ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

AUTO CHECK IN PARA ALOJAMIENTOS TURÍSTICOS CON BEACONS

AUTO CHECK IN FOR TOURIST ACCOMMODATIONS USING BEACONS

Realizado por
INMACULADA BARRERA RAMÍREZ
Tutorizado por
ISAAC AGUDO RUIZ
Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO 2017

Fecha defensa:
El Secretario del Tribunal

Resumen:

El presente Trabajo de Fin de Grado ha tenido como objetivo presentar las ventajas de utilización de la tecnología Beacon en aplicaciones donde los servicios basados en proximidad pueden ser un valor añadido. Para ello se propone una solución consistente en desarrollar un sistema de check in automatizado para alojamientos turísticos haciendo uso simplemente de teléfonos inteligentes y dichos beacons. Se realiza un estudio de los distintos protocolos Beacon para elegir el que mejor se adapta a nuestros propósitos, así como un análisis para realizar una implementación lo más ajustada a nuestros objetivos. La solución final se compone de una aplicación móvil para iOS, con distintos roles para los distintos tipos de usuario que se necesitan, así como una web desde donde administrar el sistema. El backend del sistema estará implementado bajo la plataforma Firebase, que nos facilita todos los servicios que necesitamos en cuanto a bases de datos en tiempo real y comunicación.

Palabras claves: Beacon, iBeacon, Bluetooth, iOS, Firebase, XCode, Objective-C, JavaScript, Turismo.

Abstract:

The following research has had as goal to present the advantages of using Beacon technoogy in applications where services based on proximity can be an added value. In order to achieve this, we propose a solution consisting in developing an automated check-in system for tourist accommodations using smartphones and beacons. A study of the different Beacon protocols is carried out to choose the one that best suits our purposes, as well as an analysis to make an implementation that fits our final objectives. This solution will contain a mobile application for iOS devices, with different roles for the different types of user that are needed, as well as a web application to manage the complete system. This system backend will be

implemented under the Firebase platform, which provides all the mobile services we need in terms of real time database and communication among others.

Keywords: Beacon, iBeacon, Bluetooth, iOS, Firebase, XCode, Objective-C, JavaScript, Tourism.

Indice:

1. Introducción

1.1. Motivación

El Internet de las cosas ya es una realidad. Cada vez será más habitual encontrarnos con objetos conectados, ya sea a través de Bluetooth o de cualquier otra tecnología inalámbrica.

Desde hace unos años convivimos con unos dispositivos singulares, los **Beacons**, que siguen un standard de comunicación Bluetooth de baja energía (BLE), con la ventaja de estar ya presente en los sistemas operativos móviles más recientes, tanto Android e iOS. Se trata de un sistema para determinar la localización de los dispositivos de comunicación inalámbrica y las personas, para así controlar y ajustar las operaciones de los dispositivos basándose en su localización. Tiene especial utilidad en aquellos escenarios donde no se dispone de señal GPS, como por ejemplo dentro de edificios.

Tecnologías tan relativamente simples como la de los beacons, junto con una serie de aplicaciones, aportan una flexibilidad sin precedentes a las empresas, tiendas, centros comerciales, aeropuertos, … para ofrecer servicios de valor añadido a los clientes donde la proximidad es un factor importante.

Algunos ejemplos de los sectores en los que ya se está haciendo uso de esta tecnología son:

- **En comercio físico o retail:** los establecimientos pueden enviar campañas publicitarias, ofertas y descuentos a los usuarios próximos, e incluso permitir el pago.
- **En aeropuertos:** los usuarios pueden conocer cómo llegar a la puerta de embarque correspondiente o el tiempo del que disponen.
- **En museos:** conocer toda la información de las obras.
- **Eventos:** informar de cambios en festivales o ferias, …

- **Sanidad:** informar a las personas que se encuentran en la sala de espera, ayudarles a encontrar la salida o proporcionar información al médico sobre el paciente al que está visitando.

- **Turismo:** al situarse junto a distintos monumentos o edificios con sensores instalados se podrá obtener una breve explicación sobre la historia de los mismos u otros aspectos como los horarios de visitas o precios.

1.2. Objetivos del Trabajo Fin de Grado

La solución que aquí se propone consiste en desarrollar un sistema de check in automatizado para alojamientos turísticos haciendo uso simplemente de teléfonos inteligentes y los mencionados Beacons. Esta solución contendrá una aplicación móvil, que se instalará en el terminal de cliente y que, tras pasar distintos mecanismos de validación contra la nube, realizará el check in del usuario y le proporcionará una "llave virtual" del alojamiento, dando así total independencia tanto al cliente como al propietario durante todo el proceso.

Los objetivos concretos que se pretenden cubrir son los siguientes:

- Estudio de la tecnología Beacon para encontrar la opción que se adapte a nuestro objetivo.

- Diseñar e implementar una aplicación móvil que se conecte con el sistema con distintos roles: **instalador**, para añadir beacons al sistema y asignarlos a los alojamientos; **propietario**, que permitirá gestionar sus alojamientos y reservas; y, por último, **cliente**, que permite controlar la proximidad al beacon/alojamiento reservado y realizar el check in cuando se le esté permitido.

- Diseñar e implementar un sistema de control en la nube, para el rol de **administrador**, desde el cual gestionar todo el sistema y consultar los alojamientos, beacons, usuarios y reservas.

1.3. Las Tecnologías utilizadas:

1.3.1. iBeacon

iBeacon es un sistema de comunicación desarrollado por **Apple** que aprovecha la tecnología Bluetooth Low Energy (BLE) para remitir o recibir información hacia un dispositivo electrónico, generalmente un Smartphone o Tablet. Fue presentado por Apple en su conferencia anual para desarrolladores de 2013, la WWDC'13, y estuvo soportado a partir de iOS 7. Con esta tecnología se permite a las aplicaciones móviles iBeacon extender la posición del dispositivo en una escala micro-local y entregar algún contenido diferencial en función del usuario y de su ubicación, beneficiándose también de un consumo ínfimo, ya que los sensores pueden funcionar hasta 3 años con una simple pila de botón e integran un acelerómetro, memoria Flash, procesador ARM y la citada conectividad Bluetooth.

Aunque va más allá, iBeacon puede ser considerado como un sistema de posicionamiento en interiores (IPS, Indoor Postioning System), al poder transmitir un identificador único universal que será recogido por una aplicación compatible y que puede ser convertido en una localización física o generar una acción en el dispositivo.

Como ya hemos dicho, la tecnología de comunicación subyacente es Bluetooth Low Enerygy (BLE), también conocido como Bluetooth Smart y que, como su nombre indica, está diseñado para bajo consumo de energía y costes, manteniendo un rango de comunicación similar a la de su predecesor, Bluetooth Classic. Así pues, las diferencias del BLE con el Bluetooth tradicional serían fundamentalmente tres:

- Consumo de energía: Una baliza Bluetooth contiene una batería que podría durar hasta 3 años con una sola pila de botón.
- Bajo Coste: BLE es un 60-80% más barato que el Bluetooth tradicional.
- Aplicación: BLE es ideal para aplicaciones sencillas que requieren pequeñas transferencias periódicas de datos, mientras que el tradicional resulta óptimo

para aplicaciones más complejas que requieran una comunicación constante y más rendimiento de datos.

Frente a otras tecnologías, como la NFC, a la que muchos consideran competidora directa, **iBeacons** ofrece la ventaja de que su alcance puede llegar en algunos casos hasta 100 metros (dependiendo del fabricante), lo que permite que el usuario no tenga que "pasar" su terminal por un lector para recibir la información. Por eso, y a pesar de que el entorno de NFC es muy diferente y que dispone de muchas aplicaciones no solapables, todavía se compara con iBeacons, y es interesante enumerar algunos puntos diferenciadores entre ambas tecnologías:

- El alcance de NFC es de hasta 20cm (7.87 pulgadas) pero el alcance óptimo es menor de 4cm (1.57 pulgadas). Como ya se ha dicho, los iBeacons tienen un rango significativamente mayor.

- No todos los teléfonos llevan chips NFC. (solo algunos de los últimos modelos de dispositivos iOS lo integran, si bien no se soportan aplicativos de terceros), pero la mayoría de los teléfonos tiene capacidad Bluetooth (la mayoría de los más modernos con la versión 4.0).

- Con la tecnología de aprovechamiento de energía de la batería de gasto cero en los dispositivos fabricados por Passif (el fabricante de chips de bajo consumo adquirido por Apple) hay menos uso de batería que con NFC.

1.3.2. Cocoa y el Lenguaje de Programación Objective C

Objective C es un lenguaje de programación orientado a objetos creado como un superconjunto de C y que nació en la década de los 80. Actualmente se usa como lenguaje principal, de programación para Mac OSX e iOS, además de Swift.

Cocoa, es la API de programación orientada a objetos de Mac OS e iOS. Es accesible desde Java, Python, Objective C y Swift, siendo los dos últimos los lenguajes con los que se puede obtener más rendimiento. Esta API tradicionalmente se ha dividido en dos grandes grupos de clases:

- **Foundation Framework**: un conjunto de clases de apoyo que permiten representar estructuras de datos complejas (arrays, listas, diccionarios, …) y dentro de las que no se incluyen clases relacionadas con la interfaz gráfica. Esta librería es común a Mac OS e iOS.

- **Application Kit** Framework (AppKit Framework): Aquí se incluyen casi todas las demás clases. Como puedan ser las relacionadas con la interfaz gráfica, la impresión, el acceso a audio y vídeo, el acceso a red, y todos los demás aspectos que podemos gestionar desde Cocoa. Dentro del AppKit encontramos otros kits de desarrollo especializados en determinadas tareas como puedan ser: Image Kit, QuickTime Kit, Apple Scripting Kit.

En la siguiente figura [Fig. 1] se muestra un resumen de las capas de software de programación del sistema operativo Mac OS X. Podemos ver que la principal característica de las librerías que se agrupan bajo el nombre de Cocoa es que son accesible desde Objective-C aunque luego existen librerías C que proporcionan servicios a más bajo nivel. Por ejemplo, Core Foundation envuelve en funciones C las operaciones del Foundation Framework.

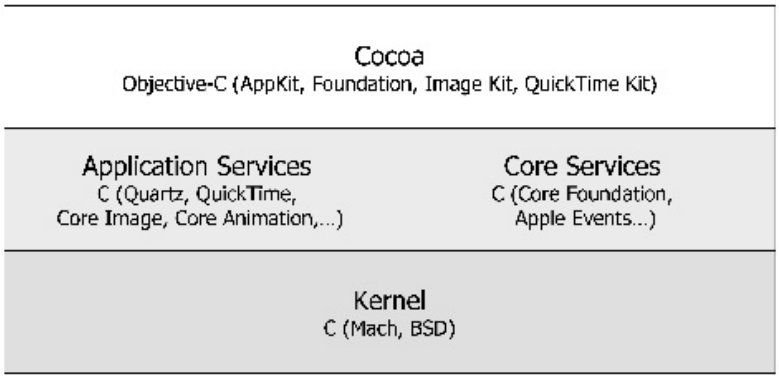


Figura 1. Capas de software de programación en Mac OS X e iOS.

1.3.3. Firebase

Firebase es la plataforma de desarrollo móvil en la nube de Google, disponible para diferentes plataformas (Android, iOS y web).

En definitiva, se trata de un servicio con una gran variedad de funcionalidades [Fig. 2] capaz de proveernos un Backend en la nube con una fuente de datos sincronizada en tiempo real y unas librerías para acceder a ella desde nuestras aplicaciones.

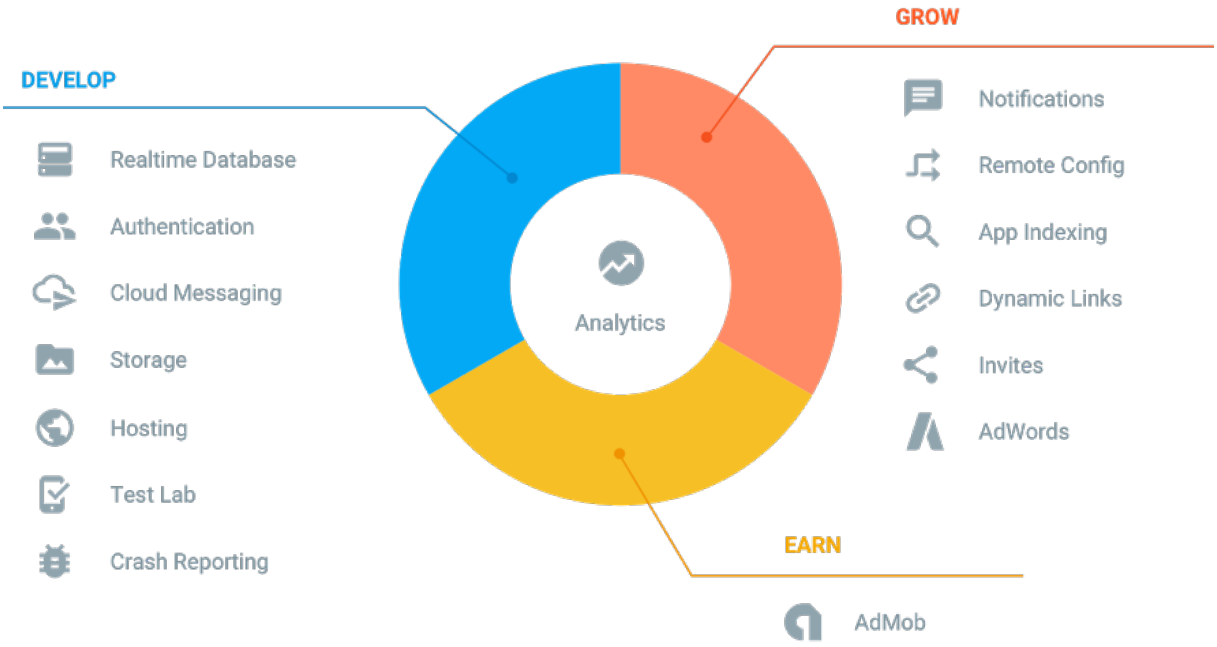


Figura 2. Funcionalidades de Firebase

Las principales características de Firebase pueden ser agrupadas en las siguientes secciones:

- **Desarrollo:** Firebase permite la construcción de mejores aplicaciones, minimizando el tiempo de optimización y desarrollo, mediante diferentes funciones, entre las que destacan la detección de errores y testeo, lo que supone un salto de calidad en la aplicación. Poder almacenar toda la información en la nube, testear la app o poder configurarla de manera remota,

son características destacables de la plataforma. Otras características destacadas con la inclusión de mensajería en la nube, autenticación o reporte de crash.

