PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

INGENIERÍA EN SISTEMAS Y COMPUTACIÓN

DISERTACIÓN PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS Y COMPUTACIÓN

"DESARROLLO DE UN APLICATIVO QUE IMPLEMENTE UN SISTEMA DE GEOREFERENCIACIÓN Y MENSAJERÍA EN DISPOSITIVOS MÓVILES iOS"

AUTOR:

SALGADO ORTIZ DAVID MARCELO

DIRECTOR:

ING. GUSTAVO CHAFLA

QUITO,2018

Contenido

Índice de tablas

Índice de diagramas

Capítulo I: Introducción

1.1. Justificación

Una de las principales características de los dispositivos móviles actuales es la integración de un GPS en su hardware. Esta funcionalidad abre las puertas a los desarrolladores para llevar a cabo un control de cada dispositivo obteniendo su ubicación actual en todo momento, siempre y cuando el usuario así lo autorice.

De ahí surgió la idea de crear este sistema como parte de un Proyecto de Investigación de Ingeniería en el área de Redes de la PUCE, el cual permite a un administrador llevar un control de la gente que se encuentra incluida en una red de trabajo conociendo los lugares que visita y en qué momento lo hace, a su vez, también le da la posibilidad de comunicarse con cada uno mediante un chat integrado.

Este aplicativo puede ser usado por visitadores médicos, camiones, buses escolares o familias que deseen llevar el control de las rutas cubiertas, teniendo un control en tiempo real de cada miembro de la red vinculada al administrador.

Hay soluciones existentes que realizan funciones similares, la ventaja de este sistema en particular y lo que lo diferencia de los demás es la capacidad de llevar un control de toda la ruta del dispositivo sabiendo en cuando y donde estuvo.

1.2. Planteamiento de problema

El rastreo en tiempo real y un mecanismo de comunicación en un aplicativo móvil es una herramienta que puede aportar mucho a personas que requieran controlar redes de empleados y los tiempos que los mismos emplean realizando cada tarea asignada.

Este aplicativo busca dar solución a estas necesidades y crear una herramienta fiable de control y contacto entre los miembros registrados en el círculo. Este aplicativo tiene que implementar 3 tareas fundamentales: un sistema de ingreso de usuarios, un sistema georeferenciación que permita enviar las coordenadas del usuario al administrador y por último uno que otorgue la posibilidad de comunicarse con su red mediante un sistema de mensajería.

Otro punto muy importante a considerar es el área legal, en muchos países el rastrear a un empleado atenta contra la privacidad de cada individuo y por lo tanto es considerado ilegal es importante analizar la ley del país para saber si un aplicativo de esta naturaleza puede operar o bajo qué condiciones puede hacerlo.

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar un aplicativo que implemente un sistema de georeferenciación y mensajería para dispositivos móviles con sistema operativo iOS.

1.3.2. Objetivos específicos

- Crear un módulo de ingreso de usuarios con autenticación al aplicativo móvil.
- Implementar un módulo que permita el rastreo de cada dispositivo mediante el uso del GPS integrado.
- Crear un módulo que permita a los usuarios de la red comunicarse mediante mensajería instantánea.
- Investigar el estado del arte de las tecnologías a aplicarse en el desarrollo del sistema
- Aplicar una metodología que asegure la agilidad y la calidad del sistema a desarrollar.
- Investigar la viabilidad legal de poder operar un sistema de rastreo móvil.

1.4. Alcance

El aplicativo móvil de rastreo y mensajería debe contener un sistema, el cual estará constituido de los siguientes módulos:

- **Módulo de autenticación. -** Este módulo permitirá a los usuarios ingresar al aplicativo autenticando su sesión por medio de email y contraseña.
- **Módulo de ingreso de miembros a la red.** – Este módulo permitirá a los usuarios administradores agregar, eliminar y ver usuarios a su red de rastreo.
- **Módulo de confirmación de cuenta. -** Este módulo enviará vía correo electrónico un código que permitirá a los usuarios activar su cuenta y habilitar su uso.
- **Módulo de envío y recepción de notificaciones push.-** Este módulo en la nube procesará las notificaciones y destinatarios para su envío, adicionalmente el aplicativo debe conectarse con el módulo para poder receptarlas.
- **Módulo de georeferencia.-** Este módulo permitirá a los usuarios activar o desactivar el rastreo de su dispositivo, enviar su posición, visualizarla en el mapa y reportar al administrador su ubicación para informar la llegada a puntos claves. Además, permitirá a los usuarios ver sus trayectos realizados junto con la información referente a ese viaje, los usuarios administradores podrán ver los trayectos de cada miembro de su red.

Capítulo II: Fundamentación teórica

2.1. Entorno de desarrollo

El entorno de desarrollo para este proyecto lo comprenden tres partes: el sistema operativo sobre el que se va a trabajar, el lenguaje de programación y el IDE a usar.

2.1.1. Sistema operativo

Capa compleja entre el hardware y el usuario, concebible también como una máquina virtual, que facilita al usuario o al programador las herramientas e interfaces adecuadas para realizar sus tareas informáticas, abstrayéndole de los complicados procesos necesarios para llevarlas a cabo. (Baz Alonso, Ferreira Artime, Álvarez Rodríguez, & García Baniello)

El sistema operativo usado para desarrollar esta aplicación móvil es denominado iOS cuyas siglas significan iPhone Operating System creado por Apple Inc. lanzando su primera versión en junio 29 del 2007 llamada iOS 1, actualmente iOS se encuentra en su versión 11 del sistema operativo. Esta es la plataforma bajo la que funcionan los dispositivos iPhone, iPods Touch, Apple Watch e iPads exclusivamente.

Como desarrolladores, comprender el funcionamiento básico de los sistemas operativos y las principales alternativas que ofrecen en muchos de sus puntos, o saber diseñar algoritmos y procesos que se ajusten mejor al sistema operativo en que vayan a ejecutarse, puede resultar en una diferencia cualitativa decisiva en el producto final. (Wolf, Ruiz, Bergero, & Meza, 2015)

Es importante saber que funcionalidades están disponibles en cada versión de un sistema operativo ya que esto nos indica

que herramientas podemos utilizar y como desarrolladores saber hasta qué punto es posible realizar lo que tenemos en mente, conociendo así las limitaciones del hardware y software con el que estamos trabajando. En la versión 4 de iOS se integró una funcionalidad llamada Multitasking la cual abre la oportunidad a desarrolladores para realizar actividades en segundo plano, es decir, continuar ejecutando actividades aun cuando la aplicación se ha cerrado. Entre las posibles funcionalidades que un desarrollador puede continuar realizando en segundo plano están las siguientes:

- Audio
- Ubicación
- VoIP (Realizar conversaciones por internet)
- Descarga de contenido en segundo plano
- Notificaciones remotas
- Comunicación con Bluetooth

Para el desarrollo de este aplicativo basado en la geolocalización del dispositivo se utilizará el modo de “Ubicación” que nos permitirá obtener las coordenadas del usuario cuando se encuentra en movimiento independientemente de si está con su celular suspendido o fuera de la aplicación.

2.1.2. Lenguaje de programación

La programación de computadoras es el arte de hacer que una computadora haga lo que nosotros queramos. En el nivel más simple consiste en ingresar en la computadora una secuencia de órdenes para lograr un cierto objetivo. (Sánchez Rodríguez & Cano Castañeda, 2014)

“El lenguaje de programación es nuestra forma de comunicarnos con el dispositivo móvil en este caso y transmitirle nuestros requerimientos” (Sánchez Rodríguez & Cano Castañeda, 2014) un programa de computadora se puede definir como una secuencia de instrucciones que indica las acciones o tareas que han de ejecutarse para dar solución a un problema determinado.

Los lenguajes de programación nos permiten crear código, el código mueve muchas de las cosas que están a nuestro alrededor. El código es el lenguaje de la tecnología y su presencia está básicamente en todo nuestro entorno cuando queremos hacer una compra en línea el código monta nuestra orden en un sistema, si queremos enviar un mensaje de texto a otra persona el código se encarga de transmitir esos caracteres a la pantalla del destinatario.

Existen dos lenguajes oficiales de programación para dispositivos móviles iOS y son Objective C y Swift. Objective C era la herramienta oficial que Apple usaba como lenguaje nativo para desarrollar aplicaciones móviles en sus plataformas hasta el año 2014, es en junio 2 cuando aparece Swift, un lenguaje más rápido y con una sintaxis simplificada.

Los lenguajes de programación son notaciones que describen los cálculos a las personas y las maquinas. Nuestra percepción del mundo en que vivimos depende de los lenguajes de programación, ya que todo el software que se ejecuta en todas las computadoras se escribió en algún lenguaje de programación. Pero antes de poder ejecutar un programa, primero debe traducirse a un formato en el que una computadora pueda ejecutarlo. (Aho, Ullman, Sethi, & Lam, 1998)

Las órdenes que nosotros escribimos están escritas en un lenguaje de alto nivel, Swift es un lenguaje de alto nivel y tiene

que ser procesado por un compilador para poder pasar a lenguaje de máquina y que el dispositivo pueda ejecutar este código.

El compilador de Swift a más de pasar el código fuente a lenguaje de máquina posee una integración con el IDE de desarrollo lo que permite colorear la sintaxis y completado de código que nos permite predecir lo que queremos realizar, estas características agilitan el proceso de escritura y desarrollo.

2.1.3. IDE

Un entorno de desarrollo integrado, llamado también IDE (sigla en inglés de Integrated Development Environment), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios. (Sanchez, 2014)

El IDE es una herramienta básica a la hora de programar por lo que es indispensable contar con ella, para este caso ya que deseamos desarrollar en Swift el entorno de desarrollo usado es XCode, el IDE introducido por Apple el 24 de octubre de 2003. La versión usada es XCode 10, este IDE es usado para desarrollar tanto en Swift como Objective C que son los dos lenguajes que maneja iOS. Xcode nos brinda ciertas funcionalidades claves a la hora de desarrollar como son: Resaltado de sintaxis, herramientas para debug, completado de código, errores y advertencias, documentación, entre otras. Si se desea desarrollar para iOS de forma nativa esta herramienta es la única y está disponible solo para usuarios de Mac OSX. (Sanchez, 2014)

2.2. Aplicaciones web vs nativas vs híbridas

A la hora de iniciar una aplicación móvil los programadores tenemos 3 opciones al momento de desarrollar: aplicaciones nativas, aplicaciones híbridas o aplicaciones web. Cada tecnología lleva consigo ventajas y desventajas y ya queda a criterio del programador saber cuál es la mejor ruta a tomar dependiendo de los requerimientos funcionales y no funcionales del aplicativo.

Las aplicaciones web para móviles son diseñadas para ser ejecutadas en el navegador del dispositivo móvil. Estas aplicaciones son desarrolladas utilizando HTML, CSS y JavaScript, es decir, la misma tecnología que la utilizada para crear sitios web.(Delia, Galdamez, Thomas, & Pesado)

Las aplicaciones nativas son aquellas que (Letelier & Penadés, 2002)se conciben para ejecutarse en una plataforma específica, es decir, se debe considerar el tipo de dispositivo, el sistema operativo a utilizar y su versión.(Delia, Galdamez, Thomas, & Pesado)

Las aplicaciones híbridas combinan lo mejor de los dos tipos de aplicaciones anteriores. Se utilizan tecnologías multiplataforma como HTML, JavaScript y CSS, pero se puede acceder a buena parte de las capacidades específicas de los dispositivos.(Delia, Galdamez, Thomas, & Pesado)

Tabla 2-1 Comparación entre aplicaciones

Característica	Página móvil	Aplicación nativa	Aplicación híbrida
Plataforma	Navegadores móviles	iPhone OS (iOS), Windows Mobile, Blackberry OS, Symbian, Android	iPhone OS (iOS), Windows Mobile, Blackberry OS, Symbian, Android
Distribución	URL y códigos QR	Tiendas de aplicaciones según plataforma	Tiendas de aplicaciones según plataforma
Instalación	Se accede directamente y puede quedar disponible mediante un *launcher* en el dispositivo	Se realiza una vez y queda disponible	Se realiza una vez y queda disponible para todas las plataformas
Costos de desarrollo	Menores	Mayores	Menores que los de las nativas
Rendimiento	HTML5 mejora la infraestructura de la red	Más rápido, especialmente si requiere procesos gráficos pesados	Se desarrolla como nativo cuando el rendimiento sea esencial
Integración de *hardware*	Limitada	Completa	Buena
Acceso fuera de línea	Solo en algunos dispositivos mediante HTML5	Completo	Completo
Usabilidad	Buena	Gran cantidad de efectos amigables en la interfaz atractivos para el usuario	Utiliza lo mejor de lo nativo y lo mejor de la red

Referencia: (Angulo, 2013)

La tabla 2-1 muestra las características de los diferentes tipos de aplicación y las funcionalidades especiales que estas proveen en los dispositivos móviles.

Tabla 2-2 Características entre aplicaciones

Característica	Página móvil	Aplicación híbrida	Aplicación nativa
Necesita acceso al *hardware* del dispositivo (cámara o GPS)	Peor	Mejor	Mejor
Debe ser funcional sin conexión	Peor	Intermedia	Mejor
Requiere cálculos en tiempo real o gráficos 3D de alto rendimiento	Peor	Intermedia	Mejor
Debe tener presencia en sitios como Google Play, AppStore y AppWorld	Peor	Mejor	Mejor
Tendrá cambios regulares en las reglas de negocio	Intermedia	Mejor	Peor
El presupuesto es reducido	Mejor	Intermedia	Peor
Depende de una constante conexión con el servidor	Intermedia	Mejor	Intermedia

Referencia: (Angulo, 2013)

La tabla 2-2 muestra las características y la calidad de las mismas dependiendo del tipo de aplicación.

