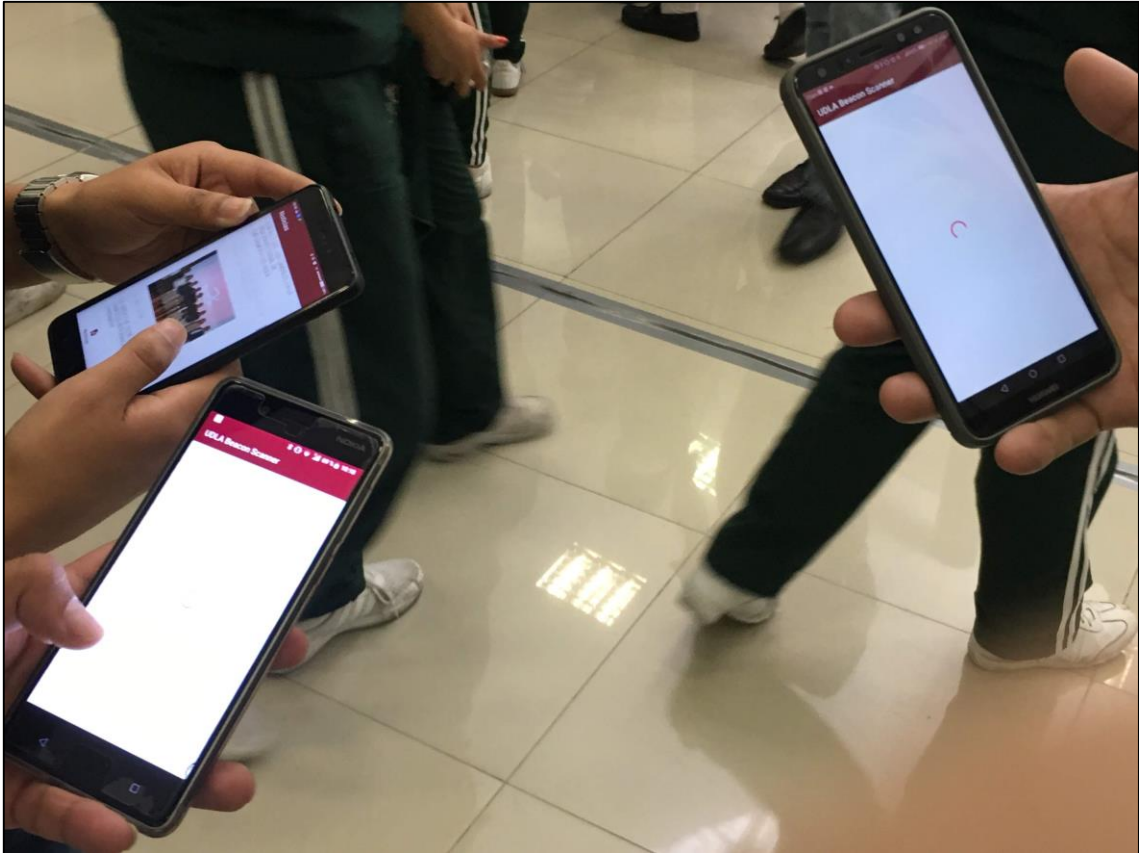
FOTOGRAFÍAS DE LA CHARLA EXPLICATIVA DEL PROYECTO DE TITULACIÓN A ALUMNOS DEL COLEGIO ALVERNIA DURANTE EL EVENTO “DÍA MUNDIAL DE LAS TELECOMUNICACIONES” EN EL CAMPUS QUERI



ANEXO 2

FOTOGRAFÍAS DEL EVENTO “DÍA MUNDIAL DE LAS TELECOMUNICACIONES” EN EL CAMPUS QUERI CON ALUMNOS PROBANDO LA APLCACIÓN EN ANDROID