- **Analítica:** proporciona, si se desea, un control máximo del rendimiento de la aplicación mediante métricas analíticas, todo desde un único panel.
- **Poder de crecimiento**: Permite gestionar los usuarios de las aplicaciones, con el añadido de poder captar nuevos mediante funcionalidades como invitaciones, indexación o notificaciones.
- **Monetización**: con la herramienta AdMob.
- **Rapidez**: Implementar Firebase puede ser fácil y rápido, gracias a su API intuitiva, sostenida en un solo SDK. Así, dependiendo de la app, se puede evitar la pérdida de tiempo en la creación de una infraestructura demasiado compleja.
- **Agilidad:** ofrece aplicaciones multiplataforma con APIs integradas a SDK individuales para iOS, Android y JavaScript, de tal forma que se pueden gestionar diferentes aplicaciones sin necesidad de salir de la propia plataforma.

1.3.3.1. Seguridad en Firebase

Firebase ofrece un sistema sencillo para determinar las reglas de seguridad de una aplicación de cara a permitir accesos y proteger datos. En este sentido, distinguimos dos puntos a destacar en Firebase: la Autenticación y la Autorización.

A. Autenticación

El reconocimiento de la identidad de un usuario permite a una aplicación guardar datos de este de manera segura en la nube y brindar la misma experiencia personalizada en todos sus dispositivos.

Firebase Authentication proporciona servicios de backend, SDK fáciles de usar y bibliotecas de IU ya hechas para autenticar usuarios en tu app. Admite autenticación con contraseñas, proveedores de identidades federadas populares, como Google, Facebook y Twitter, y más opciones.

Firebase Authentication se integra estrechamente con otros servicios de Firebase y aprovecha estándares industriales como OAuth 2.0 y OpenID Connect. Por lo tanto, se puede integrar fácilmente con nuestro backend personalizado.

B. Autorización. Reglas y seguridad en Bases de Datos.

La Firebase Realtime Database proporciona un conjunto completo de herramientas para administrar la seguridad de una app.

La identificación de los usuarios solo es un aspecto de la seguridad. Una vez que se saben quiénes son éstos, es necesaria una manera de controlar su acceso a los datos en tu base de datos. En este contexto entran en juego las Firebase Database Rules, que permiten controlar el acceso para cada usuario.

Existe un sistema sencillo para escribir las reglas de seguridad de una aplicación, por medio de un archivo de texto que tiene notación JSON. Las Firebase Realtime Database Rules determinan quién tiene acceso de lectura y escritura a tu base de datos, cómo se estructuran tus datos y qué índices existen. Estas reglas se alojan en los servidores de Firebase y se aplican de manera automática en todo momento. Cada solicitud de lectura y escritura solo se completará si lo permiten tus reglas.

Las Firebase Database Rules tienen una sintaxis de tipo JavaScript y existen cuatro clases de ellas:

- **.read**: Indica si se permite la lectura de datos por parte de usuarios y el momento para ello.
- **.write**: Indica si se permite la escritura de datos y el momento para ello.

- **.validate**: Define el aspecto de un valor con formato correcto, si este tiene atributos de campo secundario, y el tipo de datos.
- **.indexOn**: Especifica un hijo que debe indexarse para admitir el orden y las consultas.

Por otro lado, es interesante mencionar que la Firebase Realtime Database implementa otros mecanismos de seguridad en segundo plano. Por ejemplo, se usa SSL con claves de 2048 bits para los certificados y se aplican las prácticas recomendadas para los tokens de autenticación.

1.3.4. Html, JavaScript y Bootstrap

HTML, HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros.

JavaScript, es un lenguaje de programación interpretado que se puede definir como orientado a objetos basado en prototipos y que se utiliza principalmente para crear páginas web dinámicas.

Bootstrap es un framework o conjunto de herramientas de código abierto, originalmente creado por Twitter, para el diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript opcionales adicionales. Su principal virtud, es que nos permite crear de forma sencilla webs de diseño adaptable, es decir, que se ajustan a cualquier dispositivo y tamaño de pantalla, lo que se conoce como “Responsive design”.

1.3.5. Otras herramientas, técnicas y filosofías aplicadas

Por último, hay que mencionar que durante el proceso de desarrollo del trabajo se han aplicado técnicas de desarrollo Ágil, Bitbucket como repositorio del código y herramientas que facilitan el ciclo desarrollo como Source Tree y el método Git-Flow.

2. Beacons en el entorno iOS

En el apartado donde se habla de las Tecnologías que se han usado para el desarrollo del proyecto ya hemos mencionado el uso de iBeacons. Pero vamos a ver las distintas posibilidades existentes, por qué nos hemos decantado por iBeacon como la mejor opción para nuestras necesidades y qué necesitamos saber para que sea integrado en una aplicación.

Existen distintos protocolos Beacon que puede ser usados en entornos iOS, vamos a hacer un pequeño resumen de los más populares.

2.1. AltBeacon

Este protocolo ha sido desarrollado por Radius Networks y define el formato de los mensajes broadcast por proximidad de los beacons. Propone dos ideas fundamentales: definir una especificación resistente para la transmisión al dispositivo (mensajes enviados por la baliza) y un conjunto de herramientas y referencias que haga más sencillo a los desarrolladores de apps recibir el mensaje de los beacons.

Altbeacon tiene 25 a 28 bytes disponibles (MFG ID, BeaconCode, BeaconID, MFG RSVD) para datos de usuario, ligeramente superior a otros protocolos.

Está lanzado bajo una licencia de Creative Commons que permite compartir y adaptar el contenido. Viene a ser una respuesta al protocolo cerrado iBeacon propiedad de Apple, cubriendo las mismas funcionalidades que iBeacon puede ofrecer, pero no tiene un amplio apoyo aún y éste es uno de sus principales inconvenientes.

2.2. Eddystone

Es el proyecto de código abierto de Google para beacons, con el que pretende fomentar el internet de las cosas. Similar al protocolo de iBeacon pero open source. Eddystone tiene soporte oficial para iOS y Android.

Google además proporcionará las API de Nearby y Proximity para ayudar a los desarrolladores en como transmitir datos a equipos ubicados en el rango de los beacons seleccionados, a la vez que les permite monitorear los beacons.

Está diseñado para soportar múltiples tipos de paquetes de datos:

- **Eddystone-URL**: un único campo que será una URL.
- **Eddystone-UID**: un identificador único del beacon.
- **Eddystone-TLM**: paquete de datos de telemetría, se envía junto con cualquiera del resto de paquetes de datos y contiene datos sobre el estado de salud del beacon como, por ejemplo, la duración de la batería. También contiene información sobre diferentes condiciones, dependiendo de los sensores del dispositivo, tales como la temperatura, la humedad, …
- **Eddystone-EID**: emite un identificador cambiante, denominado EIK (Ephemeral Identity Key), que lo hace más seguro y que se resuelve para identificar el beacon usando alguna otra API como por ejemplo la Proximity Beacon API.

Eddystone se basa en un método único en este momento: Eddystone Discovery que es similar a iBeacon Ranging. Proporciona estimaciones de proximidad y solo funciona cuando está activa.

Eddystone es bastante flexible, pero requiere una implementación más compleja en cuanto a integración. Eso, junto con el hecho de que sea necesaria una aplicación plenamente activa para garantizar la detección de los beacons, no lo hace fiable para alcanzar los requisitos de nuestra solución. Todas las fuentes consultadas y pruebas realizadas nos han hecho llegar a esta conclusión y la realidad es que no es efectivo a la hora de localizar beacons con aplicaciones en segundo plano en entornos iOS, donde el sistema puede llegar a matar aplicaciones y esto hace que obtengamos unos pobres y poco fiables resultados. Por no hablar de la detección con la aplicación totalmente cerrada, donde no es posible por definición.

2.3. iBeacon

Es el protocolo que hemos seleccionado para trabajar en nuestra solución y, a pesar de ser nativo de iOS, es compatible con Android.

Hay que decir que no existe una API de iOS como tal exclusiva para iBeacons, sino que para trabajar con ellos se utilizan los frameworks de Bluetooth y, a diferencia de los otros protocolos, la API de CoreLocation. Y es este último el factor diferencial que permite detectar los iBeacons con la aplicación no activa.

El protocolo iBeacon emite solo un tipo de paquete de datos con un único identificador que pueden ser usados para identificar la posición física del dispositivo. Contiene los siguientes elementos:

- **UUID** (Universally Unique Identifier): que contiene 32 dígitos hexadecimales dividido en 5 grupos. Se utiliza para identificar un grupo de balizas relacionadas entre sí.

- **Major**: Una cadena de 2 bytes usado para identificar un subconjunto de beacons dentro de un grupo mayor.

- **Minor**: Otra cadena de 2 bytes que identifica beacons de forma individual.

- **Tx Power**: Usado para determinar la proximidad del beacon. Se utiliza con la medida de la intensidad para determinar a qué distancia se encuentra el beacon del dispositivo móvil (smartphone). Este campo, si queremos usarlo con sentido y fiabilidad, debería ser calibrado beacon por beacon por el usuario para ser exacto.

La aplicación de escaneo leerá el UUID, Major, Minor y los referenciará contra una base de datos para obtener información sobre el beacon, el propio beacon no lleva ninguna información descriptiva y requiere de una base de datos externa para ser útil. Ésta será descrita en capítulos sucesivos del presente documento.

Hay otro concepto que debemos tener en cuenta y son las **Regiones Beacon** que no son sino la representación física de un conjunto de todos los beacons en una región. Éstas se pueden definir de tres maneras:

- **Solo con el UUID**: se refiere a todos los beacons que tenga un UUID.

- **Con UUID y Major**: contendrá todos los beacons que usan una combinación de UUID y Major.

- **Con UUID, Major y Minor**: será un único beacon, no se pueden definir dos beacons con ésta misma combinación.

iBeacon ofrece dos métodos para detectar dispositivos ibeacons, **Ranging y Monitoring**:

- **Region Ranging**: solo funciona cuando la aplicación está activada y proporciona estimaciones de proximidad para la región sobre la que estemos realizando este rastreo.

- **Region Monitoring**: funciona incluso si la aplicación no está corriendo. Este ha sido el punto determinante para adoptar este protocolo para nuestro proyecto. Con esta monitorización se recibirán notificaciones en el sistema en base a la entrada o salida del dispositivo escuchante con respecto a una región.

  Una limitación que tenemos que mencionar en este sistema es que, a pesar de no haber prácticamente limite en el número de beacons que pueden ser incluidos en el sistema (técnicamente serían casi 4 billones), iOS sí establece un límite de 20 beacons como el máximo que se pueden monitorear.

3. Análisis y diseño de la solución

En esta sección, una vez vistas las tecnologías actuales aplicables a la idea que se pretende implementar, vamos a pasar a describir la solución completa [Fig. 3]. También veremos algunos de los elementos comunes a la aplicación antes de pasar a ver como quedarían las aplicaciones que se han de desarrollar para que se cumplan los objetivos.

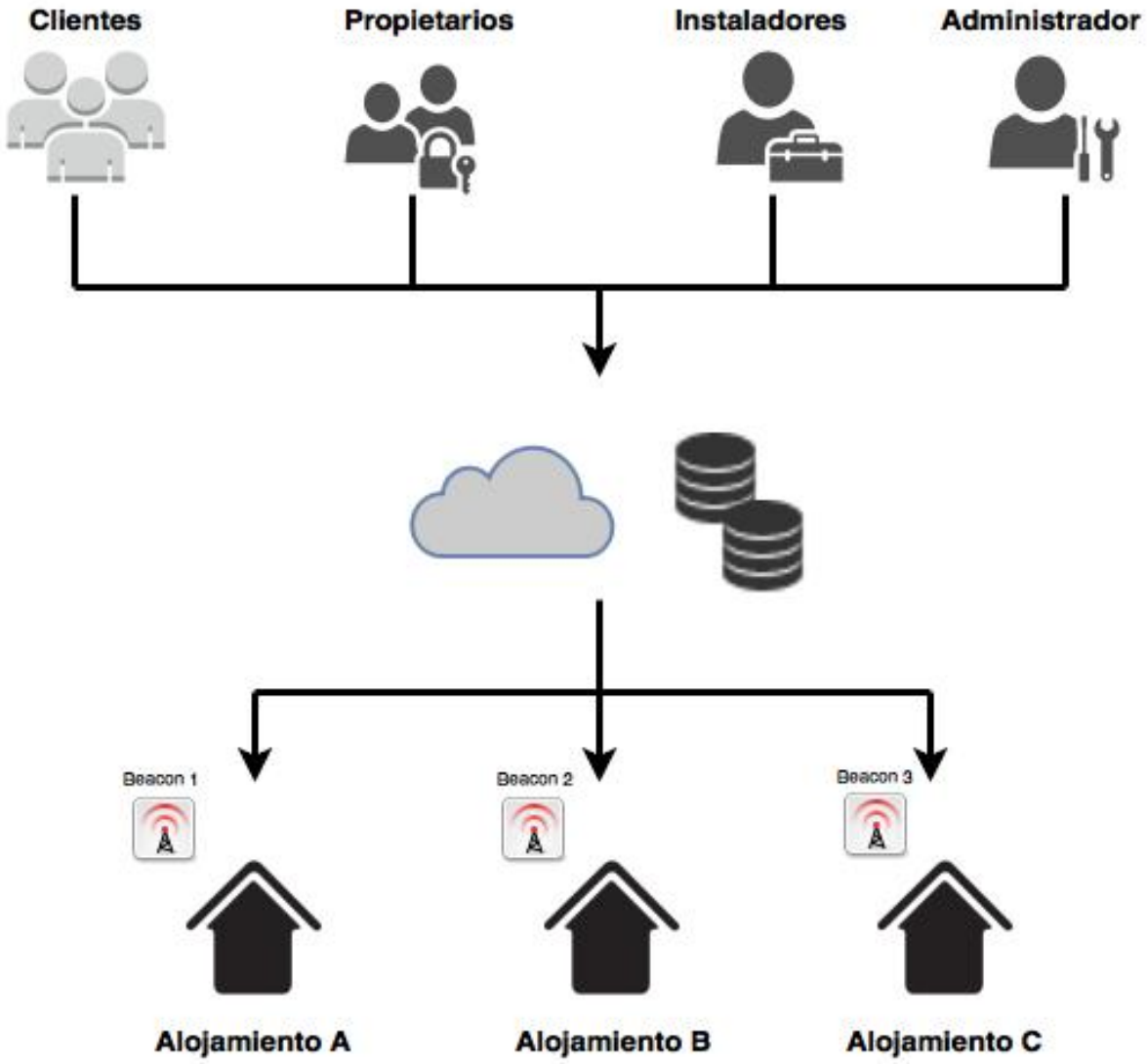


Figura 3. Diagrama de la solución completa

La solución propuesta, y objetivo del proyecto, como ya hemos mencionado anteriormente, consiste en desarrollar un sistema de check in automatizado para alojamientos turísticos haciendo uso simplemente de teléfonos inteligentes y los