Analizando el aplicativo móvil se identifica tres funcionalidades generales que serán usadas; la primera es la comunicación con el servicio de Firebase vía internet que será usado para el almacenamiento y lectura de datos, la segunda es el GPS para lo cual se requiere interactuar con el hardware del dispositivo y soporte de notificaciones push para poder enviar información relevante cuando el usuario no esté dentro del aplicativo. Por lo que basado en los cuadros de Angulo se identifican tres puntos claves en esta aplicación:

- El aplicativo va a requerir de una conexión a internet constante.
- El aplicativo va a ser uso del GPS.
- Constantemente va a tener que realizar consultas e interactuar con el servidor.

Por esto realizar una aplicación nativa es la mejor opción a seguir.

2.3. Firebase

Inicialmente conocida como Envolve, fundado por James Tamplin y Andrew Lee en el año 2011; proveía a los desarrolladores con un API, la cual permitía la integración de mensajería instantánea en línea, posterior a la liberación de este API los fundadores observaron que no era utilizada con el propósito para el cual fue creado, su uso se centraba a un mayor nivel, transfiriendo de forma sincrónica y en tiempo real, información como estados de jugadores en una plataforma en línea. Posterior a este suceso los fundadores decidieron separar su plataforma de mensajería de la arquitectura en tiempo

real y así fundar Firebase como una compañía separada en abril del 2012.

Posterior a una serie de eventos Firebase fue adquirido por Google y ellos integraron diferentes servicios, logrando que Firebase se convirtiera en una plataforma unificada para desarrolladores móviles. Firebase provee distintos servicios en la nube entre ellos Firebase Cloud Messaging (FCM), Firebase Auth (Autenticación), Realtime Database (Base de datos)

FCM fue inicialmente conocido como Google Cloud Messaging o GCM, posterior a la integración de Firebase, fue renombrado a Fire Cloud Messaging, es un servicio creado en la nube para el manejo de mensajería y notificaciones (Push Notifications) para aplicaciones desarrolladas en Android, iOS y web, posibilitando la comunicación entre las diferentes plataformas. (Google, Firebase, 2018)

Firebase Auth es el servicio que provee Firebase para la autenticación de usuarios a través de diferentes medios; como correos electrónicos, números de teléfono móvil, integración con otras aplicaciones como redes sociales y autenticación a través de este medio.

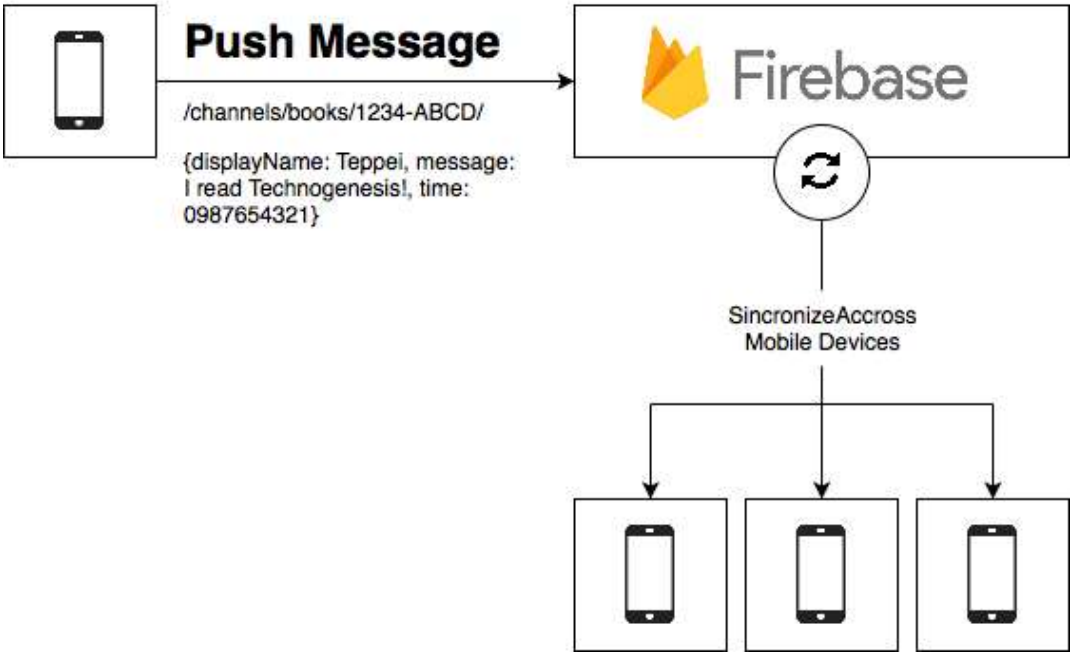
Este proceso se realiza adquiriendo las credenciales del usuario y enviándolas al SDK de Firebase Auth, este se encarga en el back-end de procesar la información y verificar que las credenciales, a través del método de autenticación son válidas, devolviendo una respuesta al cliente y de esta forma accediendo al perfil del usuario. (Google, Firebase, 2018)

Realtime Database es el servicio de Firebase que provee una base de datos NoSQL que se encuentra almacenada en la nube, la información es sincronizada sobre todos los clientes en tiempo real y permanece disponible cuando la aplicación se encuentra desconectada.

La información es almacenada en forma de JSON y se sincroniza para cada cliente conectado, independiente de la plataforma en la que se encuentre desarrollada.

Los servicios de sincronización siempre se encuentran disponibles en las aplicaciones conectadas, en el instante que la aplicación obtiene nuevamente una conexión, esta actualiza toda la información generada desde el momento que perdió la conexión. (Google, Firebase, 2018)

Diagrama 2-1 Arquitectura de Firebase



Referencia: (Stevenson, 2017)

En la imagen 2-1 se observa el proceso que se realiza sobre la base de datos en tiempo real, se envía el mensaje push

hacia la base de datos y automáticamente todos los equipos conectadas a la misma actualizan su información.

2.4. Google Maps y Google Places

Google Maps es la librería de Google que permite la visualización de mapas a través de un cliente al cual se ha integrado estos servicios, a su vez es posible observar la posición a través del par de coordenadas que se tome como parámetros y por medio de codificación mostrarlo en los mapas.

Existen diferentes tipos de vistas de los mapas de Google; vista satelital, la cual permite observar a mayor detalle un lugar en específico, vista terreno donde es más visible las calles antes que los lugares existentes, por defecto Google Maps tiene esta vista activada. (Google, Google Maps Platform, 2018)

Diagrama 2-2 Vista de Google Maps en modo terreno

Referencia: Google. (s.f). Mapa de Luxemburgo. Recuperado el 12 de Octubre 2018 de https://www.google.com.ec/maps/place/Luxembourg/@49.6108618,6.1271319,15.33z

El diagrama 2-2 muestra cómo se visualiza el mapa cuando la opción de vista de territorio se encuentra activada.

Diagrama 2-3 Vista de Google Maps en modo satélite

Referencia: Google. (s.f). Mapa de Luxemburgo. Recuperado el 12 de Octubre 2018 de https://www.google.com.ec/maps/place/Luxembourg/@49.6108618,6.1271319,15.33z

El diagrama 2-3 muestra cómo se visualiza el mapa cuando la opción de vista de satélite se encuentra activada.

Google Places forma parte de la plataforma de Google Maps, la cual recupera información de lugares y puntos de interés registrados, utilizando la posición donde se encuentra el usuario o por buscador, permite la función autocompletar en base a lo escrito por el usuario. (Google, Google Maps Platform, 2018)

2.5. Node.js

Node.js es concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node está diseñado para construir aplicaciones en red escalables. (Joyent Inc, Node.js, 2018)

Node.js es un entorno de ejecución de JavaScript que ejecuta código JavaScript fuera de un navegador permitiendo a los desarrolladores escribir herramientas de líneas de comando y realizar codificación de lado de servidor para producir páginas web dinámicas antes de ser visualizadas por el usuario en su

navegador. Tiene una arquitectura enfocada a eventos capaz de realizar operaciones asincrónicas.

Node.js contiene un gestor de paquetes conocido como node package manager o por sus siglas NPM, este facilita la instalación y desinstalación de librerías y dependencias entre ellas. (Joyent Inc, Node.js, 2018)

El entorno de Node.js hace muy sencillo el manejo de librerías para cualquier proyecto siendo muy fácil obtener paquetes que ayuden a realizar tareas en el desarrollo, ahorrando tiempos en el proceso de construcción del software. Al tener una gran comunidad detrás hace que existan se puede encontrar muchas opciones haciendo de Node.js una gran herramienta para construir aplicaciones.

2.6. Metodología para el desarrollo

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. (Letelier & Penadés, 2002)

Dentro de las metodologías para el desarrollo y la ingeniería de Software, el desarrollador ha experimentado con diversas técnicas y modelos para la creación de software. Desde modelos clásicos como son los de Cascadas, espiral, evolutivo, etc., metodologías tales como JSD, Warnier, PSP; TSP, etc.,

hasta los más recientes que de manera general han sido denominados como Metodologías ágiles de software.

El objetivo principal de una metodología ágil de desarrollo de software es potenciar las relaciones técnicas y operacionales (desarrolladores y clientes), esto con el de garantizar la calidad de uso y funcionalidades de software al finalizar su desarrollo.

2.6.1. Metodología XP

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (Letelier & Penadés, 2002)

El entorno de trabajo en metodología XP se concentra en necesidades del usuario final, para esto es necesario centrarse en relaciones interpersonales para la adecuada comunicación y gestión de la información Desarrollador – Cliente.

Este método de desarrollo tiene la principal característica de enfocar el trabajo en base a iteraciones, en donde cada una de ellas se realiza un producto o entregable y las siguientes será versiones empíricas y mejoradas de sus antecesores. Estos ciclos serán realizados de manera constante hasta que se cumplan de manera satisfactoria las necesidades del usuario final, con el objetivo de garantizar la calidad de uso y funcionalidad del software.

XP se caracteriza también por su capacidad de trabajo en escenarios muy variables con respecto al requerimiento de software, ya que estos, en ciertos puntos, por tanto, es necesario siempre tener presente el criterio del cliente o usuario final, tan imprescindible como considerarle un miembro más del equipo de desarrollo.

Este marco de métodos es propicio para grupos pequeños de programación, puesto que en ese ambiente es más fácil la comunicación, tema que es muy importante para tratar de mejor manera el flujo y puntos de control de trabajo con respecto al desarrollo del producto como tal.

La metodología XP maneja un flujo de trabajo dividió en fases. Estas son fase de exploración, planteamiento, diseño, codificación, pruebas e implementación.

2.6.1.1. Fase de exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. (Letelier & Penadés, 2002)

La primera fase de la metodología es crucial y consecuente para sus predecesores, puesto que en esta se especifica la primera entrega de un producto o entregable dentro del proceso de desarrollo de software. Se genera junto con el usuario final mediante historias de usuarios un arquetipo de lo que será el producto final de software, junto a ello de igual forma se genera o especifica las funcionalidades que deberá tener, tratando así de realizar un prototipo.

Este proceso puede tomar entre semanas o meses dependiendo de la escalabilidad o dimensión de complejidad del proyecto de software como tal.

2.6.1.2. Fase de planificación

La planificación es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario, y, asociadas a éstas, las entregas. (Joskowicz, 2008)

En esta fase se define la importancia de cada historia de usuario en un mutuo acuerdo con el cliente final con el fin de establecer las principales características o funcionalidades que deberá tener el producto final o las de primera necesidad.

De igual manera todos los programadores deberán establecer el tiempo de desarrollo para cada uno de los puntos anteriores mencionados, en base al esfuerzo y conocimiento en el que puedan desempeñarse.

Una vez organizado el desarrollo de módulos o entregables se definirá un acuerdo de entregas, las cuales se recomienda no sobre pasar los 3 meses.

Del mismo modo, se puede generar una planificación de entregas en base a la complejidad, escalabilidad, alcance del proyecto y a la experiencia y esfuerzo de los desarrolladores. Esto será base para realizar el cálculo y el número de iteraciones a realizar durante el desarrollo del software.

El resultado o entregable de esta fase será un Plan de entregas o normalmente llamado “Release Plan”

2.6.1.3. Fase de diseño

Se preparan una serie de documentación que será reflejo de la fase de exploración, a su vez esclarecerá la visión general de como deberá desarrollarse el software con respecto a sus características y funcionalidades.

XP propone una documentación con ítems sencillos y simples, mediante soluciones “Spike” y metáforas, que básicamente es dar respuesta a todo el proceso de desarrollo de manera en que todos entiendan.

Es muy importante que esta filosofía de trabajo se haya generado en mutuo acuerdo en todo el equipo de desarrollo incluyendo el usuario final.

El entregable de esta fase será la documentación constituida por todas las historias de usuario, mapas, nomenclaturas y diagramas necesarios para el evidenciar el funcionamiento principal que tendrá el software a través de su desarrollo y sus iteraciones.

2.6.1.4. Fase de codificación

En este punto se desarrollan todos los módulos, historias de usuario o funcionalidades con respecto al plan de entregas generado.

Como se ha mencionado anteriormente, el usuario final juega un papel importante, ya que las historias de usuario pueden no tener la suficiente información necesaria y técnica para poder ser desarrollada, se necesita la presencia del usuario final para poner en claro cualquier información faltante.

Es importante sugerir la presencia de reglas o estándares de programación para que todo el equipo pueda seguir un mismo formato de desarrollo y prevenir la mayoría de los inconvenientes al momento de realizar una implementación general.

XP también propone una programación con tendencia a pruebas, es decir, se tiene previamente un banco de pruebas de los cuales el módulo programado o la iteración debe superar para poder considerarse como un entregable, dado esto, el desarrollo también puede centralizarse a esta, definida como pruebas unitarias.

En esta metodología, por misma que puede llegar a ser muy cambiante, es recomendable tener siempre publicado las nuevas versiones del entregable, con el fin de evitar inconvenientes en integraciones más grandes o globales del software.

El entregable en esta etapa es el resultado de la iteración reflejado en un producto, es decir, el software.