ANEXO 3

CODIGO PARA ESCANEAR BEACONS BLUETOOTH EN ANDROID

```
//Declaramos las variables de URL de las carreras
String urlTelecom = "https://goo.gl/tgkZh2";
String urlSoft = "https://goo.gl/3oMoN2";
String urlTI = "https://goo.gl/siUqZm";

    notificacion = new NotificationCompat.Builder(this);
    notificacion.setAutoCancel(true);

    //Credenciales de Estimote Cloud
    EstimoteCloudCredentials cloudCredentials = new
EstimoteCloudCredentials("app-id", "token");

    //Creamos el proximity observer
    this.proximityObserver = new
ProximityObserverBuilder(getApplicationContext(),
cloudCredentials).withOnErrorAction(new Function1<Throwable, Unit>() {

        @Override
        public Unit invoke(Throwable throwable) {
            return null;
        }
    })

        .withBalancedPowerMode().build();

/* Zona de proximidad Telecomunicaciones */

    ProximityZone Telecomunicaciones =
this.proximityObserver.zoneBuilder().forAttachmentKeyAndValue("owner",
"telecom").inFarRange().withOnEnterAction(new
Function1<ProximityAttachment, Unit>() {
        @Override
        public Unit invoke(ProximityAttachment attachment) {

            //creamos el webview
            webView = (WebView) findViewById(R.id.webView);
            webView.loadUrl(urlTelecom);

            //creamos la notificacion
            notificacion.setSmallIcon(R.drawable.ic_stat_logo_udla_blanco);
            notificacion.setContentText("Bienvenido a la Carrera
de Telecomunicaciones");
            notificacion.setSound(soundUri);

            return null;
        }
    })

        .withOnExitAction(new Function1<ProximityAttachment,
Unit>() {
            @Override
            public Unit invoke(ProximityAttachment
proximityAttachment) {
                //creamos la notificacion
```

```

notificacion.setSmallIcon(R.drawable.ic_stat_logo_udla_blanco);
        notificacion.setContentTitle("UDLA Beacon Scanner");
        notificacion.setContentText("Gracias por
visitarnos");
                return null;
        }
    })
    .create();
    this.proximityObserver.addProximityZone(Telecomunicaciones);

/* Zona de proximidad Software */

    ProximityZone Software =
this.proximityObserver.zoneBuilder().forAttachmentKeyAndValue("owner",
"soft").inFarRange().withOnEnterAction(new
Function1<ProximityAttachment, Unit>() {
    @Override
    public Unit invoke(ProximityAttachment attachment) {

        //creamos el webview
        webView = (WebView) findViewById(R.id.webView);
        webView.loadUrl(urlSoft);

        //creamos la notificacion
notificacion.setSmallIcon(R.drawable.ic_stat_logo_udla_blanco);
        notificacion.setContentTitle("UDLA Beacon Scanner");
        notificacion.setContentText("Bienvenido a la Carrera
de Software");
        return null;
    }
})
    .withOnExitAction(new Function1<ProximityAttachment,
Unit>() {
    @Override
    public Unit invoke(ProximityAttachment
proximityAttachment) {

notificacion.setSmallIcon(R.mipmap.ic_launcher);
        notificacion.setContentTitle("UDLA Beacon Scanner");
        notificacion.setContentText("Gracias por visitarnos");

        return null;
    }
})
    .create();
    this.proximityObserver.addProximityZone(Software);

/* Zona de proximidad Tecnologías de información */

    ProximityZone T_I =
this.proximityObserver.zoneBuilder().forAttachmentKeyAndValue("owner",
"ti").inFarRange().withOnEnterAction(new
Function1<ProximityAttachment, Unit>() {
    @Override
    public Unit invoke(ProximityAttachment attachment) {

        //creamos el webview
        webView = (WebView) findViewById(R.id.webView);

```

```

        webView.loadUrl(urlTI);

        //creamos la notificacion
        notificacion.setSmallIcon(R.mipmap.ic_launcher);
        notificacion.setContentTitle("UDLA Beacon Scanner");
        notificacion.setContentText("Bienvenido a la Carrera
de Tecnologías de Información");

        return null;
    }
})
.withOnExitAction(new Function1<ProximityAttachment,
Unit>() {
    @Override
    public Unit invoke(ProximityAttachment
proximityAttachment) {
        notificacion.setSmallIcon(R.mipmap.ic_launcher);
        notificacion.setTicker("Nueva Notificación");

        notificacion.setWhen(System.currentTimeMillis());
        notificacion.setContentTitle("UDLA Beacon Scanner");
        notificacion.setContentText("Vuelve pronto");
        return null;
    }
})
.create();
this.proximityObserver.addProximityZone(T_I);

// Método para detector conexión Bluetooth

RequirementsWizardFactory
    .createEstimoteRequirementsWizard()
    .fulfillRequirements(this,
        new Function0<Unit>() {
            @Override
            public Unit invoke() {
                proximityObserver.start();
                return null;
            }
        },
        new Function1<List<? extends Requirement>,
Unit>() {
            @Override
            public Unit invoke(List<? extends
Requirement> requirements) {
                return null;
            }
        },
        new Function1<Throwable, Unit>() {
            @Override
            public Unit invoke(Throwable throwable) {
                return null;
            }
        }
    );
}