mencionados iBeacon. Se tratará de una solución móvil, por lo cual, además de un backend donde tendremos todos los datos de nuestra aplicación, contendrá una aplicación móvil, que se instalará en el terminal del cliente y recuperará los datos para validar y realizar el trámite de check in de la forma más sencilla posible. En ella, cuando el usuario esté cerca del alojamiento que tiene en reserva (detectado por el sistema de beacons) y una vez pasado todo el mecanismo de seguridad y verificación del check in, se obtendrá una "llave" virtual del alojamiento, en lugar de la llave física, la cual permitiría a ese usuario acceder directamente a la habitación o alojamiento quedando registrado cuando se accede por primera vez, y lo que es más importante, dando total independencia al cliente y al propietario en todo el proceso y en su futura estancia.

Además del acceso para el usuario/cliente del alojamiento, deberemos contemplar la existencia de otros tipos de usuario de la aplicación como lo pueden ser el propietario del alojamiento, al que pretendemos facilitarle la gestión de reservas de sus alojamientos y el control de acceso.

Por último, otro punto que queremos cubrir con esta solución es la fase de instalación de los beacons en los alojamientos, lo que sería el alta del beacon en el sistema. Así pues, daremos cabida a la inclusión de nuevos beacons en el sistema, su asignación a alojamientos y la lógica de rastreo de beacons.

3.1. Diagramas de secuencia para un uso genérico

Veamos algunos diagramas de secuencia que nos ayudaran a entender los pasos que se llevan a cabo en algunos de estos procesos fundamentales del sistema:

A. **Alta de alojamiento y su beacon en el sistema.** [Fig. 4]

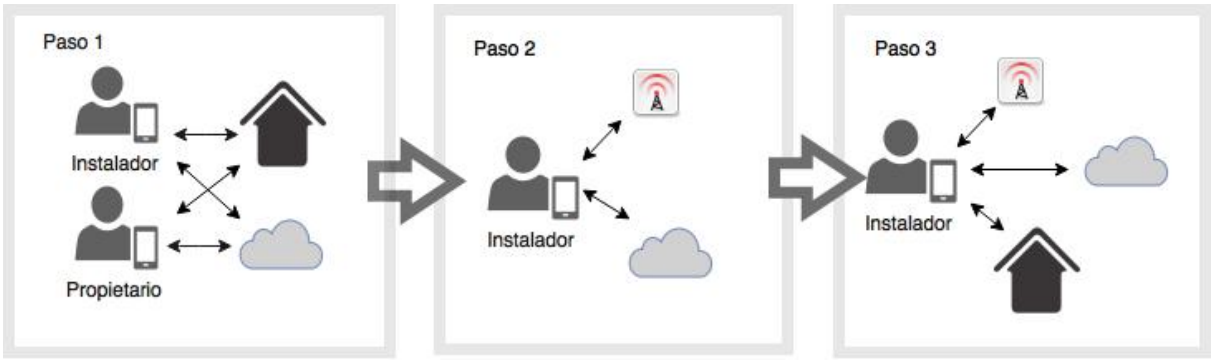


Figura 4. Diagrama de secuencia de alta de alojamiento en el sistema

Este sería el esquema del proceso completo de alta de alojamiento y beacon en el sistema. No obstante, tanto los alojamientos, como los beacons, pueden ser dados de alta de forma independiente en cualquier momento desde la aplicación.

En primer lugar, el propietario o el instalador introducen todos los datos del alojamiento en la aplicación, que serán comunicados al backend. A continuación, **paso 2**, el instalador da de alta el beacon en el sistema, aproximándose con la aplicación móvil al mismo, obteniendo sus datos y comunicándolos al sistema. Y, en último lugar, tal y como vemos en la imagen del **paso 3**, el instalador también asigna el beacon al alojamiento en cuestión.

B. **Gestión de reserva y llave por parte del propietario**. [Fig. 5]

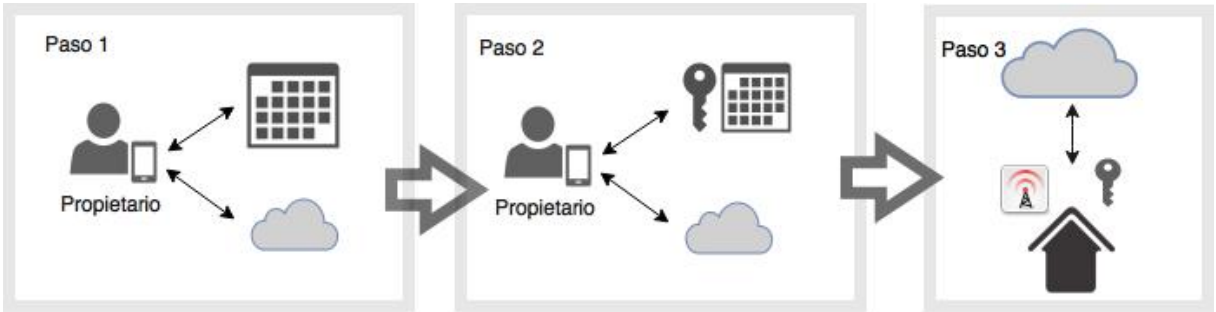


Figura 5. Diagrama de secuencia gestión de reservas del propietario

Aquí vemos el diagrama para la gestión de reserva y llave por parte del propietario. Nos sirve para reconocer cómo el propietario es el que tiene la

última palabra en el acceso a su alojamiento, ya que es él el que proporciona una llave para la reserva que ha sido efectuada en su alojamiento.

En el **paso 1**, el propietario accede a los datos de reserva de sus alojamientos en el sistema y puede modificar los datos de las mismas, incluida la llave de acceso asignada, como vemos en la imagen del **paso 2**. Esa llave se guardará en el sistema a través de los servicios de Firebase y quedará vinculada a la reserva del alojamiento y por ende al beacon para permitir la entrada o no al sistema, **paso 3**.

C. Acceso del cliente al alojamiento. [Fig. 6]

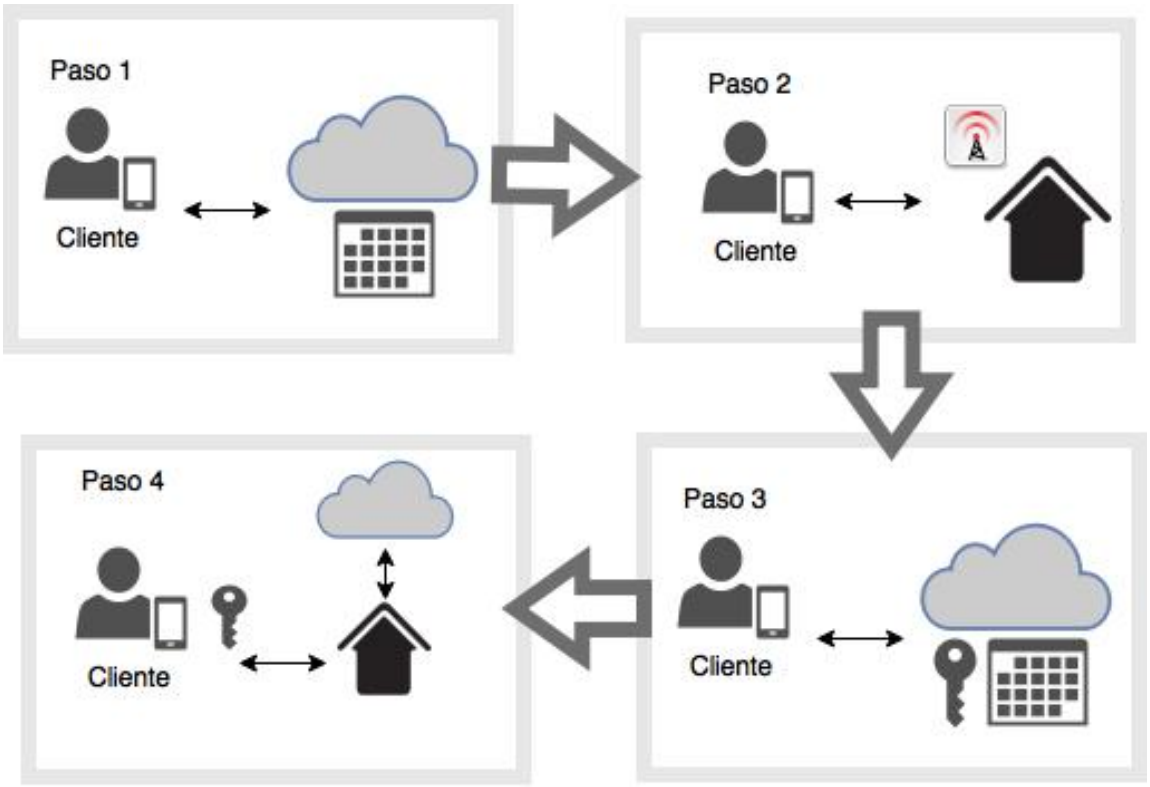


Figura 6. Diagrama de secuencia para el acceso del cliente al alojamiento

En el **paso 1**, el cliente accede desde la aplicación móvil a los datos de la reserva. Cuando se aproxima al alojamiento, **paso 2**, recibe una notificación que indica que está cercano a un alojamiento del que posee una reserva activa. En

el **paso 3** el cliente puede solicitar la llave que tiene asignada para la reserva a través de los servicios de backend que se ofrecen. Evidente, solo se recibirá la llave si se pasan todos los mecanismos de validación. Si todo es correcto, el cliente estará en posesión de la llave y en el **paso 4** podrá introducir esa llave en el dispositivo de apertura, que verificará si la llave introducida es correcta y procederá a abrir la puerta.

3.2. Roles

Para dar cobertura a todas las necesidades expuestas por el objetivo del proyecto, vamos a definir una serie de roles en la aplicación, que ya se intuyen tras las explicaciones anteriores. Estos roles de usuarios, controlados por el sistema de autenticación del backend (Firebase) y por la lógica definida en nuestra aplicación, dará acceso al usuario solo a los contenidos y funcionalidades permitidos para el mismo. Así pues, los roles de los que dispone la aplicación son:

- **Administrador:** será el usuario que tiene control y acceso sobre todos los datos de la aplicación. Este usuario tendrá acceso solo a la aplicación web que veremos más adelante.

- **Cliente:** es el usuario que ha efectuado una reserva y que pretende acceder a un alojamiento. Tendrá acceso a una versión la aplicación móvil, cuando esté cerca del alojamiento en el que tiene efectuada una reserva, recibirá una notificación (emitida por el beacon) que iniciará el proceso de recuperación de llave para ese alojamiento y reserva.

- **Propietario:** es el dueño del alojamiento en el que se pueden efectuar las reservas. Tiene acceso a la aplicación móvil, en la que podrá gestionar sus alojamientos, reservas y las llaves de acceso para las mismas.

- **Instalador:** tendrá acceso a una versión de la aplicación móvil en la que podrá gestionar los beacons del sistema y asignarlos a los alojamientos.

3.3. Diagramas de casos de uso

3.3.1. Diagrama casos de uso del administrador

En la siguiente imagen [Fig. 7] vemos el diagrama de casos de uso para la aplicación web, que como ya hemos dicho solo permite el acceso a los usuarios con rol Administración.