2.6.1.5. Fase de pruebas

Como en toda metodología de desarrollo de software podemos definir varios tipos de pruebas que pueden ser aplicados al fin de establecer la calidad de funcionamiento del producto.

- Identificación y depuración de errores.

La ideología se centra en el trabajo empírico, es decir, cada vez que se encuentre un error dentro del código fuente, este debe

ser depurado al instante, y esta será base y experiencia para la identificación de futuros errores en todo el sistema.

- Pruebas de funcionalidad y aceptación de software.

Puesto que las funcionalidades fueron generadas en base a las historias de usuario y viceversa y estos a su vez, fueron generadas en conjunto con el cliente o usuario final, este debe promover los distintos escenarios o condiciones críticas y reales para el sistema sea puesto a prueba.

Las pruebas de aceptación pueden ser consideradas como en otras metodologías como pruebas de Caja Negra, es decir, se escogerá un grupo selecto de personas, de preferencia conocedores del proceso operativo que se está automatizando para poder utilizar la versión de prueba del software. Una vez satisfecho y aprobado este banco de pruebas se puede considerar una nueva versión de entregable o producto de software.

2.6.1.6. Fase de implementación

Una vez teniendo una consideración final y satisfactoria del software puede darse por finalizado el periodo de iteraciones para dar lugar a la instalación del sistema junto con los escenarios y recursos idóneos para su correcto funcionamiento.

2.7. Enfoque del aplicativo

Dentro del área de ventas de cualquier entidad corporativa existen varias necesidades que pueden ser imperceptibles en primera instancia por los usuarios finales, estos pueden ser aprovechados para generar valores agregados a una empresa y potenciar la calidad de servicio de esta, mejorando la imagen corporativa.

En muchos giros de negocio tales como servicio a domicilio, oferta de productos y servicios puerta a puerta, transporte de productos, etc., se puede generar controles de administración en el flujo de trabajo que realiza el personal y la gestión del tiempo que se está empleando sobre el mismo.

Con el control de personal, activos y mercadería asistido por GPS, es posible garantizar la calidad del servicio que se está ejecutando, de igual manera se tiene un registro de todos los inconvenientes y errores generados durante los procesos de servicios de transporte.

De esta manera, se puede obtener la información suficiente para optimizar tiempos de respuesta en un proceso operativo de ventas, evitar cuellos de botella que puedan generar costes adicionales e innecesarios.

Si se logra optimizar y mejorar continuamente estos procesos, el empresario tendrá apertura a incrementar su rentabilidad al generar menos costes por errores de procesos, al mismo tiempo incrementará demanda de mercado por incrementar su calidad servicio.

En base a estas necesidades se desarrollará una aplicación que será evolutiva a cualquier tipo de negocio, en cuyos procesos constituyan el transporte de productos o servicios a domicilios.

Capítulo III: Metodología de desarrollo XP

En el presente capítulo se levantará toda la información, requerimientos y documentación necesaria para el proceso de desarrollo de software según nos sugiere los lineamientos de la metodología de Desarrollo XP.

3.1. Fase de exploración

3.1.1. Identificación de actores

En base a reuniones técnicas con el usuario final y líder de proyecto se ha logrado especificar los siguientes usuarios y funcionalidades dentro del que será la aplicación de Software.

3.1.1.1. Administrador

Usuario encargado de manipular toda la información con respecto a los clientes creados en la aplicación. Revisar la posición y rutas realizadas de cada miembro de su red, también recibir solicitudes de verificación de posición de usuario cliente.

Además de tener funcionalidades de mensajería instantánea.

3.1.1.2. Cliente

Son usuarios que han sido definidos y creados previamente por un administrador en específico.

Los usuarios clientes tendrán la habilidad de poder enviar y recibir mensajes a sus contactos en una red concreta de contactos, de igual manera registrarán su ruta GPS siempre y

cuando lo hayan autorizado previamente, en el caso de necesitar notificar su ubicación actual, podrán realizarlo mediante solicitudes al administrador.

3.1.2. Historias de usuario

En este apartado se muestra descripciones breves y sencillas que reflejan cada una de las funcionalidades del software, las mismas que serán establecidas por el cliente o usuario final, estas descripciones no serán técnicas necesariamente, sin embargo, dicho concepto será transformado a fichas técnicas por el equipo de desarrollo.

3.1.2.1. Inicio de sesión administrador

Tabla 3-1 Inicio de sesión administrador

Historia de Usuario			
Número: 01		Nombre de Historia:	Acceso y rol de usuarios administrativos
Usuario:	Administrador		
Prioridad en Negocio: (Alta/Media/Baja	Media	Riesgo en Desarrollo: (Alta/Media/Baja)	Media
Descripción			
Los usuarios que tendrán control total sobre los usuarios clientes serán considerados con un rol de "Administrador"			
Administrador			
EL usuario Administrador será creado previamente por un Super Usuario, dueño de la aplicación, este administrador deberá tener sus credenciales para poder ingresar a la aplicación y gestionar información de todos sus usuarios clientes (CRUD).			

Elaborado Por: David Salgado

3.1.2.2. Inicio de sesión cliente

Tabla 3-2 Inicio de sesión cliente

Historia de Usuario			
Número: 02		Nombre de Historia:	Acceso y rol de usuarios clientes
Usuario:	Administrador		
Prioridad en Negocio: (Alta/Media/Baja	Media	Riesgo en Desarrollo: (Alta/Media/Baja)	Media
Descripción			
Los usuarios que estén registrados y tengan como objetivo ser monitoreados y cuya información ser administrada se considera Usuario Cliente			
Administrador			
EL usuario Cliente será creado previamente por un Usuario Administrador, este cliente deberá tener sus credenciales para poder ingresar a la aplicación, las mismas que serán entregadas vía correo después de haberse registrado en la aplicación por primera vez, la información de estos es de libre manipulación por los usuarios Administradores.			

Elaborado Por: David Salgado

3.1.2.3. Gestión CRUD de usuario cliente

Tabla 3-3 Gestión CRUD de usuario cliente

Historia de Usuario			
Número: 03		Nombre de Historia:	Gestión de Usuarios Cientes
Usuario:	Administrador		
Prioridad en Negocio: (Alta/Media/Baja	Media	Riesgo en Desarrollo: (Alta/Media/Baja)	Media
Descripción			
Funcionalidad donde los administradores podrán crear, modificar, eliminar y buscar usuarios clientes, así mismo podrán realizar consultas de datos referenciales como rutas, posiciones referenciales y mensajes de texto.			

Elaborado Por: David Salgado

3.1.2.4. Módulo de georeferencia

Tabla 3-4 Módulo de georeferencia

Historia de Usuario			
Número: 04		Nombre de Historia:	Módulo de Geo Referencia
Usuario:	Administrador / Cliente		
Prioridad en Negocio: (Alta/Media/Baja	Alta	Riesgo en Desarrollo: (Alta/Media/Baja)	Media
Descripción			
Mediante una interfaz gráfica tanto administradores como clientes podrán monitorear sea su ubicación actual GPS como la ruta que han recorrido previa autorización de un usuario cliente, de igual manera podrán notificar a un administrador su ubicación en el caso que sea necesario.			

Elaborado Por: David Salgado

3.1.2.5. Módulo de mensajería instantánea

Tabla 3-5 Módulo de mensajería instantánea

Historia de Usuario			
Número: 05		Nombre de Historia:	Módulo de Mensajería Instantánea
Usuario:	Administrador / Cliente		
Prioridad en Negocio: (Alta/Media/Baja	Alta	Riesgo en Desarrollo: (Alta/Media/Baja)	Media
Descripción			
Tanto los usuarios administradores como sus respectivos clientes pueden enviar mensajes de texto entre sí. Cada administrador tendrá su propia red de contactos, los cuales serán compartidos entre sus propios clientes, pero no entre otros administradores.			

Elaborado Por: David Salgado

3.1.2.6. Módulo de reportes

Tabla 3-6 Módulo de reportes

Historia de Usuario			
Número: 06		Nombre de Historia:	Módulo de Reportes
Usuario:	Administrador / Cliente		
Prioridad en Negocio: (Alta/Media/Baja	Media	Riesgo en Desarrollo: (Alta/Media/Baja)	Media
Descripción			
Los reportes contendrán información sobre las rutas realizadas por los clientes en una determinada línea del tiempo. De igual manera se podrá visualizar las notificaciones de ubicaciones referenciales enviadas al administrador. Los clientes podrán visualizar sus propios reportes mientras que los administradores podrán hacerlo de todos sus clientes creados.			

Elaborado Por: David Salgado

3.1.2.7. Función de registro (Check-In)

Tabla 3-7 Función de registro (Check-In)

Historia de Usuario			
Número: 07		Nombre de Historia:	Función Check-In
Usuario:	Administrador / Cliente		
Prioridad en Negocio: (Alta/Media/Baja	Media	Riesgo en Desarrollo: (Alta/Media/Baja)	Media
Descripción			
Los clientes tienen la habilidad de notificar su ubicación a su respectivo administrador mediante un botón "Check-In", el cual será automáticamente reportado y guardado en un registro para futuros reportes.			

Elaborado Por: David Salgado

3.1.3. Análisis de iteraciones

Una vez definido todas las funcionalidades, roles e historias de usuario para el proceso de desarrollo de software, se realizará el respectivo análisis para obtener tiempos de desarrollo adecuados para la generación de los entregables en cada

iteración, con el fin de sostener de una manera estable el flujo de trabajo y cumplir con el plan de entrega sugerido por la metodología de desarrollo de software.

3.1.3.1. Estimación de horas por historia de usuario

Se estiman las horas para desarrollar las historias de usuario en referencia a la siguiente información:

No. de desarrolladores: 1

Horas desarrollo por día: 8

Días por Semana: 5

Con las variables antes mencionadas se realiza el siguiente cálculo para obtener las horas promedio adecuadas de desarrollo a la semana.

1 persona x 8 horas x 5 días = 40 horas / semana

Tabla 3-8 Promedio de horas para el desarrollo de las historias de usuario

No.	Historia de Usuario	Riesgo	Horas Promedio
1	Acceso y rol de usuarios Administrativos	Media	30
2	Acceso y rol de usuarios Clientes	Media	20
3	Gestión de Usuarios Clientes	Media	20
4	Módulo de Geo Referencia	Alta	80
5	Módulo de Mensajería Instantánea	Alta	80
6	Módulo de Reportes	Media	20
7	Función de Check-In	Baja	10
8	Banco de Pruebas	Alta	40
	TOTAL		300

Elaborado Por: David Salgado

Tal como se muestra en la tabla 3-8, es factible el desarrollo de la aplicación puesto que se realiza un estimado de 2 meses, dentro de este rango existe un total de 320 horas de las cuales se necesitan 300 para la programación de todos los módulos más el periodo de banco de pruebas respectivo, el resto de las horas se las consideran aplicadas en caso de inconvenientes y retrasos de procesos de implementación no planeados.

3.1.3.2. Evaluación de prioridades

Dentro de la priorización con respecto a las historias de usuario se debe evaluar y clasificar la importancia de cada una de ellas, se planteará en la siguiente escala:

Alta: Estos módulos son de vital importancia, por tanto, deben ser implementados a la brevedad posible.

Media: Son consideradas importantes después de su antecesor.

Baja: Son consideradas las menos importantes, no obstante, deben implementarse después de su antecesor

A continuación, se muestra la clasificación en función a la nomenclatura antes mencionada.

Tabla 3-9 Priorización de casos de uso

Priorización de Historia de Usuarios

Alta		Media		Baja	
No.	Historia	No.	Historia	No.	Historia
4	Módulo de Geo Referencia	1	Acceso y rol de usuarios administrativos		
5	Módulo de Mensajería Instantánea	2	Acceso y rol de usuarios clientes		
		3	Gestión de usuarios clientes		
		6	Módulo de Reportes		
		7	Función de Check-In		