```


ANEXO 4

CÓDIGO PARA ESCANEAR BEACONS EN IOS

```
import UIKit
import WebKit
import EstimoteProximitySDK
import UserNotifications
import CoreBluetooth

class WebViewController: UIViewController, WKNavigationDelegate {

    @IBOutlet weak var webView: WKWebView!
    @IBOutlet weak var myActivityIndicator: UIActivityIndicatorView!

    //Declaramos las url de las carreras
    let urlSoft = URL(string: "https://goo.gl/3oMoN2")
    let urlTi = URL(string: "https://goo.gl/siUqZm")
    let urlTelecom = URL(string: "https://goo.gl/tgkZh2")

    //Propiedad para mantener el Proximity Observer
    var proximityObserver: EPXProximityObserver!

    override func viewDidLoad() {
        super.viewDidLoad()

        //Agregamos credenciales de aplicacion de Estimote Cloud
        let cloudCredentials = EPXCloudCredentials(appID: "app-id",
            appToken: "token")

        //Creamos el Proximity Observer
        self.proximityObserver = EPXProximityObserver(
            credentials: cloudCredentials,
            errorBlock: { Error in
                print("Error del Proximity Observer: \(Error)")
            })

        //Definimos zonas de proximidad

        //Zona de proximidad Software

        let zone1 = EPXProximityZone(range: .far, attachmentKey: "owner", attachmentValue:
"soft")
        zone1.onEnterAction = { attachment in
            //Envio de notificacion
            let content = UNMutableNotificationContent()
            content.title = "UDLA Beacon Scanner"
            content.body = "Bienvenidos a la Carrera de Software"
            content.badge = 1

            let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)
            let request = UNNotificationRequest(identifier: "timerDone", content: content, trigger:
trigger)
```

```

UNNotificationCenter.current().add(request, withCompletionHandler: nil)

//Cargamos el webView con el url de la carrera
self.webView.load(URLRequest(url: self.urlSoft!))
self.webView.navigationDelegate = self

}
zone1.onExitAction = {attchmement in
//Envio de notificacion
let content = UNMutableNotificationContent()
content.title = "UDLA Beacon Scanner"
content.body = "Gracias por visitarnos"
content.badge = 1

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)
let request = UNNotificationRequest(identifier: "timerDone", content: content, trigger:
trigger)

UNNotificationCenter.current().add(request, withCompletionHandler: nil)
}

//Zona de proximidad Telecomunicaciones

let zone2 = EPXProximityZone(range: .far, attachmentKey: "owner", attachmentValue:
"telecom")
zone2.onEnterAction = { attachment in
print("Bienvenidos a la Carrera de Telecomunicaciones")
//Envio de notificacion
let content = UNMutableNotificationContent()
content.title = "UDLA Beacon Scanner"
content.body = "Bienvenidos a la Carrera de Telecomunicaciones"

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)
let request = UNNotificationRequest(identifier: "timerDone", content: content, trigger:
trigger)

UNNotificationCenter.current().add(request, withCompletionHandler: nil)

//Cargamos el webView con el url de la carrera
self.webView.load(URLRequest(url: self.urlTelecom!))
self.webView.navigationDelegate = self

}
zone2.onExitAction = {attchmement in
//Envio de notificacion
let content = UNMutableNotificationContent()
content.title = "UDLA Beacon Scanner"
content.body = "Gracias por visitarnos"

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)
let request = UNNotificationRequest(identifier: "timerDone", content: content, trigger:
trigger)

UNNotificationCenter.current().add(request, withCompletionHandler: nil)
}

//Zona de proximidad Tecnologías de Información

```

```

let zone3 = EPXProximityZone(range: .far, attachmentKey: "owner", attachmentValue: "ti")
zone3.onEnterAction = { attachment in
    //Envio de notificacion
    let content = UNMutableNotificationContent()
    content.title = "UDLA Beacon Scanner"
    content.body = "Bienvenidos a la Carrera de Tecnologias de la Informacion"

    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)
    let request = UNNotificationRequest(identifier: "timerDone", content: content, trigger:
trigger)

    UNUserNotificationCenter.current().add(request, withCompletionHandler: nil)

    //Cargamos el webView con el url de la carrera
    self.webView.load(URLRequest(url: self.urlTi!))
    self.webView.navigationDelegate = self

}
zone3.onExitAction = {attchmement in
    //Envio de notificacion
    let content = UNMutableNotificationContent()
    content.title = "UDLA Beacon Scanner"
    content.body = "Gracias por visitarnos"
    content.badge = 1

    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)
    let request = UNNotificationRequest(identifier: "timerDone", content: content, trigger:
trigger)

    UNUserNotificationCenter.current().add(request, withCompletionHandler: nil)
}

self.proximityObserver.startObserving([zone1, zone2, zone3])
}

```