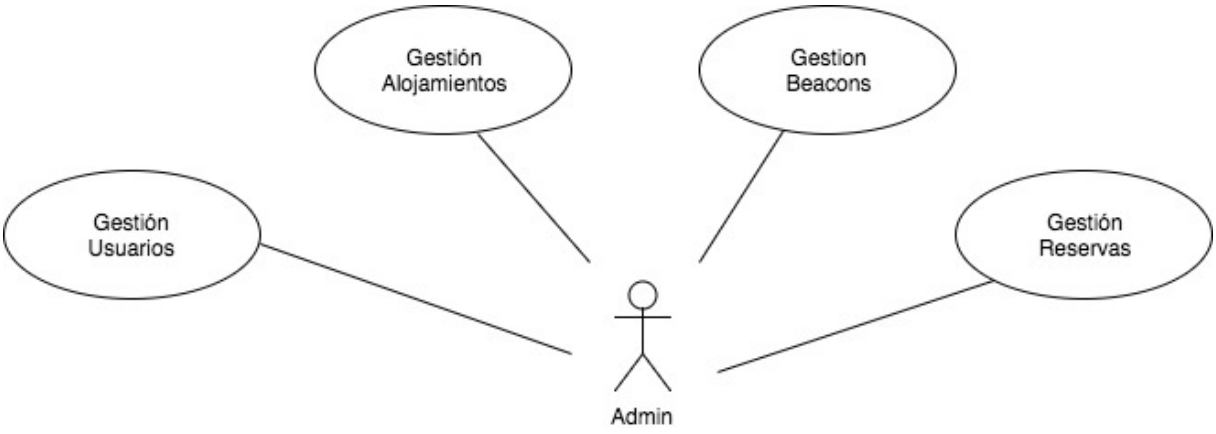


Figura 7. Casos de uso de la Aplicación web.

Como podemos observar, el administrador se encargará de la gestión de todos los datos de la aplicación, que quedan divididos en Usuarios, Alojamientos, Beacons y Reservas.

3.3.1. Diagrama de casos de uso del cliente

En realidad, las únicas acciones que puede forzar el cliente desde su aplicación serán: la consulta de sus reservas y la petición de la llave de una reserva siempre que se cumplan una serie de requisitos como lo son: que se trate del alojamiento de una reserva activa en el tiempo y que se encuentre próximo al alojamiento o, lo que es lo mismo, al beacon que tiene instalado el mismo. [Fig. 8]

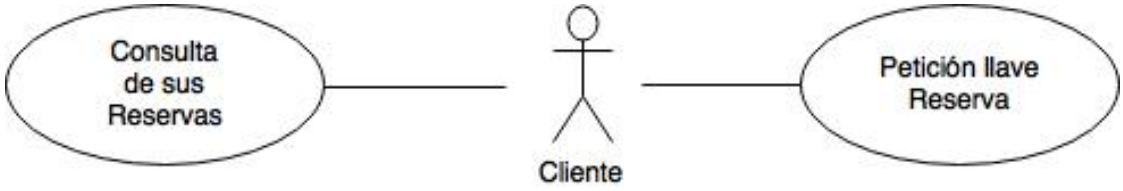


Figura 8. Diagrama de Casos de uso del Cliente

3.3.2. Diagrama de casos de uso del propietario

Este es el diagrama de casos de uso para el usuario con rol propietario, básicamente tiene control sobre sus alojamientos y las reservas de estos. [Fig. 9].

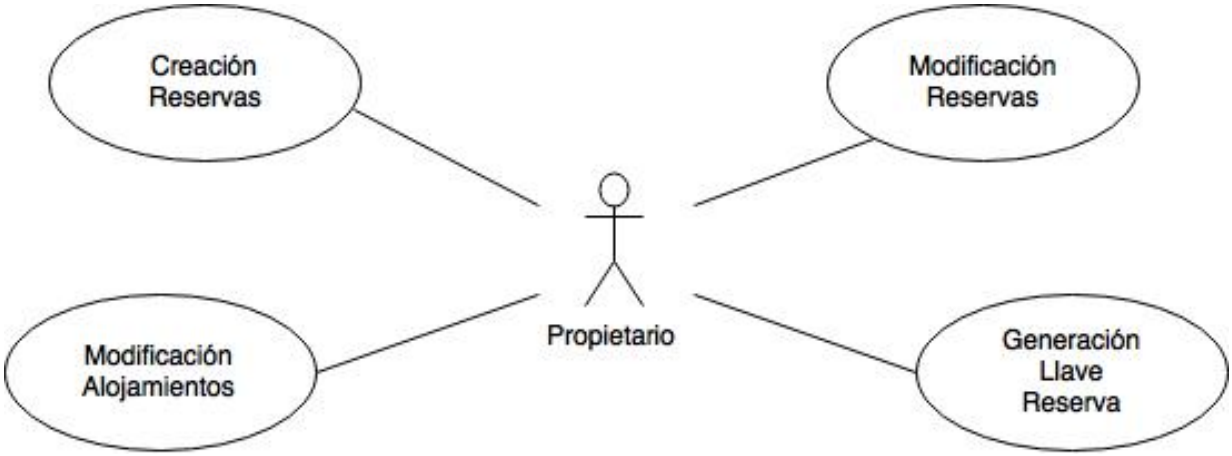


Figura 9. Diagrama de Casos de Uso del Propietario

3.3.3. Diagrama de casos de uso del instalador

En la siguiente figura [Fig. 10] vemos todos los casos de uso para el instalador, que necesita realizar todas las acciones relativas los alojamientos, los iBeacons y la vinculación de iBeacons y alojamientos en el sistema.

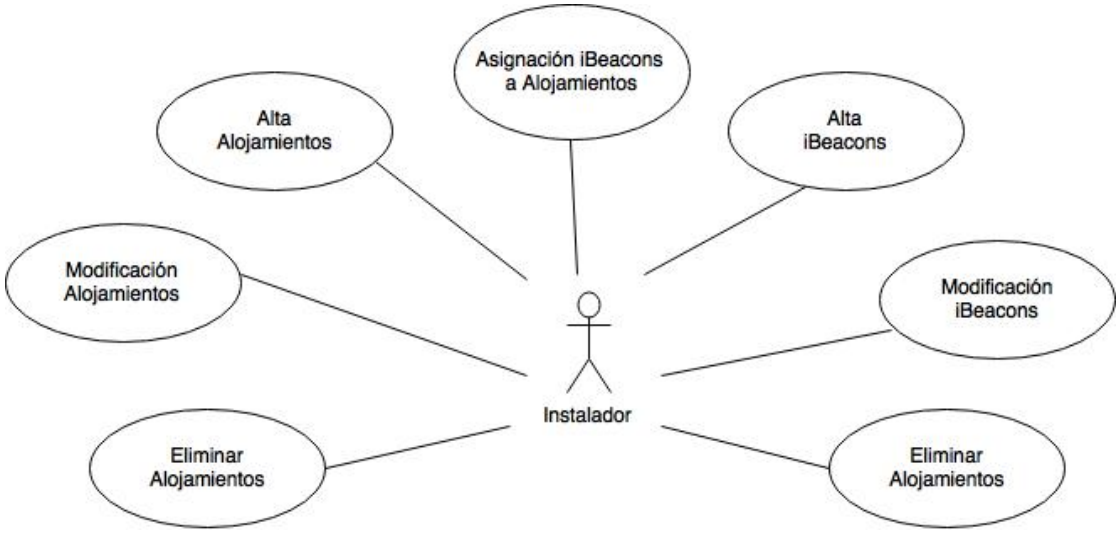


Figura 10. Diagrama de Casos de Uso del Instalador

3.4. La base de datos

El backend de todo el sistema está montado con Firebase, del que ya hemos hablado. Firebase provee una consola desde web donde poder administrar nuestro proyecto. [Fig. 11]

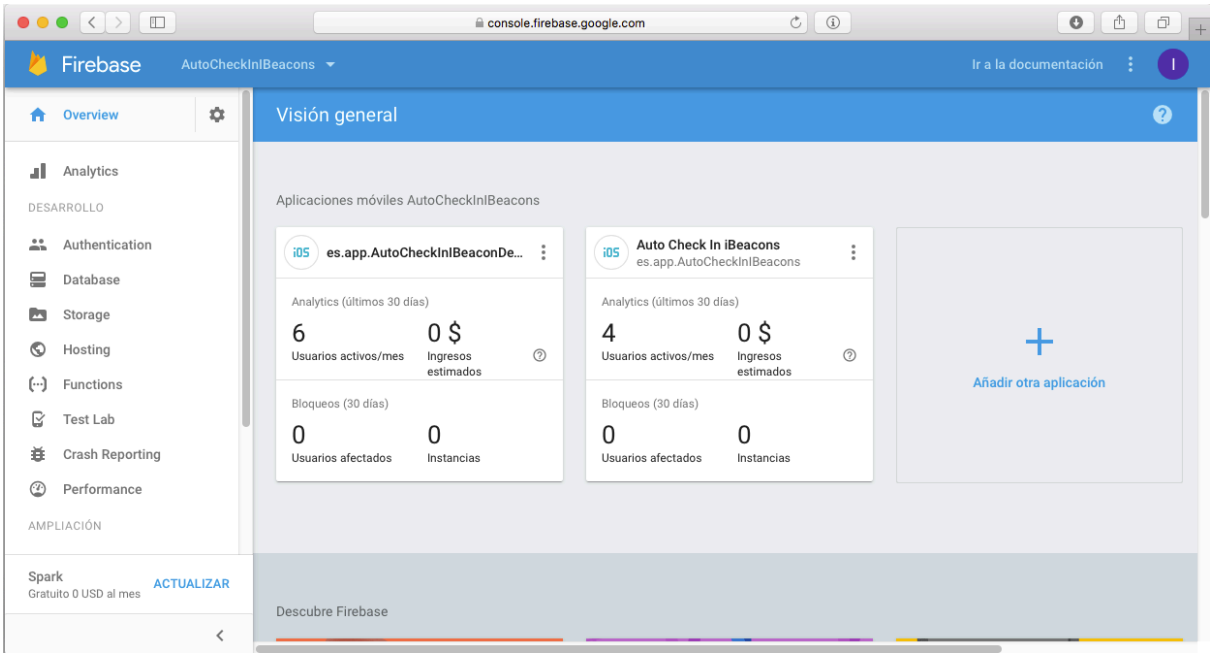


Figura 11. Consola de Firebase

En la imagen anterior podemos observar todos los servicios que nos provee la plataforma, entre ellos la Base de Datos en Tiempo Real, el que nos ocupa en esta sección.

Todos los datos de Firebase Realtime Database se almacenan como objetos JSON. Se puede pensar en la base de datos como un árbol JSON alojado en la nube. A diferencia de la base de datos SQL, no existen tablas ni registros. Cuando se agregan datos al árbol JSON, estos se convierten en un nodo en la estructura JSON existente.

Aun así, a la hora de definirla hemos realizado un diagrama entidad relación que será el que se aplica de forma lógica a la hora de crear y añadir nuevos registros a la misma. Podemos ver este diagrama en la siguiente figura [Fig. 12]

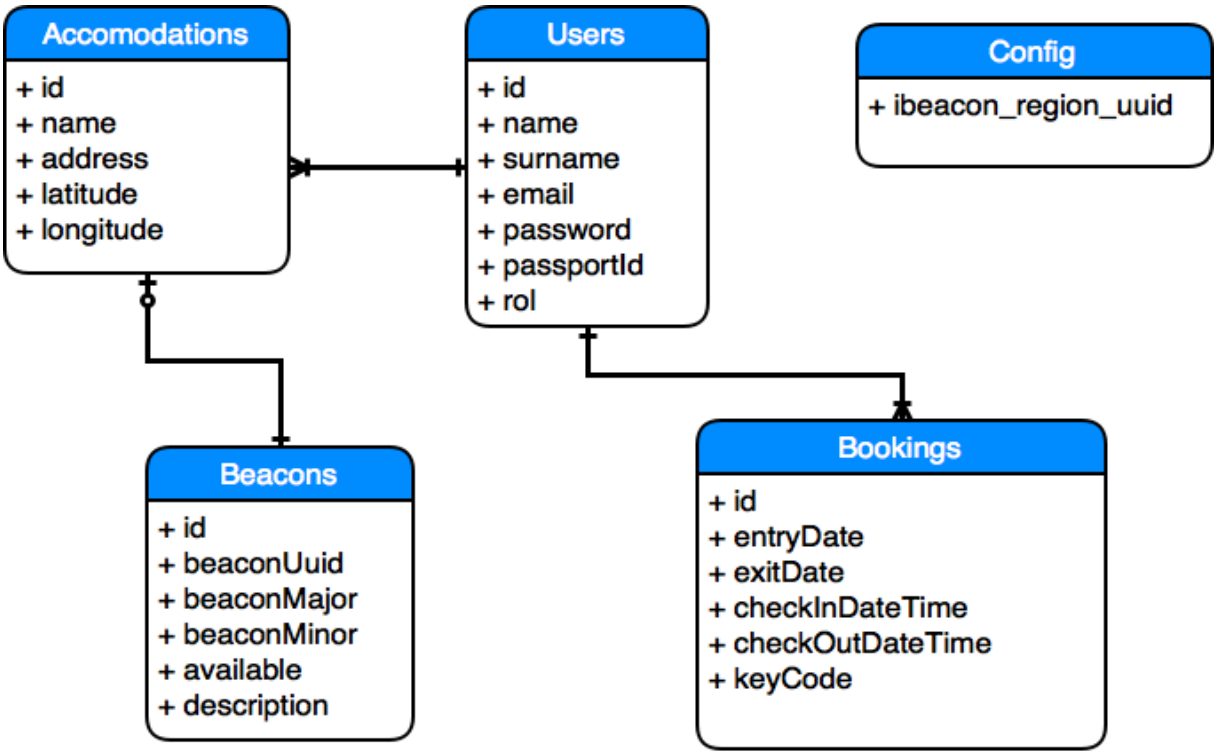


Figura 12. Diagrama del Modelo

Como se puede observar se ha definido un modelo bastante sencillo pero que abarca todas las necesidades de la solución que vamos a implementar. Pero como

hemos dicho, esto será almacenado en Firebase como fichero JSON, por lo que quedaría una estructura real con la siguiente forma: [Fig. 13] y [Fig. 14]