Elaborado Por: David Salgado

3.1.4. Casos de uso

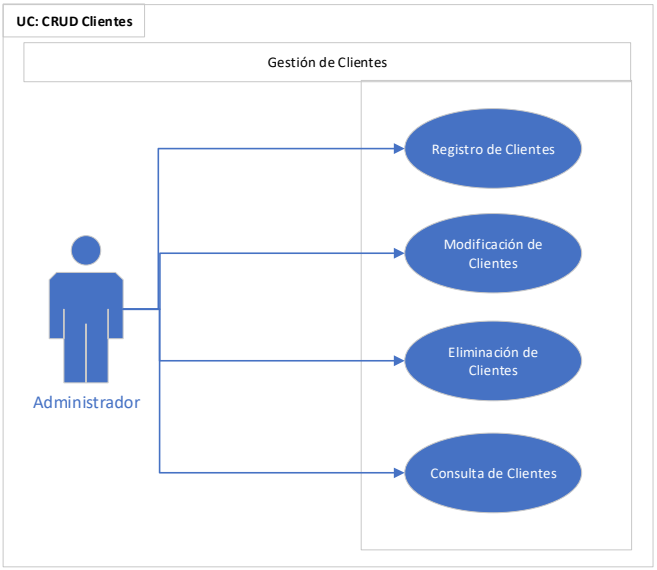
3.1.4.1. Usuarios

Diagrama 3-1 Usuarios

Elaborado Por: David Salgado

3.1.4.2. CRUD Clientes

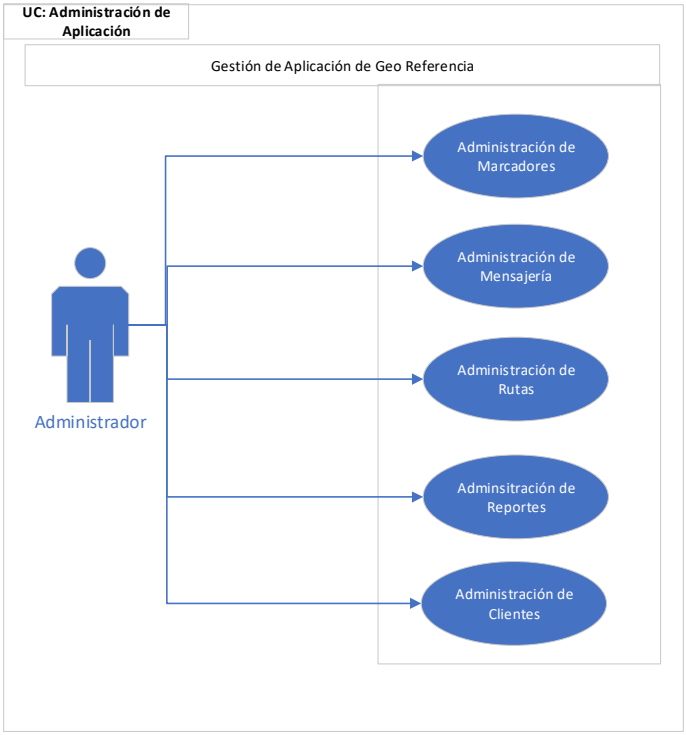
Diagrama 3-2 CRUD Clientes



Elaborado Por: David Salgado

3.1.4.3. Georeferencia

Diagrama 3-3 Georeferencia



Elaborado Por: David Salgado

3.1.5. Análisis y detalle de historias de usuario

3.1.5.1. Acceso y rol de usuario administrador

Tabla 3-10 Acceso y rol de usuario administrador

Nombre	Acceso y rol de usuarios administrativos	
Código:	HU001	
Actor:	Administrador	
Descripción:	El usuario Administrador será creado previamente por un Super Usuario, dueño de la aplicación, este administrador deberá tener sus credenciales para poder ingresar a la aplicación y gestionar información de todos sus usuarios clientes (CRUD)	
Precondiciones:	Se debe definir un administrador para poder registrarlo	
Post-Condiciones:	Se inicia la apertura de creación de red de clientes para el Administrador creado	
Flujo Principal:	ACTOR	SISTEMA
	Super Usuario inicia el proceso de creación de Administrador	2. El sistema crea un nuevo perfil con rol Administrador en la Base de datos
		3. El sistema define y genera las nuevas credenciales para el Administrador.
	5. El administrador inicia sesión en su aplicación móvil con las credenciales enviadas	El Sistema envía las credenciales al correo proporcionado por el Administrador
		6. El sistema reconoce las credenciales de la Base de datos y aprueba el acceso del Administrador
Flujo alternativo:		5.1 El sistema genera excepción por motivos de credenciales incorrectas
		5.1 El sistema pide al usuario revisar las credenciales enviadas y brinda nueva apertura para el inicio de sesión
Resultado:	El Administrador tiene acceso a la aplicación para la administración de clientes	

Elaborado Por: David Salgado

3.1.5.2. Acceso y rol de usuario cliente

Tabla 3-11 Acceso y rol de usuario cliente

Nombre	Acceso y rol de usuarios clientes	
Código:	HU002	
Actor:	Cliente	
Descripción:	El usuario Cliente será creado previamente por un usuario Administrador, este cliente deberá tener sus credenciales para poder ingresar a la aplicación, las mismas que serán entregadas vía correo después de haberse registrado en la aplicación por primera vez, la información de estos es de libre manipulación por los Administradores	
Precondiciones:	Se debe definir un cliente para registrarlo	
Post-Condiciones:	Se autoriza la red del administrador para la creación de clientes adicionales.	
Flujo Principal:	ACTOR	SISTEMA
	Administrador inicia el proceso de creación de clientes.	2. El sistema crea un nuevo perfil con rol de Cliente en la Base de datos
		3. El sistema define y genera las nuevas credenciales para el Cliente.
	5. El cliente inicia sesión en su aplicación móvil con las credenciales enviadas	4. El Sistema envía las credenciales al correo proporcionado por el Administrador
		6. El sistema reconoce las credenciales de la Base de datos y aprueba el acceso con rol de cliente
Flujo alternativo:		5.1 El sistema genera excepción por motivos de credenciales incorrectas
		5.1 El sistema pide al usuario revisar las credenciales enviadas y brinda nueva apertura para el inicio de sesión
Resultado:	El cliente tiene acceso a la aplicación para que pueda ser monitoreado.	

Elaborado Por: David Salgado

3.1.5.3. Gestión de clientes

3.1.5.3.1. Ingreso de clientes

Tabla 3-12 Ingreso de clientes

Nombre	Ingreso de usuarios clientes	
Código:	HU0031	
Actor:	Administrador	
Descripción:	El Administrador ingresa un nuevo usuario cliente mediante un formulario por parte de la aplicación móvil	
Precondiciones:	El administrador debe estar registrado y con inicio de sesión en el sistema	
Post-Condiciones:		
Flujo Principal:	ACTOR	SISTEMA
	1. El Administrador accede a la pantalla de Ingreso de clientes.	2. El Sistema presenta el formulario para crear un cliente.

	3. El Administrador ingresa los datos del usuario cliente.	
	4. El Administrador presiona ¨Crear Usuario¨.	5. El Sistema crea el nuevo cliente en la base de datos.
		6. El sistema envía las credenciales generadas al correo del cliente.
Flujo alternativo:		5.1 El sistema genera excepción por información incorrecta
		5.1 El Sistema pide de nuevo el ingreso de información del cliente al Administrador
Resultado:	El usuario cliente ha sido ingresado.	

Elaborado Por: David Salgado

3.1.5.3.2. Visualización de cliente

Tabla 3-13 Visualización de cliente

Nombre	Visualización de usuarios Clientes	
Código:	HU0032	
Actor:	Cualquier actor	
Descripción:	Cualquier usuario puede visualizar los contactos de la red de un usuario Administrador	
Precondiciones:	El administrador y por lo menos un cliente deben estar registrados y con inicios de sesión.	
Post-Condiciones:		
Flujo Principal:	ACTOR	SISTEMA
	1. El Usuario ingresa a la pantalla de contactos.	2. El Sistema presenta la tabla de contactos totales.
		3. El Sistema se conecta a la base de datos y recupera los contactos de la red del respectivo administrador.
		4. El sistema despliega todos los contactos obtenidos de la base de datos.
	5. El usuario visualiza todos los contactos proveídos por el sistema.	
Flujo alternativo:		3.1 El sistema muestra un mensaje de error por desconexión de internet.
	3.2. El usuario inicia proceso de conexión del dispositivo a internet.	3.3. El sistema intenta nuevamente recuperar los contactos de la base de datos.
Resultado:	El usuario puede visualizar los contactos de la red del administrador.	

Elaborado Por: David Salgado

3.1.5.3.3. Modificación de cliente

Tabla 3-14 Modificación de cliente

Nombre	Modificación de información de usuario cliente.	
Código:	HU0033	
Actor:	Cualquier actor	
Descripción:	Cualquier usuario puede modificar su propio perfil mediante la pantalla de gestión de información de perfil	
Precondiciones:	El administrador y por lo menos un cliente deben estar registrados y con inicios de sesión.	
Post-Condiciones:		
Flujo Principal:	ACTOR	SISTEMA
	1. El Usuario ingresa a la pantalla de información de perfil.	2. El sistema presenta la pantalla de perfil de usuario.
		3. El sistema presenta la información guardada en función al perfil del usuario que ha iniciado sesión.
	4. El usuario se dispone a modificar la información necesaria de su perfil.	
	5. El usuario presiona el botón de ¨Guardar¨.	6. El sistema valida la información ingresada por el usuario.
		7. El sistema se conecta con la base de datos y se dispone a modificar la información de perfil.
		8. El sistema presenta notificación de información actualizada correctamente.
Flujo alternativo:		6.1. El sistema presenta mensaje de error por información mal ingresada.
	6.2. El usuario vuelve a ingresar la información necesitada.	6.3. El sistema se dispone a validar de nuevo la información ingresada por el usuario.
Resultado:	La información del usuario es actualizada.	

Elaborado Por: David Salgado

3.1.5.3.4. Eliminación de usuario

Tabla 3-15 Eliminación de usuario

Nombre	Eliminación de usuarios cliente	
Código:	HU0034	
Actor:	Administrador	
Descripción:	El usuario Administrador tiene la capacidad de eliminar usuarios clientes mediante la pantalla de ingreso de clientes	
Precondiciones:	El administrador y por lo menos un cliente deben estar registrados.	
Post-Condiciones:		
Flujo Principal:	ACTOR	SISTEMA
	1. El Administrador ingresa a la pantalla de ingreso de usuarios clientes.	2. El Sistema presenta pantalla de ingreso de clientes con lista de clientes creados.

	3. El Administrador visualiza y determina el usuario cliente objetivo para eliminar.	
	4. El Administrador presiona el botón ¨Elimina¨.	5. El sistema presenta mensaje para confirmar la acción de eliminado de usuario.
	6. El Administrador presiona el botón de confirmación para borrado de usuario.	7. El Sistema se conecta con la base de datos y elimina el usuario objetivo.
Flujo alternativo:		2.1. y 7.1 El sistema presenta mensaje de error por desconexión de internet.
	2.2. y 7.2. El Administrador vuelve a conectar el dispositivo a internet.	2.3. y 7.3. El sistema se conecta a la base datos para realizar la acción requerida.
Resultado:	La información del usuario cliente es eliminada de la base de datos.	

Elaborado Por: David Salgado

3.1.5.4. Módulo de georeferencia

3.1.5.4.1. Activar geolocalización

Tabla 3-16 Activar geolocalización

Nombre	Activar geolocalización	
Código:	HU00421	
Actor:	Cliente	
Descripción:	El cliente activa la geolocalización y envía la coordenada actual.	
Flujo Principal:	ACTOR	SISTEMA
	1. El cliente presiona el switch de permitir tracking	2. El sistema enciende la geolocalización del equipo.
		3. El sistema empieza a enviar las coordenadas de latitud y longitud hacia la base de datos
		4. El pin de localización aparece en el mapa del Administrador
Flujo Alternativo		2.1. Si la geolocalización ya se encuentra encendida, será desactivada
Resultado:	Las coordenadas se empiezan a enviar a la base de datos y se realiza el tracking del cliente	

Elaborado Por: David Salgado

3.1.5.4.2. Desactivar geolocalización

Tabla 3-17 Desactivar geolocalización

Nombre	Desactivar geolocalización	
Código:	HU00422	
Actor:	Cliente	
Descripción:	El cliente desactiva la geolocalización	
Flujo Principal:	ACTOR	SISTEMA
	1. El cliente presiona el switch de permitir tracking.	2. El sistema desactiva la geolocalización del equipo
Flujo Alternativo		2.1. Si la geolocalización ya se encuentra encendida, será desactivada
Resultado:	El proceso de geolocalización se detiene.	

Elaborado Por: David Salgado

3.1.5.5. Módulo de mensajería instantánea

3.1.5.5.1. Enviar mensaje

Tabla 3-18 Enviar mensaje

Nombre	Enviar mensaje	
Código:	HU00431	
Actor:	Cualquier actor	
Descripción:	El actor envía un mensaje hacia otro equipo	
Flujo Principal:	ACTOR	SISTEMA
		1.El sistema muestra todos los actores a los cuales se puede enviar un mensaje
	2. El actor selecciona del listado de usuarios a quien desea enviar el mensaje	3.El sistema abre la pantalla de chat con el actor seleccionado
	4. El actor escribe en la caja de texto el mensaje a enviar	5. El Sistema envía el mensaje hacia la base de datos.
	6. El actor presiona el botón de enviar mensaje.	
Resultado:	El actor ha enviado un mensaje hacia otro actor de manera exitosa.	

Elaborado Por: David Salgado

3.1.5.5.2. Recibir mensaje

Tabla 3-19 Recibir mensaje

Nombre	Recibir mensaje	
Código:	HU00432	
Actor:	Cualquier actor	
Descripción:	El actor recibe un mensaje	
Flujo Principal:	ACTOR	SISTEMA
		1.El sistema verifica la existencia de nuevos mensajes
		2. El sistema actualiza los mensajes nuevos y muestra notificación de nuevos mensajes
	3.El actor abre la pantalla que tiene mensajes nuevos	4.El sistema abre la pantalla del actor seleccionado
		5.El sistema muestra todos los mensajes entre ambos actores
Flujo alternativo:		1.1 Si el sistema no encuentra mensajes nuevos, no se realiza ninguna acción
Resultado:	El actor recibe exitosamente un mensaje de otro actor	

Elaborado Por: David Salgado

3.1.5.6. Módulo de reportes

3.1.5.6.1. Visualización de marcadores

Tabla 3-20 Visualización de marcadores

Nombre	Visualización de marcadores	
Código:	HU00441	
Actor:	Cualquier actor	
Descripción:	El actor observa sus marcadores si es cliente o todos los marcadores si es administrador	
Flujo Principal:	ACTOR	SISTEMA
	1.El actor ingresa a la pantalla visualización de marcadores	2.El sistema carga todos los marcadores dependiendo del rol
		3.El sistema despliega los marcadores en una lista con la información necesaria
Resultado:	El actor visualiza sus marcadores	

Elaborado Por: David Salgado

3.1.5.6.2. Visualización de marcadores en mapa

Tabla 3-21 Visualización de marcadores en mapa

Nombre	Visualización de marcadores en mapa	
Código:	HU00442	
Actor:	Cualquier actor	
Descripción:	El actor observa sus marcadores si es cliente o todos los marcadores si es administrador	
Flujo Principal:	ACTOR	SISTEMA
	1.El actor ingresa a la pantalla visualización de marcadores	2.El sistema carga todos los marcadores dependiendo del rol
		3.El sistema despliega los marcadores en una lista con la información necesaria
	4.El actor presiona en uno de sus marcadores	5.El sistema muestra en un mapa los marcadores existentes
Resultado:	El actor visualiza sus marcadores en el mapa	

Elaborado Por: David Salgado

3.1.5.7. Función de registro (Check-In)

3.1.5.7.1. Enviar localización de forma manual

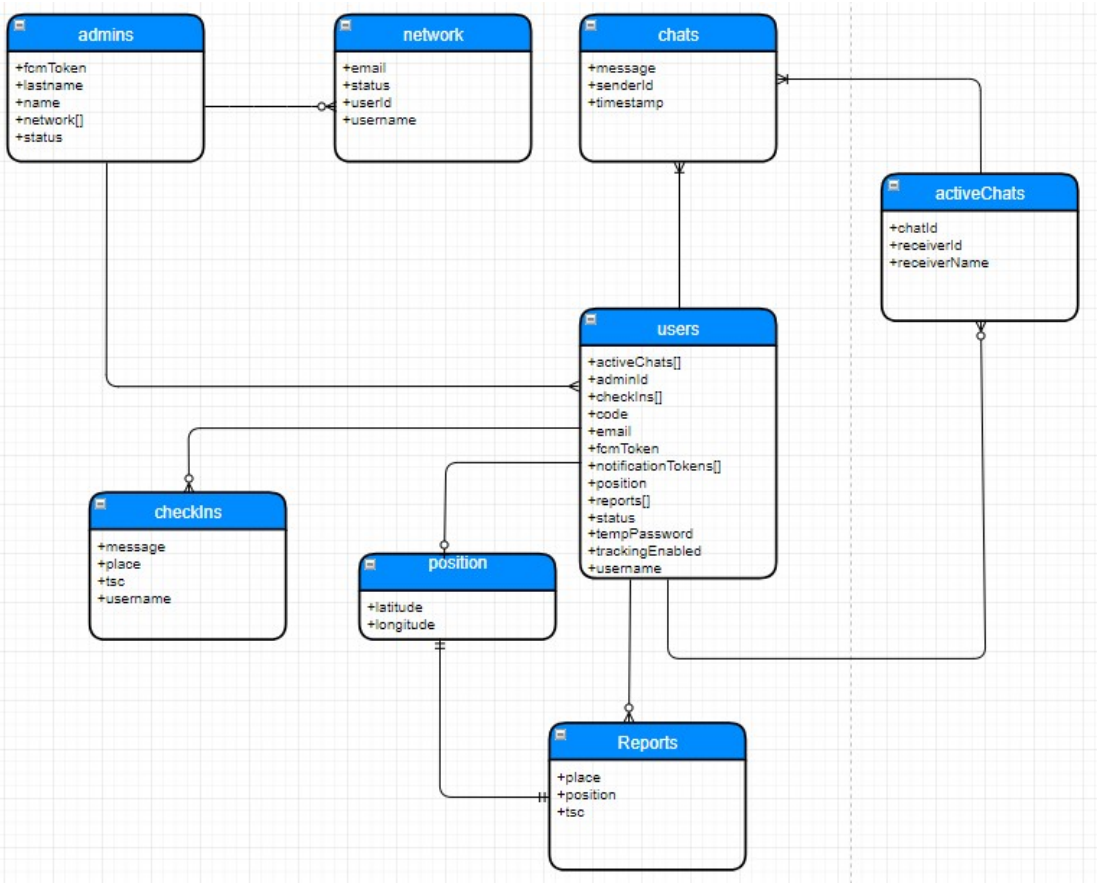
Tabla 3-22 Enviar localización de forma manual

Nombre	Enviar localización de forma manual	
Código:	HU00451	
Actor:	Cliente	
Descripción:	El cliente presiona el botón check-in para enviar localización	
Flujo Principal:	ACTOR	SISTEMA
	1.El actor ingresa a la pantalla del mapa	2.El sistema muestra la pantalla del mapa
	2.El actor presiona el botón de check-in	3.El sistema obtiene las coordenadas actuales del actor
		4.El sistema convierte las coordenadas en dirección a través de un método
		5.El sistema envía las coordenadas al administrador.
	6.El administrador visualiza en la pantalla principal el check-in del actor	
Resultado:	El actor envía su localización	

Elaborado Por: David Salgado

3.1.6. Diagrama conceptual

Diagrama 3-4 Diagrama conceptual de base de datos



Elaborado Por: David Salgado

Capítulo IV: Implementación

El presente capitulo trata sobre los recursos utilizados durante la implementación y el proceso técnico que se llevo a cabo para construir el aplicativo móvil.

4.1. Recursos utilizados

Para el presente proyecto de desarrollo de software se ha establecido necesario el uso de los siguientes recursos.

4.1.1. Software

En la siguiente tabla se expone los elementos de software que serán utilizados para desarrollar el proyecto.

Tabla 4-1 Recursos de software utilizados para el proyecto de desarrollo

Item.	Descripción	Software
1	IDE SW para desarrollo de APP	Xcode
2	Motor de Base de Datos	Firebase
3	Tipo de Base de Datos	No relacional
4	Sistema Operativo de desarrollo	OS X

Elaborado Por: David Salgado

4.1.2. Hardware

En la siguiente tabla se expone los elementos de hardware que serán utilizados para desarrollar el proyecto.

Tabla 4-2 Recursos de hardware utilizados para el proyecto de desarrollo

Item.	Descripción	Software
1	Computador para desarrollo de SW	MacBook Pro, Procesador Intel core i5, Memoria de 8 GB HDD 128 GB.
2	Servicio de almacenamiento en nube	Firebase
3	Dispositivo Móvil	Iphone 5

Elaborado Por: David Salgado

4.2. Tabla de Costos

El presente es un análisis te costos tentativo del proyecto que muestra los valores de los recursos humanos, hardware, software y costos varios.

Tabla 4-3 Costos durante el proceso de desarrollo de software

No.	Item	Cant / Mes	Costo
	Recursos Humanos		
1	Desarrollador	2	1400
	Recursos HW		
2	Dispositivo Móvil	1	600
3	Laptop	1	1400
	Recursos SW		
4	IDE Xcode	1	100
5	Firebase	1	0
	Otros		
6	Servicios Básicos	2	520
7	Transportes	2	60
8	Servicio de Internet	2	150
9	Papelería	2	100
	TOTAL		4330

Elaborado Por: David Salgado

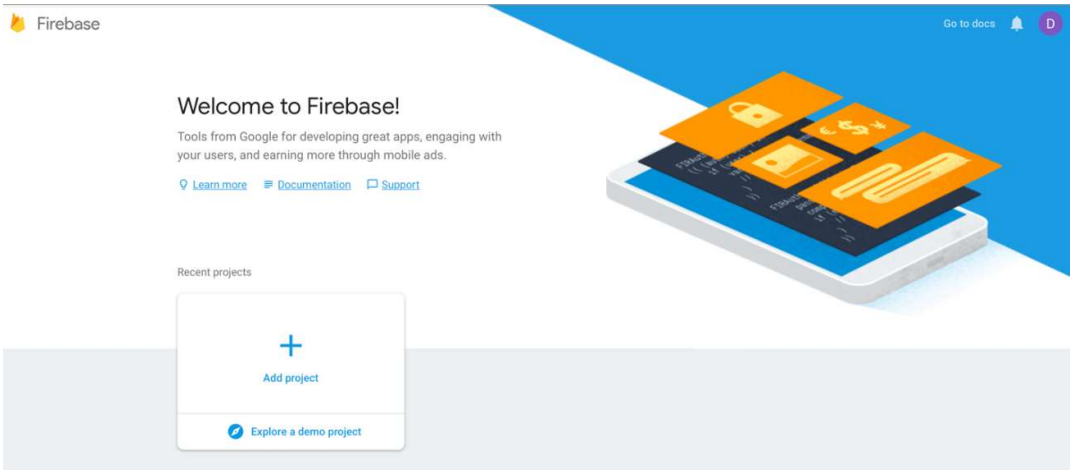
4.3. Configuración de Firebase

Para la configuración de Firebase en el presente proyecto hay que tomar en cuenta tres aspectos: la configuración del proyecto, la base de datos y los métodos de autenticación.

4.3.1. Configuración del proyecto y base de datos

Para la realización del proyecto inicialmente es necesario configurar tanto el proyecto de Xcode como el backend (Firebase) contra el cual se va a conectar el aplicativo móvil. Para esto se debe crear una cuenta de Google la cual debe tener acceso a Firebase Console ingresando al URL "https://console.firebase.google.com", donde será creado el proyecto.

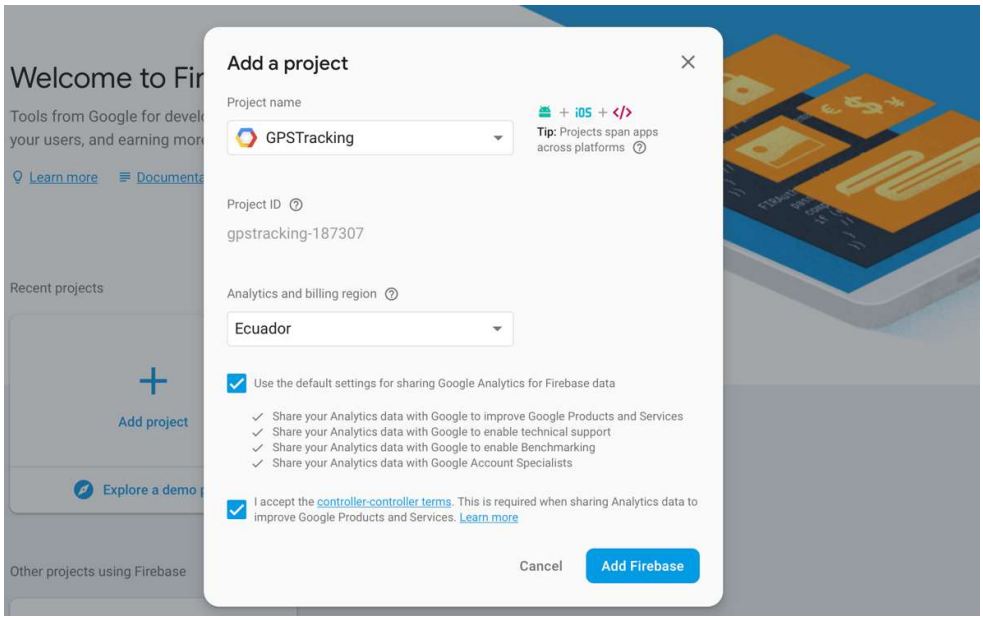
Diagrama 4-1 Vista de la consola de Firebase



Elaborado Por: David Salgado

El diagrama 4-1 nos indica cómo se visualiza la consola de Firebase vacía y lista para que podamos crear el nuevo proyecto. Para realizar esto, se debe dar clic en "Add Project" e ingresar los datos referentes al proyecto que son: nombre del proyecto, país de facturación y aceptar los términos y condiciones de Firebase como se observa en el diagrama 4-2.

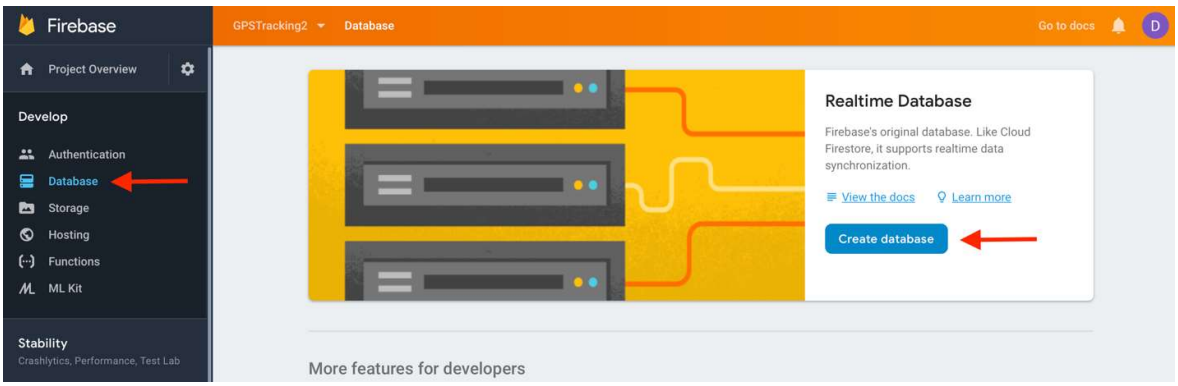
Diagrama 4-2 Vista de la pantalla de creación del proyecto en Firebase



Elaborado por: David Salgado

Una vez llenos estos datos, se agrega el proyecto a Firebase presionando “Add Firebase”. Cuando el proyecto se encuentre creado se debe crear una base de datos donde toda la información de la aplicación se va a almacenar dirigiéndose a la pestaña de “Database” y presionar el botón de “Create database” como se ve en el diagrama 4-3.

Diagrama 4-3 Vista de la pestaña de “Database”

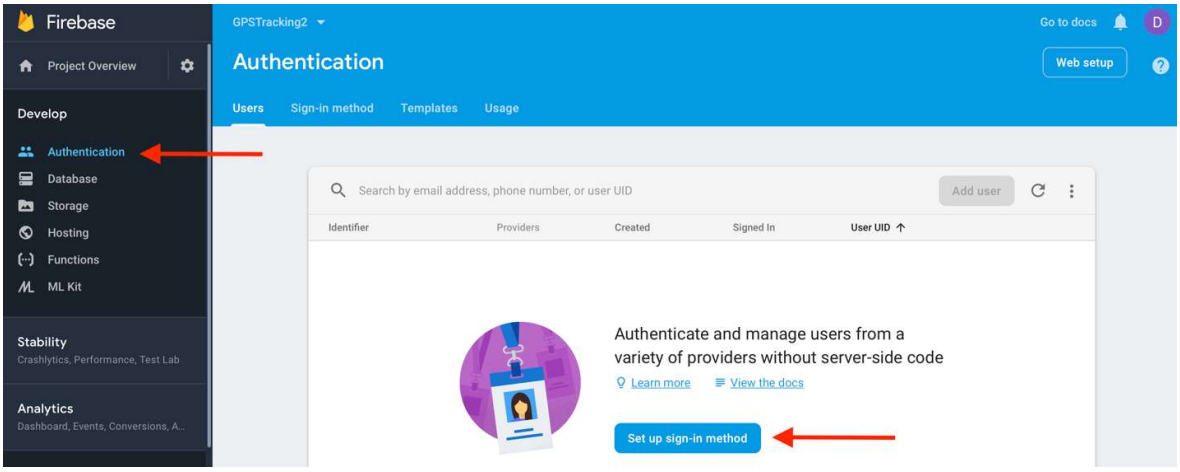


Elaborado por: David Salgado

4.3.2. Configuración de la autenticación

Firebase también cuenta con un servicio para autenticación, para activarlo es necesario ir a la pestaña de "Authentication" y agregar un método para iniciar sesión presionando el botón "Set up sign-in method".