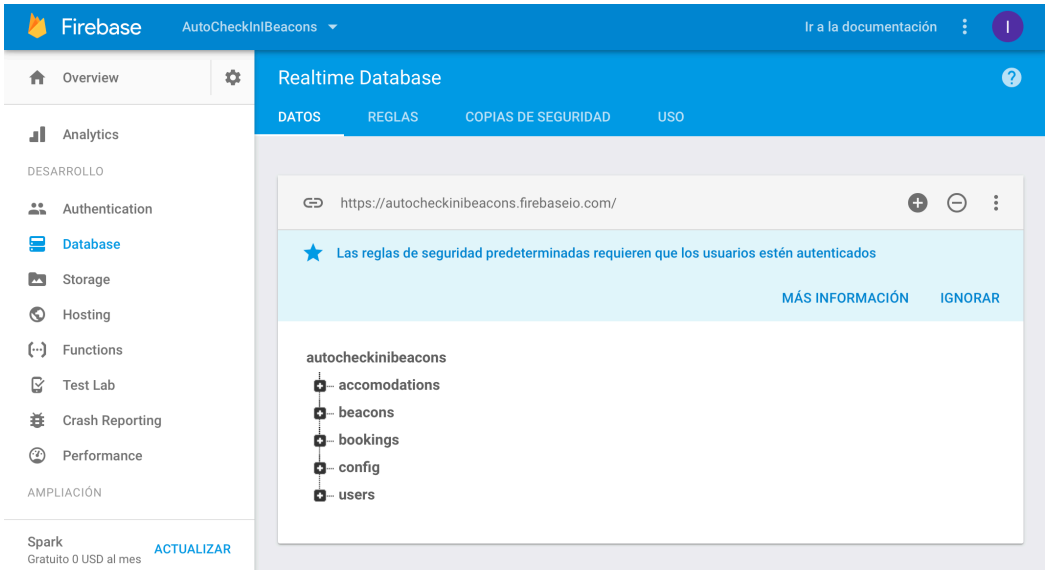


Figura 13. Estructura de datos en Firebase

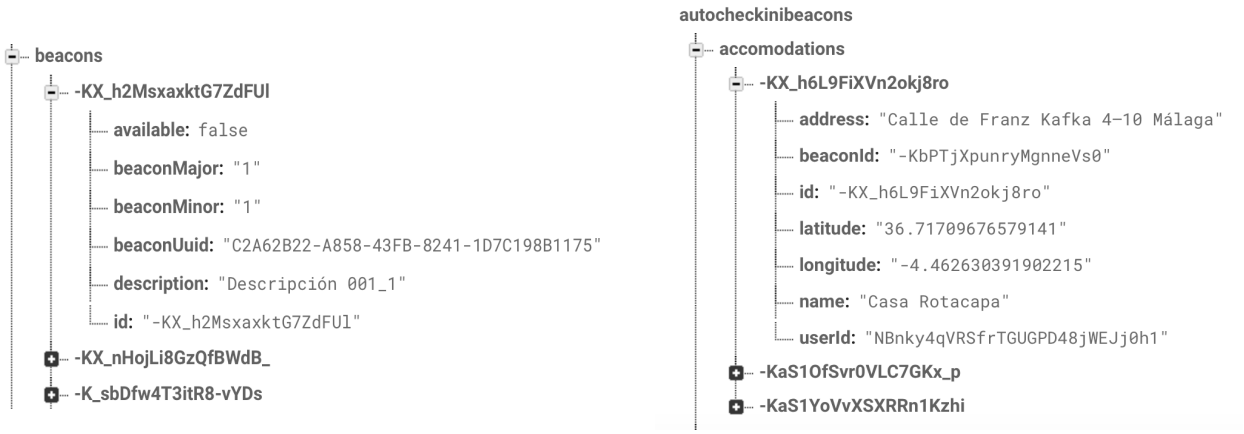


Figura 14. Ejemplos estructura de datos para Beacons y Alojamientos en base de datos

4. Aplicación web

A groso modo podemos decir que la aplicación web que forma parte de la solución aquí presentada es la que permite la gestión total de toda la información y relaciones que contiene el sistema. Esta aplicación será solo accesible por los usuarios con rol Administrador.

Como ya se especificó en capítulos anteriores, la aplicación web se ha desarrollado con Html, JavaScript y Bootstrap. El acceso a la Base de Datos de Firebase se realiza mediante el SDK de JavaScript que proporciona Firebase. De cara a su inicialización con toda la seguridad que Firebase incluye, es necesario el siguiente código de inicialización en la aplicación en el que se setean una serie de parámetros, y para lo que necesitamos unos valores que se encuentran en la consola Firebase de nuestro proyecto. El código quedaría así: [Fig. 15]

```
// Initialize Firebase
var config = {
    apiKey: "AIzaSyAl5UsYgWMygkXzmG8Vf29XMQXQM1_pFcE",
    authDomain: "autocheckinibeacons.firebaseapp.com",
    databaseURL: "https://autocheckinibeacons.firebaseio.com",
    storageBucket: "autocheckinibeacons.appspot.com",
    messagingSenderId: "915004550307"
};
firebase.initializeApp(config);
```

Figura 15. Código de inicialización de Firebase en la aplicación web.

4.1. Interfaz entorno administración

La idea del interfaz a desarrollar es hacer un entorno lo más sencillo posible, ya que consideramos que el núcleo de la solución es la parte móvil y preferimos dedicar más recursos a ello. A continuación, se detallan las distintas secciones del mismo:

4.1.1. Login

La web comienza en una pantalla de login [Fig. 16] donde el usuario podrá introducir sus credenciales.

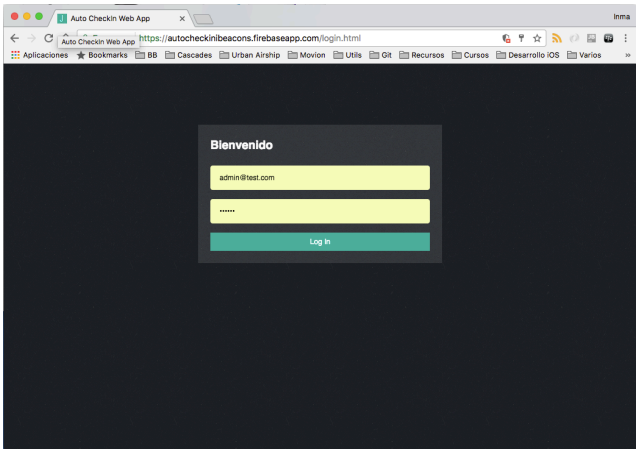


Figura 16. Pantalla de Login

Esta autenticación se hace accediendo usando Firebase Authentication como podemos ver en el siguiente código: [Fig. 17]

```
// Sign in with email and pass.
firebase.auth().signInWithEmailAndPassword(email,
password).catch(function(error) {
  // Handle Errors here.
  var errorCode = error.code;
  var errorMessage = error.message;
  if (errorCode === 'auth/wrong-password') {
    showErrorMessage('Wrong password.');
  } else {
    showErrorMessage(errorMessage);
  }
});
```

Figura 17. Código base para la autenticación del usuario

Cuando un usuario se registra o accede, pasa a ser el usuario actual de la instancia de autenticación. La instancia de autenticación de Firebase conserva el

estado del usuario. Por lo tanto, al actualizarse la página (en un navegador) o reiniciar la app no se pierde la información del usuario.

Cuando el usuario cierra la sesión, la instancia de autenticación deja de conservar una referencia al objeto de usuario y con ello el estado de este.

4.1.2. Panel de Inicio

Una vez se pasa la autenticación, se accede a una pantalla inicial [Fig. 18] desde donde se podrá acceder a todas las secciones del sistema

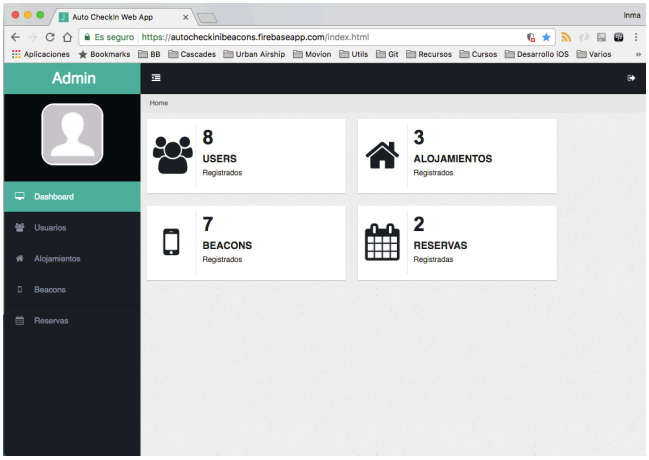


Figura 18. Panel de Inicio

4.1.3. Sección Usuarios

La sección de usuarios [Fig 19], al igual que el resto de secciones, contiene lo que denominamos un CRUD. Un CRUD es el acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: Create, Read, Update and Delete), con él nos referimos a la capa de persistencia en un software que dará cobertura a todas esas operaciones.

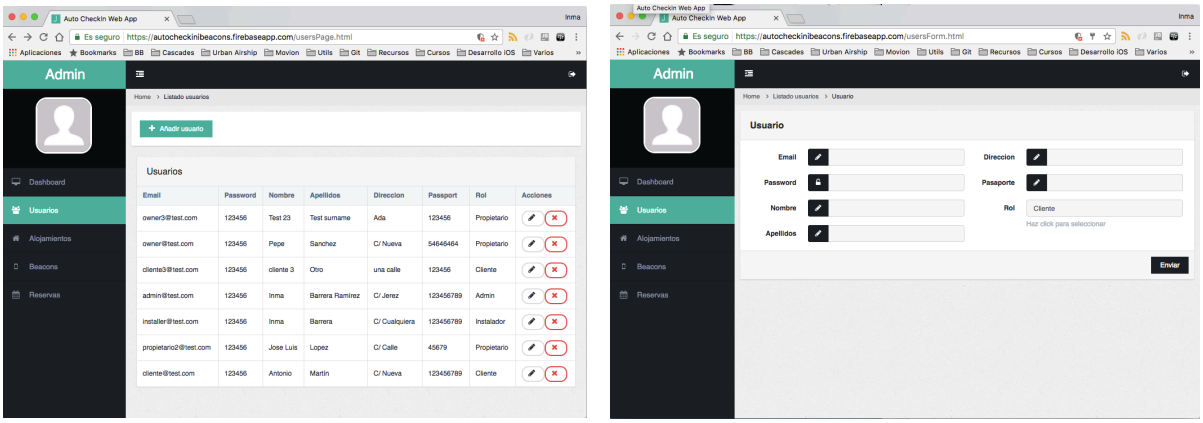


Figura 19. Sección de Usuarios

4.1.4. Sección Alojamientos

En la siguiente figura [Fig. 20], encontramos el diseño de la sección de alojamientos.

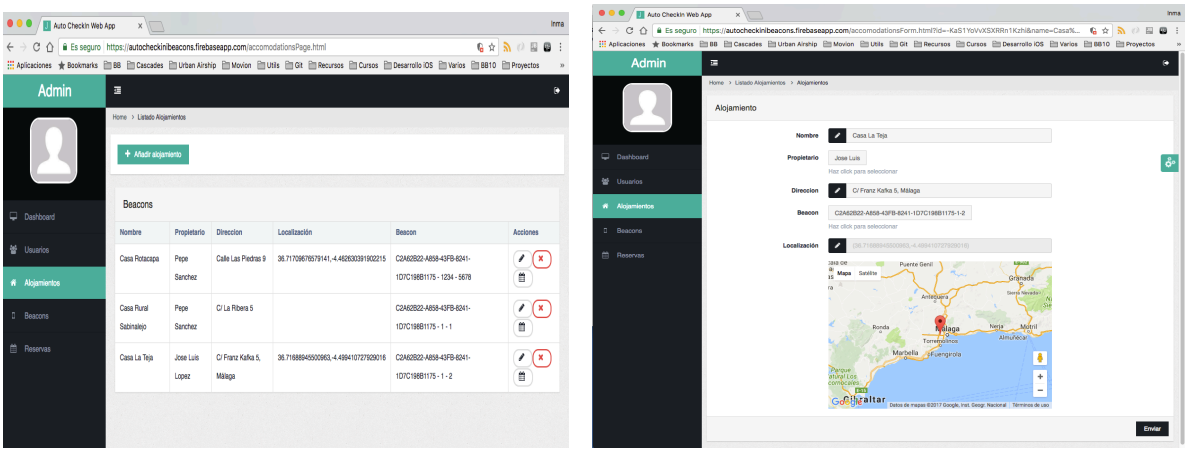


Figura 20. Sección Alojamientos

Cabe destacar que desde el listado de alojamientos podremos acceder al registro de entradas y salidas (Check In/Check Out) de los distintos usuarios al alojamiento seleccionado [Fig 21]. Si bien la modificación de esos datos no está permitida, al ser registrada esa información desde la aplicación móvil del usuario cliente cuando realmente efectúe tales acciones.