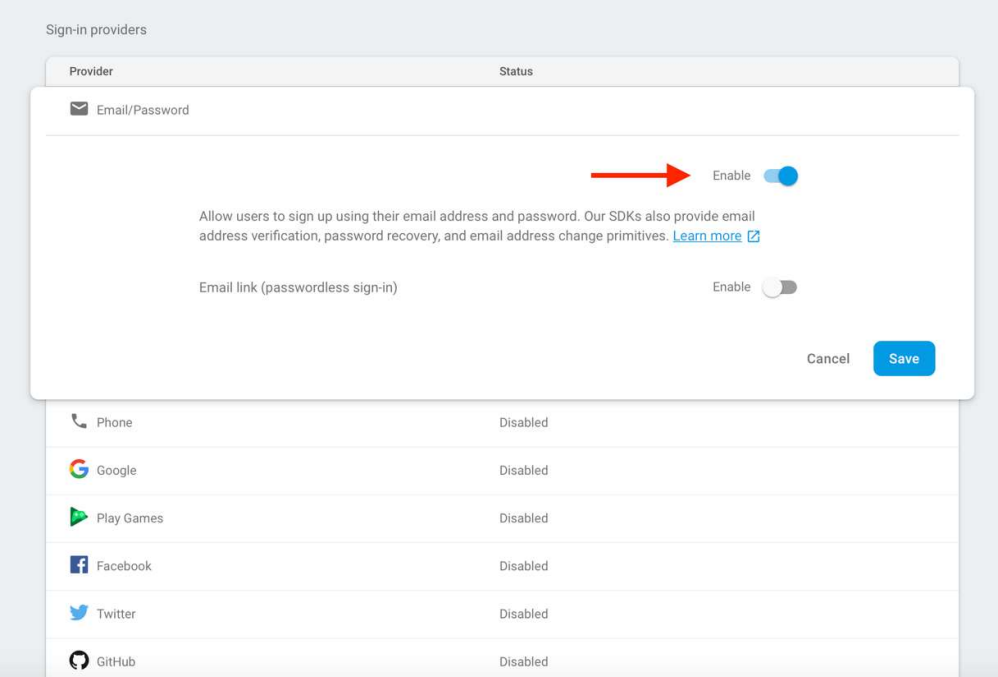
Diagrama 4-4 Vista de la pestaña de "Authentication"



Elaborado por: David Salgado

Entre los distintos métodos de autenticación se va a elegir y habilitar el método de "Email/Password", el cual permite a la aplicación registrar el correo del usuario; y mediante una contraseña validar el inicio de sesión como se ve en el diagrama 4-5.

Diagrama 4-5 Vista de los métodos de autenticación



Elaborado por: David Salgado

4.4. Configuración del proyecto de iOS

Para configurar el proyecto en iOS como se puede ver en el diagrama 4-6 se requieren las librerías de Firebase, que son usadas para la comunicación con los servicios creados tanto para base de datos, notificaciones push y autenticación. Para interactuar con Google Maps es necesaria la librería de mapas y geolocalización y Google Places que se usa para transformar la latitud y longitud en direcciones legibles. Estas librerías se las instala mediante el manejador de dependencias "pod", incluyéndolas en el "podfile" del proyecto y correr el comando "pod install" en la raíz del proyecto. Las otras dependencias usadas proveen de componentes para el desarrollo del interfaz del aplicativo y no es necesario incluirlas.

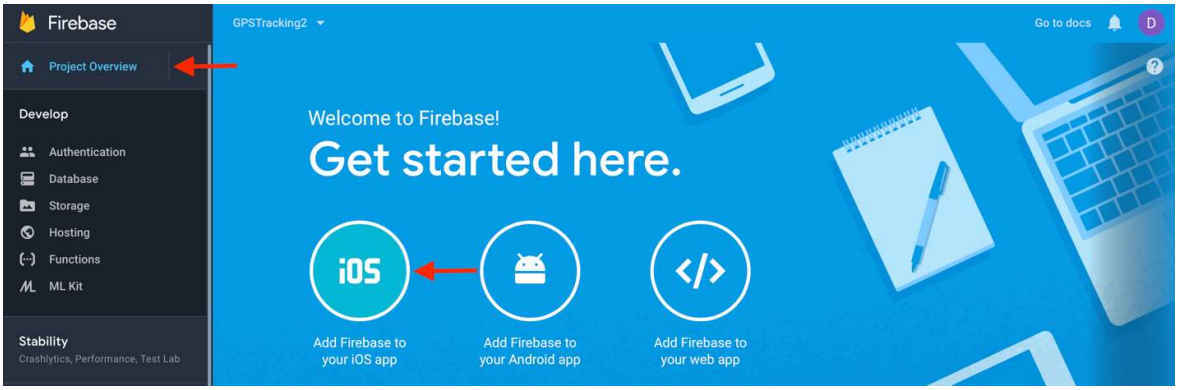
Diagrama 4-6 Vista del archive podfile con las dependencias necesarias

```
# Uncomment the next line to define a global platform for your project
platform :ios, '10.0'

target 'GPSTracking' do
  # Comment the next line if you're not using Swift and don't want to use dynamic
frameworks
  use_frameworks!

  # Pods for GPSTracking
    pod 'Firebase/Core'
    pod 'Firebase/Auth'
    pod 'Firebase/Database'
    pod 'Firebase/Messaging'
    pod 'GoogleMaps'
    pod 'GooglePlaces'
    pod 'InteractiveSideMenu'
    pod 'RAMPaperSwitch'
    pod 'PopupDialog', '~> 0.6'
    pod 'GrowingTextView', '~> 0.5.3'
    pod 'lottie-ios'

end
```

Elaborado por: David Salgado

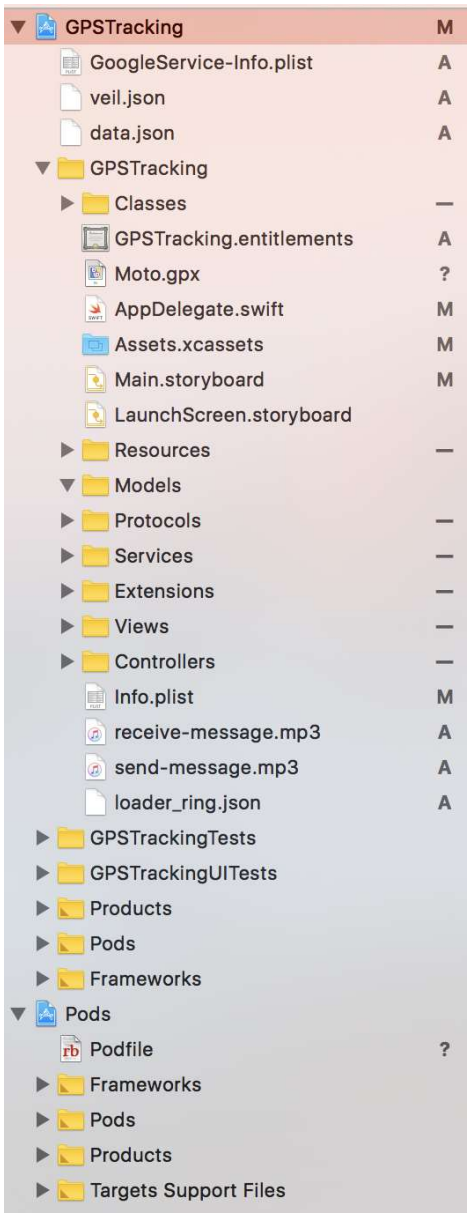
Para integrar los servicios de Firebase en el aplicativo se requiere del archivo GoogleService-Info.plist el cual contiene las configuraciones de Firebase para el proyecto iOS como se ve en el diagrama 4-7, este se lo descarga desde la consola de Firebase en “Project Overview” presionando el botón “Add Firebase to iOS app”, registrando nuestro proyecto de iOS y descargando el “config file”, el cual se coloca en la raíz del proyecto, terminando con una estructura del proyecto como se ve en el diagrama 4-8.

Diagrama 4-7 Vista de la pantalla de “Project Overview”



Elaborado por: David Salgado

Diagrama 4-8 Estructura del proyecto en Xcode



Elaborado por: David Salgado

4.5. Implementación del módulo de autenticación

La interfaz de la pantalla de autenticación se aprecia en el diagrama 4-9. Esta es parte del módulo de ingreso al sistema que van a usar tanto usuarios de tipo administrador como usuarios normales para ingresar al sistema.

Diagrama 4-9 Interfaz de la pantalla de ingreso

Elaborado por: David Salgado

Esta pantalla cuenta con dos campos de texto; uno para ingresar el usuario que viene a ser el correo electrónico y otro para la contraseña. Contiene también un botón de ingresar el cual llamará a la función de autenticación de Firebase para validar la sesión y si esta validación es satisfactoria pasará a la siguiente pantalla.

Para la validación, como se observa en el diagrama 4-10 debemos importar “FirebaseAuth” en el controlador y llamar a la función signIn de la clase estática Auth a la cual debemos pasarle los parámetros: email de tipo String, password de tipo String y completion de tipo AuthResultCallback que maneja el resultado del inicio de sesión.

Diagrama 4-10 Función para el inicio de sesión con Firebase

```
@IBAction func login(_ sender: Any) {

    if userTextField.text != nil && passwordTextField.text != nil {
        self.view.endEditing(true)
        self.loginBtn.animateButton(shouldLoad: true, title: "", finish: {
            if let email = self.userTextField.text, let password = self.passwordTextField.text {
                Auth.auth().signIn(withEmail: email, password: password, completion: { (user, error) in
                    if error == nil {
                        if let user = user {
                            print(user)
                            DataService.instance.userIsAdmin(uid: user.uid, handler: { (status) in
                                if !status {
                                    DataService.instance.userIsVerified(uid: user.uid, handler: { (status) in
                                        if !status {
                                            UpdateService.instance.updateFirebaseDBUserStatus(uid: user.uid, status:
                                                "verified")
                                        }
                                    })
                                }
                                UpdateService.instance.updateUserFCMToken(uid: user.uid, isAdmin: status)
                                self.loginBtn.animateButton(shouldLoad: false, title: "INGRESAR", finish:{})
                                self.performSegue(withIdentifier: "login", sender: self)
                            })
                        }
                    } else {
                        self.loginBtn.animateButton(shouldLoad: false, title: "INGRESAR",finish: {})
                        if let errorCode = AuthErrorCode(rawValue: error!._code) {
                            switch errorCode {
                            case .wrongPassword:
                                self.showAlert(ERROR_MSG_WRONG_PASSWORD)
                            default:
                                self.showAlert(ERROR_MSG_UNEXPECTED_ERROR)
                            }
                        }
                    }
                })
            }
        })
    }

}
```

Elaborado por: David Salgado

Adicionalmente en esta función se guardan datos relevantes que serán usados en otros módulos como el token de FCM, que es necesario para el envío y recepción de mensajes push y; si el usuario es nuevo se actualiza su estado de pendiente a verificado.

4.6. Rol de usuario administrativo

Para el manejo de este rol se debe crear previamente una colección en la base de datos llamada “admins”, el key usado para cada registro va a ser el UID asignado por Firebase al momento de su creación. Cada registro de esta colección contendrá la estructura descrita en el Diagrama 3-3. Basado en esta estructura se crea funciones que permitan identificar cuando un usuario es administrador y cuando no, para esto se usa la función “userIsAdmin” que se observa en el diagrama 4-11 a la cual se le pasa como parámetro el UID del usuario y esta se encarga de verificar si este key existe en la colección de “admins”, si este registro existe significa que el usuario si es administrador caso contrario es un usuario normal.

Diagrama 4-11 Función para identificar si un usuario es administrador

```
func userIsAdmin(uid: String, handler: @escaping (_ status: Bool) -> Void) {
    REF_ADMINS.child(uid).observeSingleEvent(of: .value, with: { (snapshot) in
        if snapshot.exists() {
            handler(true)
        }else{
            handler(false)
        }
    })
}
```

Elaborado Por: David Salgado

Los usuarios de tipo administrador tienen varios privilegios sobre los usuarios normales, entre estos se encuentran: monitorear la posición de cada usuario y monitorearlos en el mapa, crear y eliminar usuarios en la red, visualización de todos los reportes creados en la red y la recepción de “Check Ins” provistos por los usuarios de la red. El interfaz varía acorde al rol, este se verifica cada vez que se muestre la pantalla de mapas usando la función “adaptUIByRole” como se ve en el diagrama 4-12.