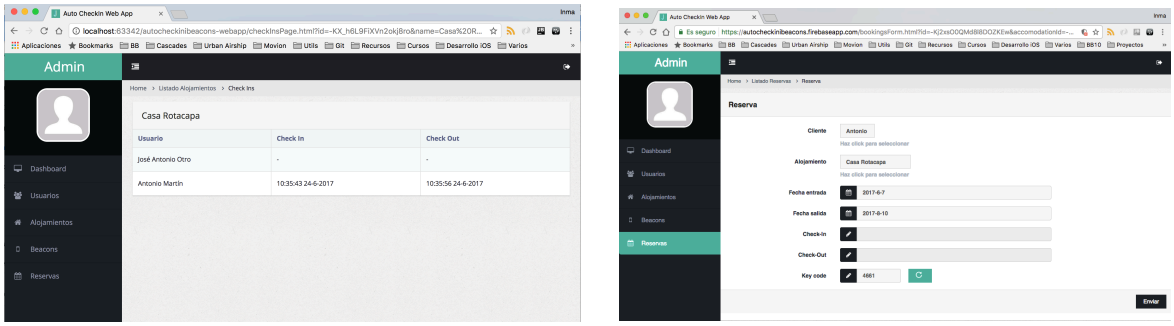


Figura 21. Registro de entradas y salidas al alojamiento

4.1.5. Seccion Beacons

De forma idéntica a las secciones anteriores, en esta encontramos el CRUD de beacons. [Fig. 22]

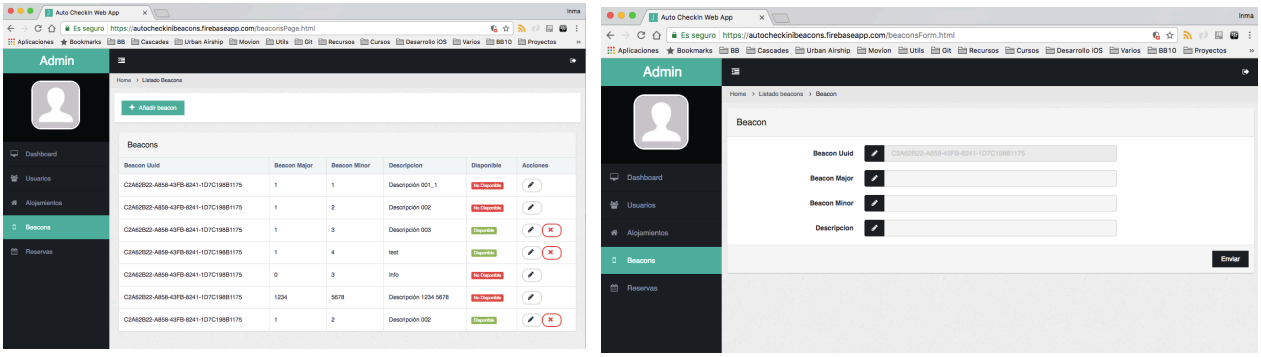


Figura 22. Sección Beacons

4.1.6. Sección Reservas

La última de las secciones de las aplicaciones la que contiene el CRUD para la gestión de reservas. [Fig. 23]. Vemos como es aquí donde se puede setear la llave necesaria para abrir la puerta de un alojamiento para una reserva, aunque el usuario propietario también podrá hacerlo desde la aplicación móvil y será el que tendrá prioridad, como veremos más adelante.

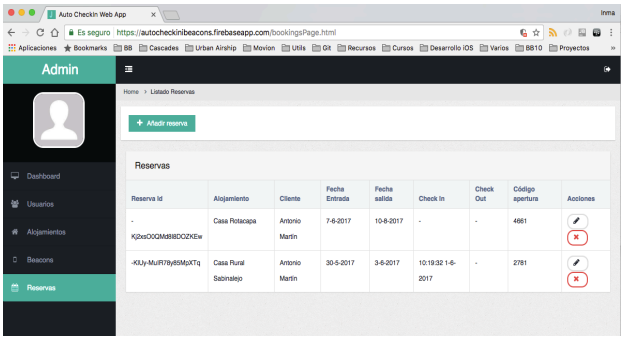


Figura 23. Sección de Reservas.

5. La aplicación móvil

La aplicación móvil es, sin duda alguna, el núcleo de la solución. Esta aplicación está diseñada para ser usada por el resto de perfiles de usuario del sistema: Instaladores, Clientes y Propietarios. Para ello, desde el login inicial se comprobará el rol del usuario que se pretende autenticar para entrar en la aplicación con el perfil adecuado y mostrar las vistas correspondientes a ese perfil. Más adelante veremos estos perfiles en la aplicación.

5.1. Arquitectura de la aplicación

La aplicación se ha diseñado siguiendo una filosofía MVC separando la capa de datos, la capa de vistas y la de negocio, encargada de toda la lógica de la misma.

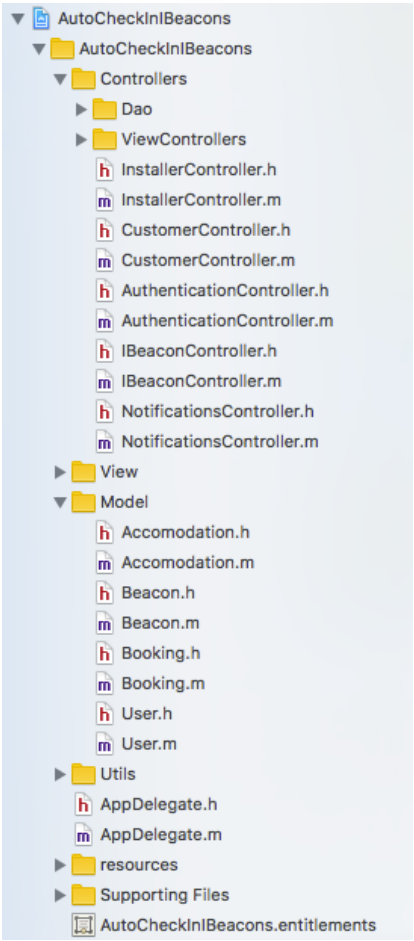


Figura 24. Estructura del proyecto.

Esta sería la estructura de la misma. [Fig. 24] donde encontramos algunos paquetes a destacar: l

En el paquete **Controller** está toda la lógica de negocio de la aplicación. A groso modo encontramos:

- los controladores para la capa de negocio de los distintos perfiles de usuario.
- Los Dao (Data Access Object). Clases que implementan la capa de conexión con Firebase. Se encarga tanto de la persistencia como de propia conexión con la base de datos.
- ViewControllers: controladores para las vistas que lo necesitan.

En el paquete **View** está toda la capa de visualización de la aplicación. Tanto los Storyboards, como cualquier componente visual customizado.

En el paquete **Model** se encuentran todas las clases que contiene el modelo de datos que ya hemos definido en capítulos anteriores.

El paquete **Utils** contiene todas las clases de utilidades genéricas que se usan, principalmente para formateo de datos y de estilos.

Por último, encontramos los paquetes **Resources** y **Supporting Files,** donde irán situados todos los ficheros de recursos (Imágenes, textos de localización, …) y los archivos de configuración necesarios para el proyecto en XCode respectivamente.

5.1.1. Controlador iBeacons

La clase IBeaconController es la encargada de la gestión de los beacons en cuanto a su descubrimiento. Se implementan aquí todos los métodos necesarios para su rastreo o monitorización [Fig. 22] y [Fig. 23]. De este modo, los controladores de los distintos perfiles podrán llamar a este, implementado como Singleton, para hacer uso de la funcionalidad deseada, ya que, como veremos unas veces necesitaremos rastreo de los iBeacons cercanos, mientras que otras tendremos que activar la monitorización de alguna región.

```
@interface IBeaconController: NSObject


@property(nonatomic, weak) id<IbeaconDelegate> delegate;


+ (id)sharedManager;
- (id)init;


-(void) requestPermissions;
```

```
-(void)startRangingAllBeaconsAppRegion;

-(void)startMonitoringAllBeaconsAppRegion;


-(void)startMonitoringAllBeacons☹NSArray*)allBeacons;

-(void)startRangingAllBeacons☹NSArray*)allBeacons;

-(void)startMonitoringBeacon☹Beacon*)beacon;

-(void)startRangingBeacon☹Beacon*)beacon;


-(void)stopMonitoringAllBeacons☹NSArray*)allBeacons;

-(void)stopRangingAllBeacons☹NSArray*)allBeacons;

-(void)stopMonitoringBeacon☹Beacon*)beacon;

-(void)stopRangingBeacon☹Beacon*)beacon;

-(void)stopMonitoringForRegion☹CLBeaconRegion *)41egión;

-(void)stopRangingForRegion☹CLBeaconRegion *)41egión;


-(void)stopAllMonitoredRegions;

-(void)stopAllRangedRegions;
```

Figura 22. Fichero de cabecera del Controlador de Beacons

```
-(void)startRangingAllBeaconsAppRegion {

    CLBeaconRegion * region = [[CLBeaconRegion alloc] initWithProximityUUID:[Util
stringToUUID:IBEACONS_REGION_UUID_DEFAULT] identifier:APP_ID];

    [self.locationManager startRangingBeaconsInRegion:region];

}

-(void)startMonitoringBeacon:(Beacon*)beacon {

    CLBeaconRegion *region = [beacon toRegion];

    if(region){

        [self updateNotificationRegion:region];

        [self.locationManager startMonitoringForRegion:region];

    }


}
```

Figura 23. Detalle de implementación de algunos de los métodos del controlador

5.2. Modo Instalador

Los usuarios con rol de instalador tendrán acceso a determinadas funcionalidades de la aplicación. La función básica del instalador es incluir iBeacons en el sistema y asignarlos a alojamientos.

5.2.1. Interfaz de usuario del instalador

En la interfaz del instalador separaremos las operaciones sobre los alojamientos y las de los iBeacons en tabs distintas.

Sobre los alojamientos, podemos acceder al listado de todos los que ya esten el sistema y podremos editar cualquiera de los existentes [Fig. 25]. También vemos que en la pantalla de detalle del alojamiento podemos asignar la posición actual al alojamiento, haciendo uso de los servicios de localización de iOS. De igual modo podemos asignar al alojamiento el propietario, así como el beacon deseado, accediendo al listado de propietarios y beacons respectivamente.

En la sección de iBeacons [Fig. 26], de igual forma, accedemos a todos los que ya están en el sistema o que se encuentren próximos al terminal que está ejecutando la aplicación que mostrarán la etiqueta “Online”. Esta búsqueda la realiza el controlador `IBeaconController` del que ya hemos hablado anteriormente, realizando un rastreo atendiendo a la región de beacons definida para la solución. También podemos añadir beacons de forma manual.

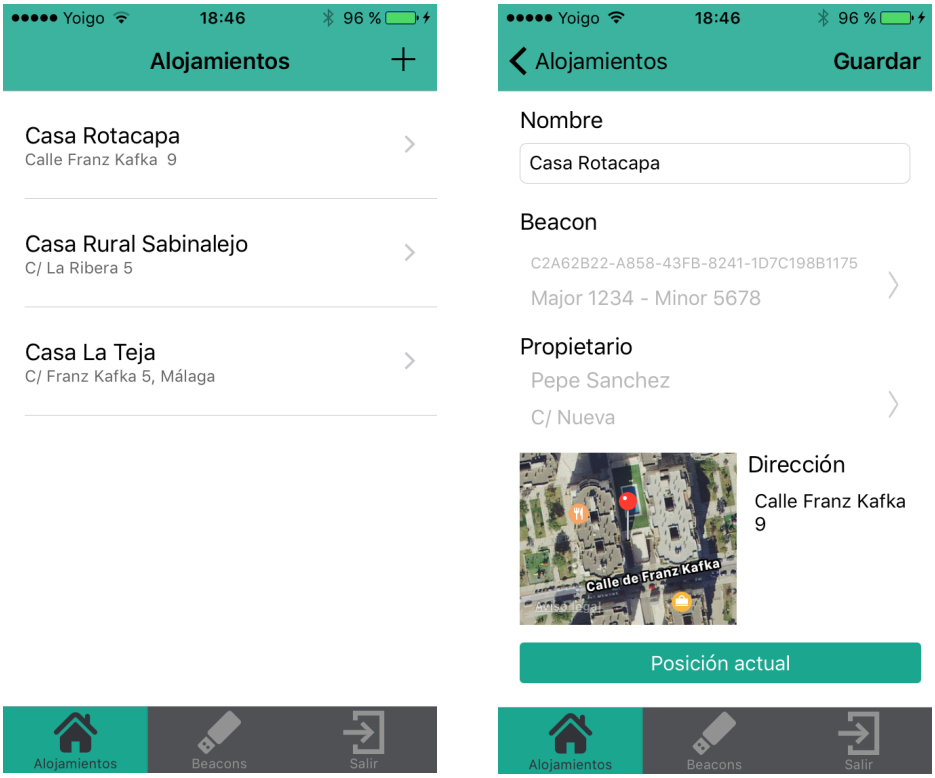


Figura 25. Sección de alojamientos, listado y detalle de alojamiento.

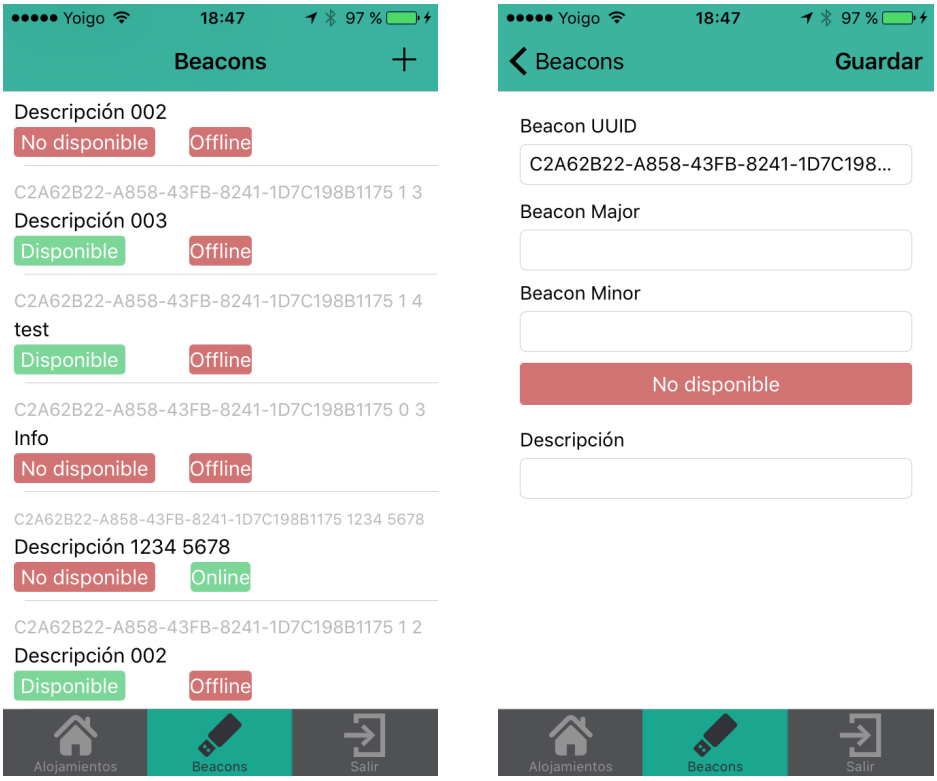


Figura 26. Sección iBeacons, listado y detalle de beacon.