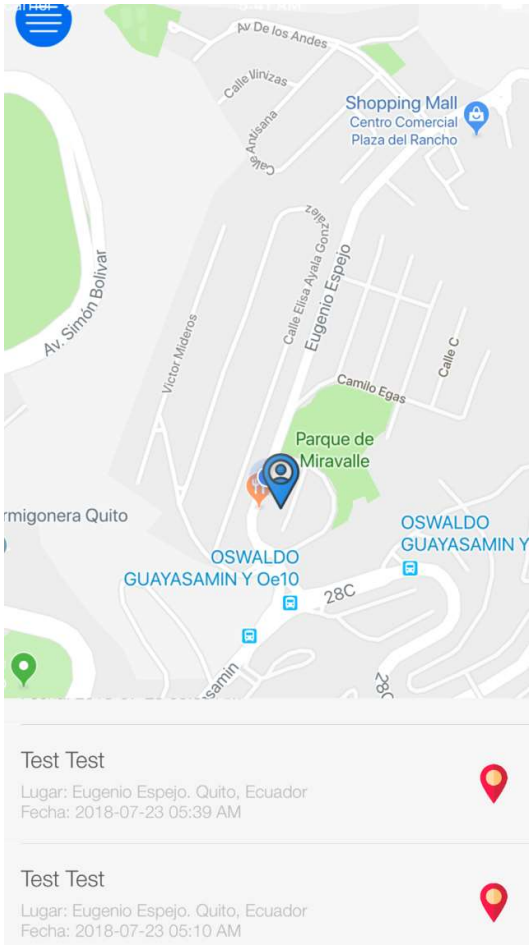
Diagrama 4-12 Función para adaptar el interfaz acorde al rol

```
func adaptUIByRole() {
    DataService.instance.userIsAdmin(uid: self.currentUserId!) { (status) in
        if(status){
            self.uiShouldHide(shouldHide: true)
            DataService.instance.getAdminNetworkRef(uid: self.currentUserId!).observeSingleEvent(of: .value, with:
                { (circleSnapshot) in
                if let circleSnapshot = circleSnapshot.children.allObjects as? [DataSnapshot] {
                    for user in circleSnapshot {

                        let userDict = user.value as? NSDictionary
                        let userId = userDict?["userId"] as? String ?? ""

                        self.createObserverForUser(userKey: userId)

                    }
                }
            })
        }else{
            self.loadUserTrackingStatus()
        }
    }
}
```

Elaborado Por: David Salgado

Los componentes que un usuario administrador no deba observar y sean pertenecientes a un usuario normal tal como el interruptor de rastreo y el botón de “Check In” se ocultarán y tan solo quedará a la vista la tabla de recepción de “Check Ins” diagrama 4-13.

Diagrama 4-13 Interfaz de administradores

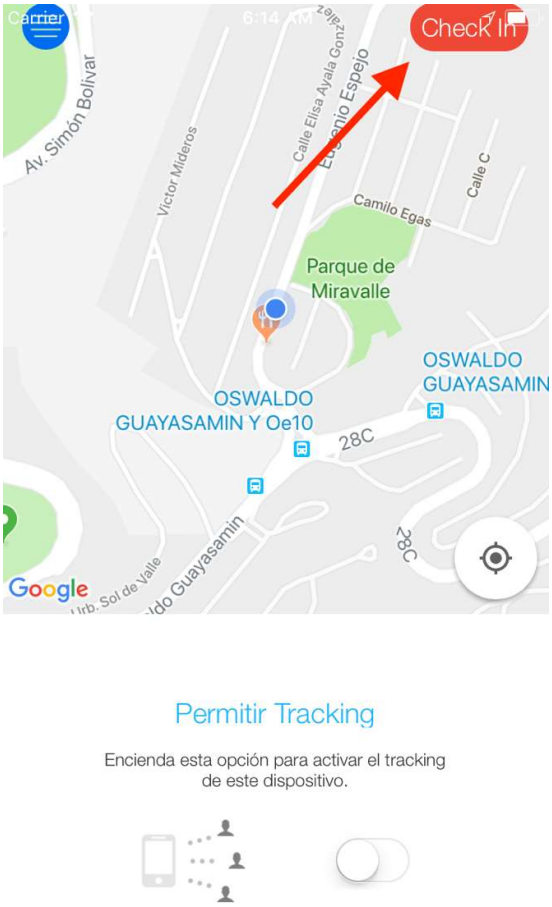


Elaborado Por: David Salgado

4.7. Rol de usuarios clientes

Para los usuarios normales del aplicativo se desplegará un interfaz distinto a los administradores, en este se mostrará el interruptor de rastreo y el botón de “Check In”, que son usados por el módulo de geolocalización y se ocultará la opción de creación de usuarios. El interfaz resultante es el que se puede apreciar en el diagrama 4-14

Diagrama 4-14 Interfaz de usuarios clientes



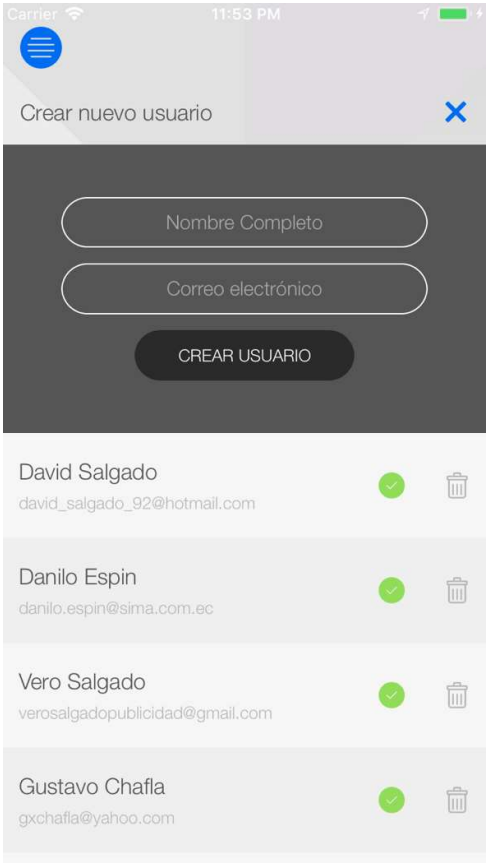
Elaborado Por: David Salgado

Ya que el usuario administrador debe tener la capacidad de visualizar la posición de todos los usuarios pertenecientes en la red se crea observadores para cada miembro de la red de este administrador.

4.8. Gestión de usuario cliente

Los usuarios que tienen rol administrador, si se dirigen al menú lateral en la pestaña de creación de usuarios tienen la posibilidad de agregar o eliminar usuarios a la red que administran. Para esto se manejará el interfaz que se puede apreciar en el diagrama 4-15.

Diagrama 4-15 Interfaz de gestión de clientes



Elaborado Por: David Salgado

Esta pantalla cuenta con una sección que permite agregar usuarios de la red ingresando su nombre completo, un correo electrónico y presionando el botón de “Crear Usuario” al presionar este botón llamamos a la función “createUserButtonWasPressed” que se puede observar en el diagrama 4-16 la cual se encarga de obtener los valores ingresados en los campos de texto de nombre completo y correo electrónico y pasarlos a la función “createUser” de la clase estática Auth proveniente de la librería “FirebaseAuth” para crear un usuario en el servicio de “Authentication” de Firebase y nos devuelve como respuesta un usuario en caso de que la creación sea exitosa o un error en caso de que haya surgido algún problema. Del usuario que obtenemos como respuesta usaremos la propiedad UID como key para crear un

registro en la colección de usuarios empleando la función “createNetworkUser” del diagrama 4-17 la cual tiene como parámetros: el uid de tipo String, username de tipo String, password de tipo String y el adminId de tipo String, esta función, usando la estructura descrita en el diagrama 3-3 para el objeto de la colección users, crea un registro tipo user con la información que se ingresó, le asigna estado pendiente, ya que este usuario aún no ingresa sesión y agrega a la vez un registro de tipo network en el objeto del administrador que está creando este nuevo usuario.

Diagrama 4-16 Función llamada al momento de presionar el botón “Crear Usuario”

```
@IBAction func createUserButtonWasPressed(_ sender: Any) {
    if emailTextField.text != nil && nameTextField.text != nil {
        let password = self.generateRandomPassword(length: 6)
        self.view.endEditing(true)
        self.createUserButton.animateButton(shouldLoad: true, title: "", finish: {
            if let email = self.emailTextField.text, let name = self.nameTextField.text {

                Auth.auth().createUser(withEmail: email, password: password, completion: { (user, error) in
                    if error != nil {
                        if let errorCode = AuthErrorCode(rawValue: error!._code) {
                            switch errorCode {
                            case .invalidEmail:
                                self.showAlert(ERROR_MSG_INVALID_EMAIL)
                            default:
                                self.showAlert(ERROR_MSG_UNEXPECTED_ERROR)
                            }
                        }
                    } else {

                        if let user = user {
                            DataService.instance.createNetworkUser(uid: user.uid, username: name, email: email, password:
                                password, adminId: self.currentUserId!)

                            self.showAlert(SUCCESS_MSG_USER_CREATED)
                        }

                    }
                    self.createUserButton.animateButton(shouldLoad: false, title: "CREAR USUARIO", finish: {})
                })
            }
        })
    }
}
```

Elaborado Por: David Salgado

Diagrama 4-17 Función para agregar un nuevo cliente en el sistema

```
func createNetworkUser(uid: String, username:String, email: String, password: String, adminId: String){

    let usersPost = ["code": uid,
                "username": username,
                "email": email,
                "position": [0,0],
                "adminId": adminId,
                "tempPassword": password,
                "fcmToken": "",
                "notificationTokens": [],
                "trackingEnabled": false,
                "status": "pending"] as [String : Any]

    REF_USERS.child(uid).updateChildValues(usersPost)

    let adminsCirclePost = ["userId": uid,
                            "email": email,
                            "username": username,
                            "status": "pending"]

    self.getAdminNetworkRef(uid: adminId).child(uid).updateChildValues(adminsCirclePost)
}
```

Elaborado Por: David Salgado

La segunda sección de esta pantalla es la lista de usuarios de la red, esta se carga al inicio usando la función “getAdminNetwork” del diagrama 4-18 la cual se encarga de retornar todos los usuarios presentes y su estado de la base de datos, una vez obtenida esta información se llena la tabla y se la presenta al usuario.

Diagrama 4-18 Función para obtener los usuarios presentes en la red del administrador

```
func getAdminNetwork(uid: String, handler: @escaping (_ response: [User]) -> Void) {
    var users:[User] = []
    DataService.instance.getAdminNetworkRef(uid: uid).observe(.value, with: { (snapshot) in
        if let usersSnapshot = snapshot.children.allObjects as? [DataSnapshot] {

            for userSnapshot in usersSnapshot {
                // Get user value
                let userDict = userSnapshot.value as? NSDictionary
                let username = userDict?["username"] as? String ?? ""
                let email = userDict?["email"] as? String ?? ""
                let status = userDict?["status"] as? String ?? ""
                let userId = userDict?["userId"] as? String ?? ""

                let user = User(username: username, email: email, status: status, userId: userId)

                users.append(user)
            }

            handler(users)
        }
    })
}
```

Elaborado Por: David Salgado

En esta lista podemos realizar dos acciones, la primera es visualizar el estado de un usuario los cuales pueden ser pendiente o activo y también eliminar un usuario de la red. Para que un usuario pueda pasar de estado pendiente a activo debe iniciar sesión por primera vez, para esto se usa la función “updateUserStatus” del diagrama 4-19, la cual recibe como parámetros el uid del usuario a modificar y el nuevo estado; y esta se encarga de modificar el estado tanto en el objeto del usuario como en la red de empleados del administrador.

Diagrama 4-19 Función para modificar el estado de un usuario

```
func updateUserStatus(uid: String, status: String){

    let statusPost = ["status": status]
    DataService.instance.REF_USERS.child(uid).updateChildValues(statusPost)

    DataService.instance.REF_USERS.child(uid).observeSingleEvent(of: .value, with: { (snapshot) in
        if snapshot.exists() {
            let userDict = snapshot.value as? NSDictionary
            let adminId = userDict?["adminId"] as? String ?? ""

            DataService.instance.getAdminNetworkRef(uid: adminId).child(uid).updateChildValues(statusPost)
        }
    })
}
```

Elaborado Por: David Salgado

Si se presiona el botón con el ícono de basurero localizado en la parte derecha de cada celda de la lista se elimina el usuario de la red llamando a la función “deleteUser” del diagrama 4-20, la cual recibe como parámetros el uid del usuario a eliminar y el id del administrador para a su vez poder eliminarlo de la red de usuarios.

Diagrama 4-20 Función para eliminar un cliente de la red

```
func deleteUser(uid: String, adminId: String){
    DataService.instance.REF_USERS.child(uid).removeValue()
    DataService.instance.getAdminNetworkRef(uid: adminId).child(uid).removeValue()
}
```

Elaborado Por: David Salgado

4.9. Módulo de georeferencia

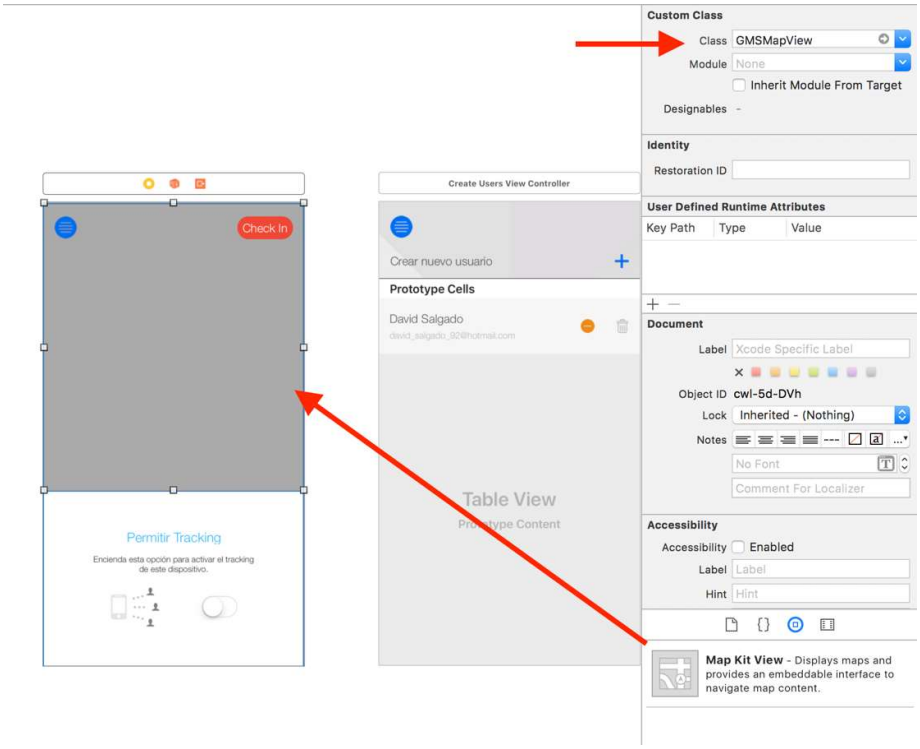
Este módulo se encarga de configurar los mapas de Google en el aplicativo, configurar el GPS, registrar los cambios de posición del usuario y usar estas coordenadas cuando sea necesario, en caso de un usuario administrador, visualizar la posición de cada miembro de la red y para los usuarios clientes permite activar o desactivar el rastreo cuando sea necesario. El interfaz usado en esta pantalla se lo puede visualizar en el

diagrama 4-13 y 4-14 tanto para usuario administrador como para clientes.