5.3. Modo Propietario

El propietario tiene acceso a sus alojamientos y al sistema de reservas, donde podrá modificar los datos y sobre todo la llave que tiene asignada cualquiera de sus reservas para tener control absoluto sobre la puerta de su alojamiento.

5.3.1. Interfaz de usuario del propietario

La podemos dividir en dos partes, alojamientos y reservas, accesibles desde sus correspondientes tabs.

En la sección de alojamiento [Fig. 28] tendremos acceso al listado de todos los alojamientos del propietario y desde ahí al detalle de los mismos, para consulta o modificación. Nótese como desde esa pantalla de detalle podremos acceder al listado de reservas del alojamiento.

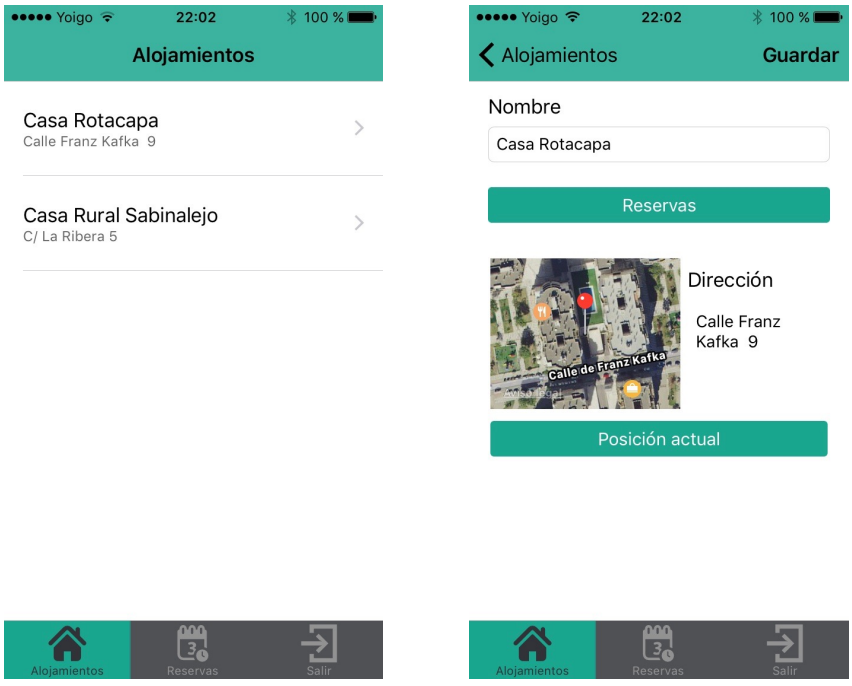


Figura 28. Sección alojamientos, listado y detalle de alojamiento

Por otro lado, el propietario también puede entrar en la sección de reservas de sus alojamientos [Fig. 29], desde donde poder añadir una nueva o editar alguna de las existentes. Vemos que en esa pantalla de detalle hay un botón desde el que generar llaves de forma aleatoria.

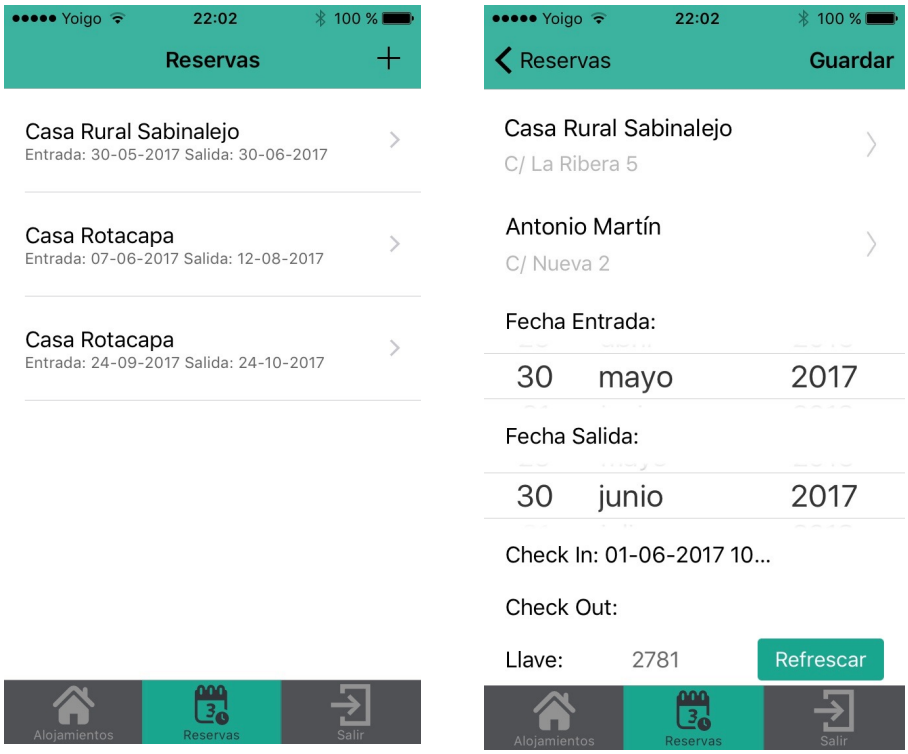


Figura 29. Sección reservas, listado y detalle de reservas

5.4. Modo Cliente

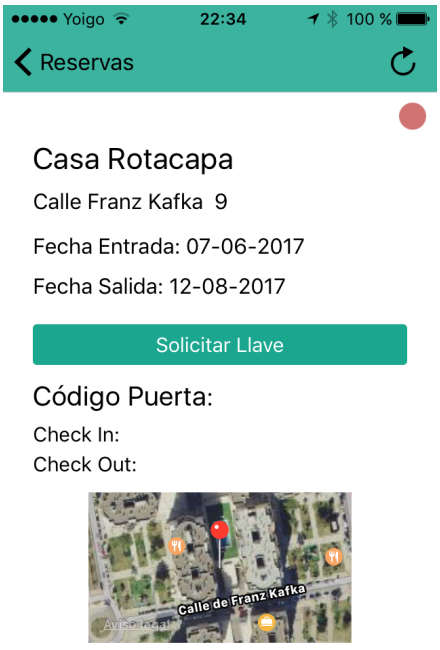
La función principal de la aplicación móvil para el usuario con rol cliente es solicitar la llave del alojamiento tras recibir la notificación de proximidad a un beacon asignado a un alojamiento en el que el cliente tiene una reserva activa. Para ello, la aplicación será sencilla, pero la lógica será algo más compleja al tener programar el **monitoreo** del iBeacon de ese alojamiento para que el terminal reciba notificación incluso con la aplicación en segundo plano o cerrada.

5.4.1. Interfaz de usuario del cliente

Veamos cómo queda el interfaz de la aplicación para usuarios con rol cliente.

En esta imagen [Fig. 31] vemos como queda la pantalla inicial de la aplicación para los clientes. Es simplemente un listado con las reservas que él tiene activas en el sistema con los datos básicos de la misma.

Figura 31. Listado de reservas del cliente



En la pantalla de detalle de la reserva [Fig. 32] el usuario puede consultar los datos de la misma. Se incluye una imagen de la localización del alojamiento al que pertenece que da la opción de abrir la aplicación de mapas de Apple si pulsamos sobre él y así iniciar la navegación.

Observamos en la esquina superior un círculo que podrá ser verde o rojo dependiendo de la cercanía de la puerta del alojamiento, basándonos en la proximidad del beacon de la entrada del alojamiento.

Figura 32. Detalle de la reserva

También se muestra un botón para solicitar la llave del alojamiento. Si se pulsa cuando no procede, es decir, cuando no ha encontrado el beacon

correspondiente se mostrará un mensaje de error; mientras que si se hace cuando está próximo se devuelve el código de entrada al alojamiento. [Fig. 33]

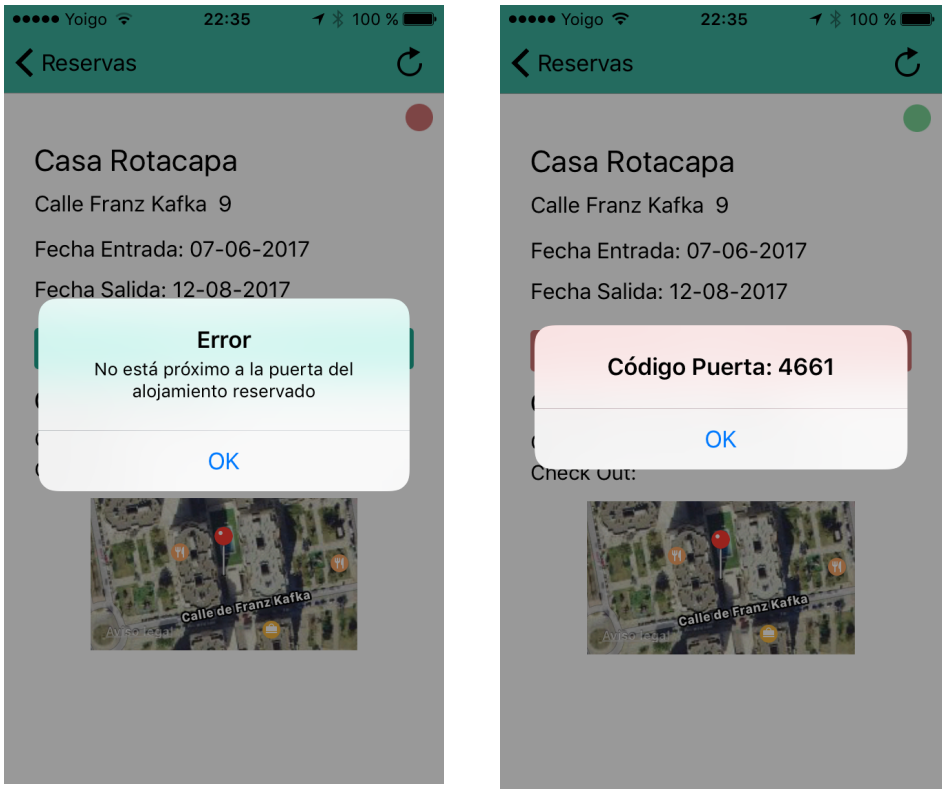
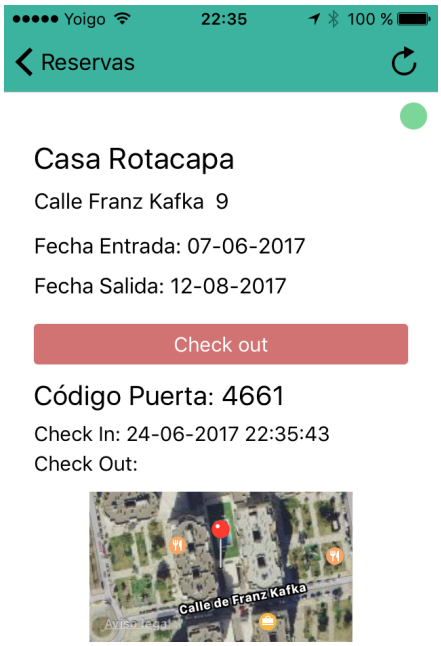


Figura 33. Resultados solicitud llave del alojamiento



En la imagen de al lado [Fig. 34] vemos la pantalla de la reserva una vez se ha realizado el check in. Vemos que desde ese momento se podrá realizar check out de igual manera y dejar constancia de esos datos en el sistema.

Como ya se ha dicho, el móvil que tiene la aplicación instalada recibe notificaciones cuando se encuentra próximo al alojamiento de la reserva. En las imágenes inferiores [Fig. 35] vemos la notificación recibida al centro de notificaciones cuando la aplicación está cerrada o en segundo plano y en la notificación cuando la aplicación se encontraba abierta.

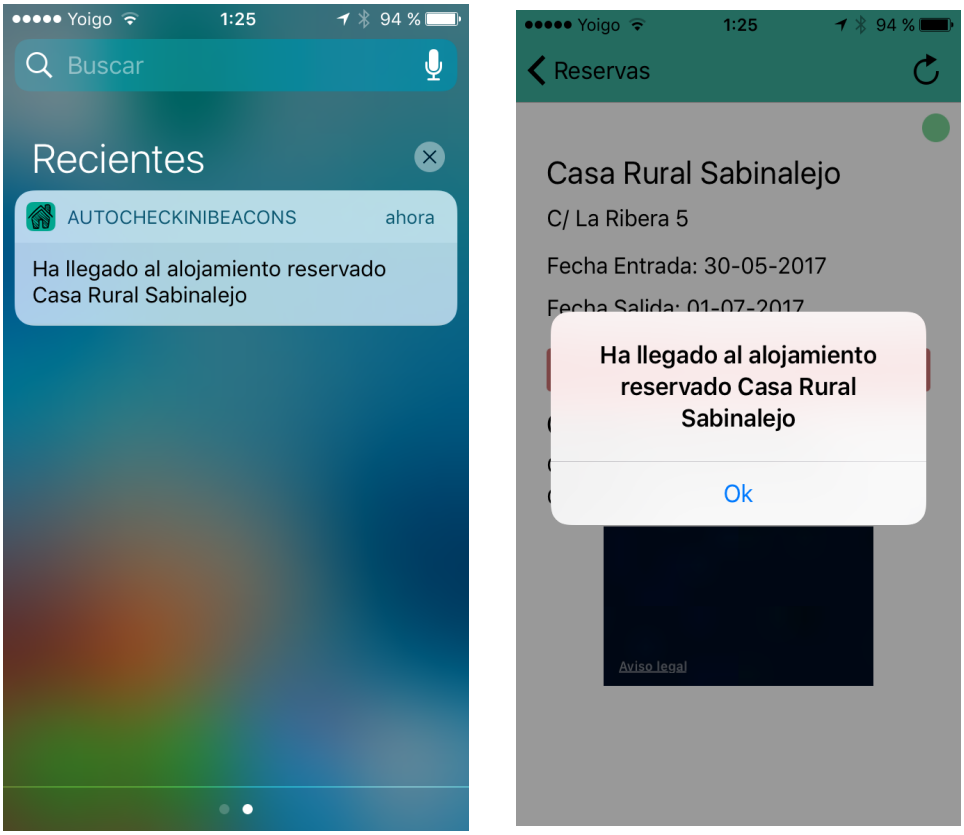


Figura 35. Notificaciones de proximidad al alojamiento

6. La aplicación para el dispositivo de la puerta

Cuando se inició el estudio del proyecto no nos planteamos como objetivo la implementación o desarrollo de ningún dispositivo específico de comprobación de llave o de apertura de la puerta del alojamiento, simplemente nos limitaríamos a usar alguno de los beacons existente en el mercado o usar Arduino para programar algún componente a nuestra medida. Con esto conseguiríamos emitir como un beacon en el formato elegido y nos serviría de apoyo en la solución para conseguir las implementaciones necesarias para realizar el rastreo y monitorización de los beacons, la asignación de estos a alojamientos, ...

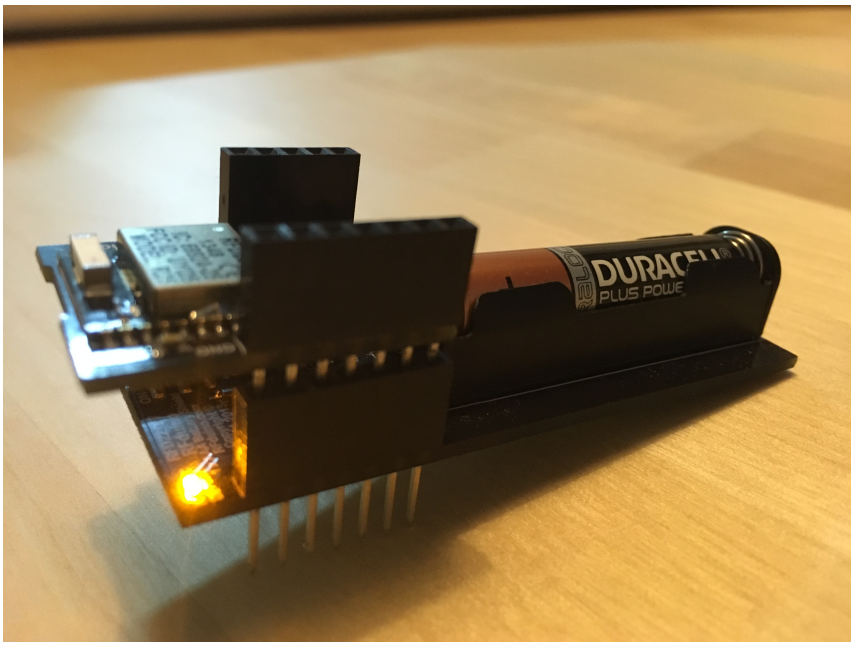


Figura 36. RFduino RFD22301

Y así lo hicimos, durante la mayor parte del tiempo se ha trabajado haciendo uso de Arduino, concretamente hemos usado el componente RFduino RFD22301 [Fig. 36] con capacidad Bluetooth y programado como iBeacon, obteniendo unos resultados óptimos en todo momento.

Aun así, necesitábamos algo más. Como ya hemos dicho, el RFduino cumplía perfectamente su misión como iBeacon, pero para emular una puerta de una forma más real necesitábamos, por un lado, algún dispositivo de entrada (con teclado numérico) para que el cliente pudiera introducir la llave recibida y, por supuesto,

conectividad a internet para, de este modo, hacer las comprobaciones pertinentes contra el backend y permitir o no la apertura de la puerta.

Después de evaluar el tiempo/esfuerzo necesario para implementar este componente en Arduino, se optó por la realización de un prototipo rápido usando otro dispositivo iOS de forma que no se necesitara programación a bajo nivel y que pudiéramos utilizar muchos aspectos de la lógica de conexión y acceso a datos en Firebase de una forma similar al resto de la aplicación ya implementada. Así pues, finalmente se implementa una pequeña aplicación que hace el prototipo más completo y que realiza dos funciones básicas:

- la de **ibeacon**, para que emita como tal, el instalador lo pueda asignar al alojamiento y el cliente reciba notificación cuando esté próximo a él, es decir, replicar la funcionalidad del componente RFduino que ya habíamos programado;
- y la de **simulación de mecanismo de entrada**, para que se permita al usuario cliente introducir la llave recibida y efectúe la comprobación de ese código introducido.

La implementación de este sistema para la puerta se ha realizado usando las mismas tecnologías y lenguaje que para la aplicación móvil: Objective C, Frameworks CoreLocation y CoreBluetooth y SDK de Firebase. De este modo, y al menos para todo el proceso de pruebas, podemos usar un terminal iPhone que simulará el dispositivo que se instalaría en la puerta del alojamiento.

6.1. Interfaz de usuario

Veamos una breve descripción de la aplicación implementada.

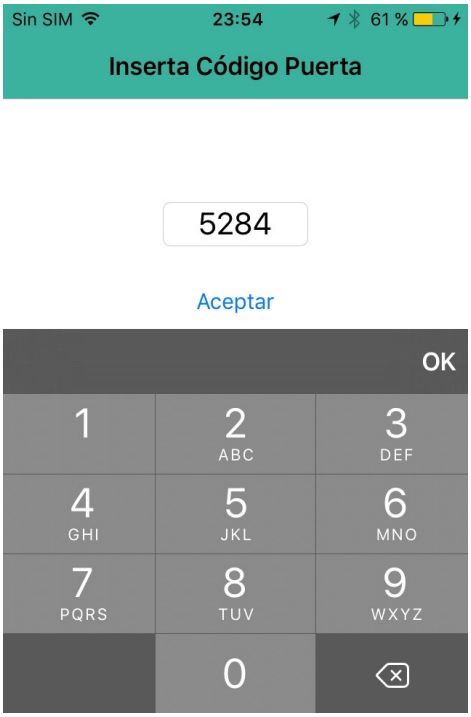


Figura 37. Pantalla Principal

Esta es la pantalla principal [Fig. 37], en ella el cliente introducirá el código que ha recibido en la aplicación para, si es correcto, realizar la apertura de la puerta. Este es el código o llave que recibirá una vez se ha situado próximo al beacon/alojamiento en el que tiene una reserva activa, ha pasado todos los mecanismo de validación y ha recibido el código.

Figura 38. Mensaje de apertura.

Si el usuario introduce el código de la puerta correcto, se recibirá un mensaje informativo como el de la imagen [Fig. 38]. Esto simula la apertura de la puerta en el sistema.

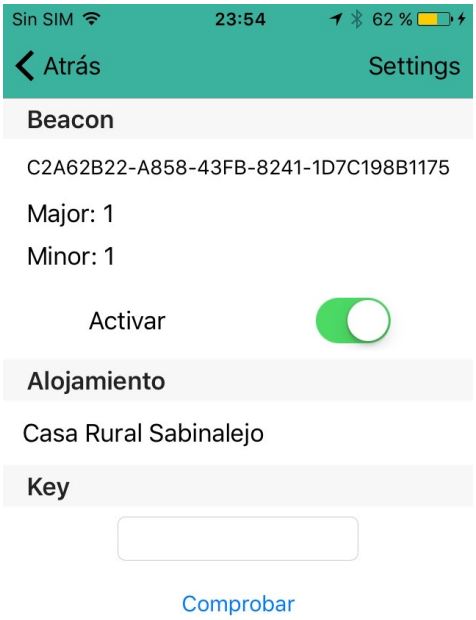


Figura 39. Pantalla de modo avanzado

Esta sería la pantalla de modo avanzado [Fig. 39], a la que accederíamos desde la pantalla principal introduciendo un código definido para él. Accederíamos a ella si quisiéramos ver los parámetros del beacon, a qué alojamiento pertenece y cuáles son los valores del beacon. Desde aquí también se podría comprobar si la llave que queremos usar para abrir la puerta es la correcta.

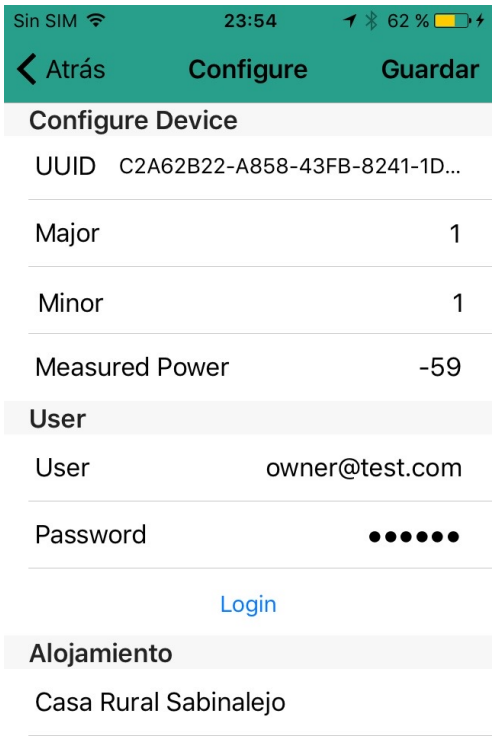


Figura 40. Pantalla de Configuración

Esta es la pantalla de Configuración [Fig. 40]. En ella podemos definir los parámetros de emisión del iBeacon. Además, como necesitamos acceder a la base de datos del sistema para realizar las distintas comprobaciones, se necesita autenticación por parte del propietario del alojamiento.

6.2. Planteamientos futuros para el dispositivo de la puerta

Esta aplicación para simular el dispositivo de la puerta cumple su función en el conjunto del prototipo, pero, aunque ya estaría fuera del ámbito del proyecto, de cara a una posible implementación real hemos estudiado algunas posibilidades y llegamos a la conclusión de que se podría conseguir un dispositivo de bastante calidad a un coste bastante asequible.

Básicamente necesitaríamos un dispositivo programable con conectividad y que pueda ser conectado a un mecanismo de apertura. Esto se podría conseguir con los siguientes elementos:

- **Raspberry Pi Zero W**: este modelo, que puede ser encontrado por unos 45€, tiene conectividad Wifi y puede ser programado con alguno de los SDK de Firebase para hacer las comprobaciones en la nube de las que ya hemos hablado.

- **Teclado numérico**: 10€ aprox. Necesitamos un dispositivo de entrada para poder introducir la llave para proceder a su validación.

- **Relé**: podemos encontrarlo por apenas 5€. Es simplemente, un dispositivo electromagnético que, estimulado por una corriente eléctrica muy débil, abre o cierra un circuito. Podemos conectarlo con la Raspberry y la puerta para que se abra cuando se necesite consiguiendo el comportamiento deseado.

7. Conclusiones

El presente Trabajo de Fin de Grado se ha centrado en el diseño e implementación de un sistema de check in automatizado para alojamientos turísticos mediante el uso de teléfonos inteligentes y beacons, y podemos decir que se han alcanzado todos los objetivos definidos en su inicio.

Para un sistema de estas características vimos que era necesario algún sistema de almacenamiento en la nube. De este modo para el desarrollo del backend nos decantamos por el uso de Firebase, una plataforma prácticamente nueva y con algunos aspectos aún en desarrollo, pero que cuenta con unos SDK bastante flexibles para movilidad y que nos han facilitado el trabajo en todo momento.

Después del estudio de las distintas tecnologías existentes, se decidió usar el protocolo **iBeacon**, no solo por tener detrás a Apple, sino principalmente por el hecho de que los resultados que se obtenían para el descubrimiento de los beacons en distintas situaciones eran simplemente los mejores para nuestras pretensiones. Además, se ha demostrado como una tecnología tan relativamente simple como ésta, combinada con una serie de aplicaciones y un poco de imaginación, puede ofrecer servicios de calidad añadidos a los ya existentes para las empresas, tiendas, centros comerciales, aeropuertos, …

En definitiva, este trabajo de investigación es una mínima muestra del gran potencial que atesora esta tecnología, que si además la combinamos con un sector como es el turismo y la actual proliferación de alojamientos turísticos (desde hoteles a viviendas de particulares) obtenemos un interesante nicho de mercado donde aplicarla y obtener sugerentes resultados.

8. Bibliografía y referencias:

[1] Título: iBeacons Bible 2.0.

Autor/es: Andy Cavallini

Publicación: 2014

[2] Título: Getting started with Bluetooth Low Energy

Autor/es: Carles Cufi, Robert Davidson

Publicación: 2014

[3] Referencia: https://developer.apple.com/ibeacon/

Descripción: Documentación oficial Apple de iBeacon

Publicación: 2017

[4] Referencia: https://developers/google.com/beacons/eddystone

Descripción: Documentación oficial de Eddystone

Publicación: 2017

[5] Referencia: https://firebase.google.com/

Descripción: Documentación oficial Firebase

Publicación: 2017

[6] Título: Programación con Objective-C

Autor/es: Stephen G. Kochan

Publicación: 2012

[7] Título: Javascript: La Guía Definitiva

Autor/es: Flanagan David

Publicación: 2007