4.9.1. Configuración de los mapas de Google

Para configurar los mapas de Google se debe ingresar al Storyboard del proyecto, aquí se agrega un “Map View” en la pantalla en la cual vamos a el mapa y en el “Identity Inspector” le asignamos a este mapa la clase “GMSMapView”, se vincula este componente al controlador y ya es posible usarlo. Este proceso se puede visualizar en el diagrama 4-21.

Diagrama 4-21 Proceso para agregar un mapa de Google Maps en la pantalla del controlador



Elaborado Por: David Salgado

Una vez vinculado el mapa en la cabecera del controlador se importa la librería “GoogleMaps” que nos permite accede a todas las funciones para el manejo del mapa y marcadores.

4.9.2. Configuración del GPS

Para usar el GPS en la plataforma iOS se debe usar la clase “CLLocationManager” esta permite activar el GPS del dispositivo, detectar los cambios de ubicación del dispositivo y arrojar estos datos en un “delegate” para darles el uso requerido.

En el controlador de mapas del aplicativo móvil se inicializa el GPS usando la función initGPS presente en el diagrama 4-22, la cual se llama la primera vez que una instancia de este controlador aparece, previamente se inicializa una variable alojando un objeto de la clase “CLLocationManager”, en este caso se la nombra “locationManager”.

Diagrama 4-22 Configuración e inicialización del GPS en iOS

```
func initGPS(){
    locationManager.requestAlwaysAuthorization()

    if CLLocationManager.locationServicesEnabled() {
        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyBestForNavigation
        locationManager.activityType = .automotiveNavigation
        locationManager.distanceFilter = kCLDistanceFilterNone
        locationManager.pausesLocationUpdatesAutomatically = false
        locationManager.allowsBackgroundLocationUpdates = true
        locationManager.startUpdatingLocation()
    }

    self.mapView.settings.myLocationButton = true
}
```

Elaborado Por: David Salgado

En esta función initGPS se asigna el mismo controlador como “delegate” del GPS, así el objeto “locationManager” puede enviar las coordenadas al controlador, se fija el nivel de precisión, se activa el modo de actualización de posición en segundo plano

para poder seguir receptando coordenadas aun cuando no se encuentra dentro de la aplicación y finalmente se llama a la función “startUpdatingLocation” que comienza a registrar los cambios de posición.

4.9.3. Registro de los cambios de posición

Para poder receptar los cambios de ubicación se requiere implementa un “delegate” del tipo “CLLocationManagerDelegate” en el mismo controlador como se puede ver en el diagrama 4-23.

Diagrama 4-23 Implementación del CLLocationManagerDelegate

```
extension MapViewController: CLLocationManagerDelegate {
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {

        let location: CLLocation = locations.last!
        self.currentLocation = CLLocationCoordinate2D(latitude: location.coordinate.latitude, longitude:
            location.coordinate.longitude)



        if(!isMapInit){
            isMapInit = true

            let camera = GMSCameraPosition.camera(withLatitude: location.coordinate.latitude,
                                                  longitude: location.coordinate.longitude,
                                                  zoom: self.mapZoom)

            if mapView.isHidden {
                mapView.isHidden = false
                mapView.camera = camera
            } else {
                mapView.animate(to: camera)
            }

            mapView.isMyLocationEnabled = true
            mapView.camera = camera
        }
    }
}
```

Elaborado Por: David Salgado

En este “delegate” recibimos la posición del dispositivo y la podemos almacenar en una variable, con esta información podemos ya interactuar con el mapa del aplicativo, moviendo la cámara y teniendo como centro la ubicación actual del dispositivo.

4.9.4. Registro de coordenadas en la base de datos

Los clientes del aplicativo van a tener la posibilidad de reportar su posición en tiempo real por lo que los datos de la posición tienen que ser enviados a la base datos y registradas en la colección de este usuario. En la aplicación se usa un switch que permite activar o desactivar el envío de coordenadas en el aplicativo. Al momento que se activa, en la base datos se modifica la propiedad “trackingEnabled” a true o false dependiendo del estado, esto permite crear observadores para un usuario administrador y saber cuándo desplegar un pin en el mapa y cuando quitarlo. El manejo de este estado debe hacérselo al momento de instanciar el controlador para no crear inconsistencias entre la base de datos y el aplicativo, recuperando el estado ya sea true o false y adaptar el interfaz acorde el estado.

Para actualizar los cambios de posición en la base datos, se usa las coordenadas provenientes del “CLLocationDelegate” y cada vez que se registre un cambio en la posición, validar si el switch de rastreo esta encendido; y si es así, enviar estos datos a la base como se ve en el diagrama 4-24.

Diagrama 4-24 Envío de la ubicación a la base de datos

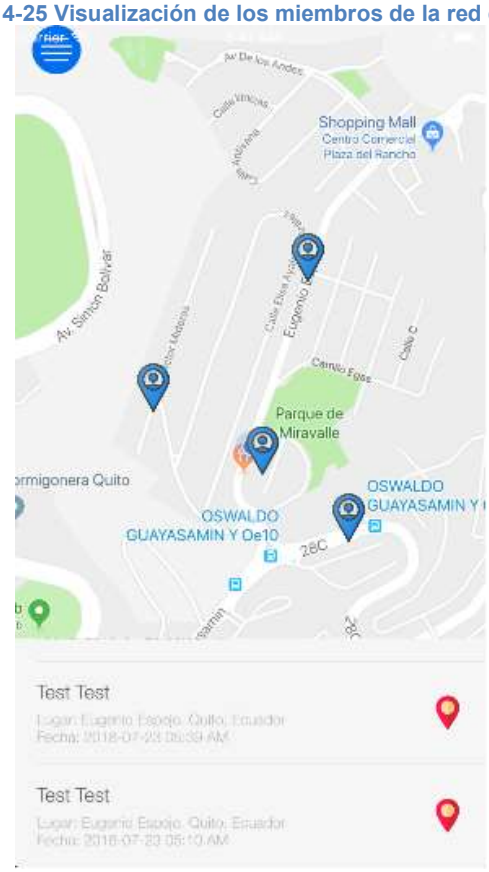
```
if (self.startTrackingSwitch.isOn){
    UpdateService.instance.updateUserLocation(uid: self.currentUserId!, withCoordinate: self.currentLocation )
}
```

Elaborado Por: David Salgado

4.9.5. Visualización de usuarios de la red en el mapa

Los usuarios de tipo administrador tienen la posibilidad de ver en un mapa la posición de cada usuario de su red, representada con un pin como se visualiza en el diagrama 4-25.

Diagrama 4-25 Visualización de los miembros de la red en el mapa



Elaborado Por: David Salgado

Para esto el aplicativo al momento que se inicia el controlador con un usuario de tipo administrador se activa un observador, este observador se encarga de ver si hay alguna modificación en la red de usuarios del administrador. Para esto se realiza un barrido por el objeto “network” que se encuentra en cada objeto de la colección administradores, al hacer esto se obtiene el código de cada usuario miembro de la red y se monta un observador en cada usuario, registrando cualquier cambio en la posición del mismo. Esto se realiza con la función “createObserverForUser” la cual recibe como parámetro el uid del usuario como se puede apreciar en el diagrama 4-26.

Diagrama 4-26 Función para crear observadores en cada usuario de la red

```
func createObserverForUser(userKey: String) {

    DataService.instance.REF_USERS.child(userKey).observe(.value) { (snapshot) in
        if snapshot.exists() {
            let userDict = snapshot.value as? NSDictionary
            let enabled = userDict?["trackingEnabled"] as? Bool ?? false


            if (enabled){
                let positionArray = userDict?["position"] as? NSArray ?? []
                let userCoordinate = CLLocationCoordinate2D(latitude: positionArray[0] as! CLLocationDegrees, longitude:
                    positionArray[1] as! CLLocationDegrees)

                let annotation = UserAnnotation(coordinate: userCoordinate, withKey: userKey)
                annotation.position = userCoordinate
                annotation.icon = UIImage(named: "user-marker")

                let userIsVisible = self.updateIfMapContainsAnnotation(key: userKey, userCoordinate: userCoordinate)

                if !userIsVisible {
                    self.userMarkers.append(annotation)
                    annotation.map = self.mapView
                }

            } else {
                for annotation in self.userMarkers {
                    if annotation.key == userKey {
                        self.userMarkers.remove(at: self.findIndexForAnnotationKey(key: userKey))
                        annotation.map = nil
                    }
                }
            }
        }
    }
}
```

Elaborado Por: David Salgado

Esta función a más de activar el observador coloca, los marcadores en el mapa de acuerdo a la posición del usuario. Cuando los observadores detectan un cambio ya sea de posición o de cualquier otra propiedad del objeto, retorna el objeto actualizado. Al obtener el usuario que se actualizó, esta función verifica el estado del rastreo para conocer si este usuario posee el rastreo habilitado, y así colocar el marcador o caso contrario verificar si existe uno ya creado y removerlo del mapa. En caso de tener el rastreo habilitado, se obtiene la posición del usuario y se crea un marcador en el mapa en esa posición o si ya existe, es actualizado.

4.10. Módulo de mensajería instantánea (Chat)

Si los usuarios se dirigen al menú lateral en la pestaña de "Chat" tienen la capacidad de intercambiar mensajes instantáneos entre ellos, el aplicativo les permite escoger en un menú con cual miembro de la red empezar una conversación, visualizar sus conversaciones activas y enviar mensajes.

4.10.1. Carga de contactos

La carga de contactos se realiza al momento que se inicializa el controlador. Para esto se requiere declarar un arreglo donde se alojarán los objetos que se van a traer de Firebase y que posteriormente será usado para llenar la lista de contactos como se puede ver en el diagrama 4-27.

Diagrama 4-27 Arreglo de contactos y función para llenar la lista de contactos

```
var contacts:[Dictionary<String, AnyObject>] = []

func fetchUsers()  {
    DataService.instance.getUserContacts(uid: currentUserId!) { (response) in
        self.contacts = response
        self.collectionView.reloadData()
        let navVC = self.navigationController as? ChatNavigationViewController
        navVC?.stopHud()

    }
}
```

Elaborado Por: David Salgado

Antes de llenar la lista se requiere usar la función getUserContacts que se puede apreciar en el diagrama 4-28 la cual recibe como parámetro el uid del usuario, esta función se va a encargar de consultar primero si el uid que se le pasó pertenece a un usuario administrador o un usuario normal, en el caso de ser un administrador la función consulta en la base de datos la red del administrador y retorna un arreglo de contactos, caso contrario si el usuario es un cliente, se debe consultar en la base de datos a

que administrador responde este usuario, una vez obtenida esta respuesta se consulta la red de ese administrador para así obtener los contactos.

Diagrama 4-28 Función para carga de contactos

```
func getUserContacts(uid: String, handler: @escaping (_ response: [Dictionary<String, AnyObject>]) -> Void) {
    var users:[Dictionary<String, AnyObject>] = []

    self.userIsAdmin(uid: uid) { (status) in
        if(status){
            //Get admin network
            self.getAdminNetworkRef(uid: uid).observeSingleEvent(of: .value, with: { (snapshot) in
                if snapshot.exists() {
                    if let networkSnapshot = snapshot.children.allObjects as? [DataSnapshot] {
                        for userSnapshot in networkSnapshot {
                            let userDict = userSnapshot.value as? Dictionary<String, AnyObject>
                            users.append(userDict!)
                        }
                        handler(users)
                    }
                }
            })
        }else{
            self.REF_USERS.child(uid).child("adminId").observeSingleEvent(of: .value, with: { (snapshot) in
                //Get admin ID to find his network
                let adminId = snapshot.value as? String
                self.getAdminNetworkRef(uid: adminId!).observeSingleEvent(of: .value, with: { (snapshot) in
                    if snapshot.exists() {
                        if let networkSnapshot = snapshot.children.allObjects as? [DataSnapshot] {
                            for userSnapshot in networkSnapshot {
                                let userDict = userSnapshot.value as? Dictionary<String, AnyObject>
                                if(userSnapshot.childSnapshot(forPath: "userId").value as! String != uid){
                                    users.append(userDict!)
                                }
                            }
                            self.REF_ADMINS.child(adminId!).observeSingleEvent(of: .value, with: { (snapshot) in
                                let adminDict = snapshot.value as? Dictionary<String, AnyObject>
                                let name = adminDict!["name"] as? String
                                let lastname = adminDict!["lastname"] as? String
                                let adminUser =
                                    [
                                        "username":  "Admin " + name! + " " + lastname! ,
                                        "userId": adminId!
                                    ] as [String:AnyObject]
                                users.append(adminUser)
                                handler(users)
                            })
                        }
                    }
                })
            })
        }
    }
}
```

Elaborado Por: David Salgado

4.10.2. Carga de conversaciones

La carga de conversaciones se realiza al momento de cargar el controlador, para esto se usa la función fetchConversation. Esta función se encarga de llamar a la función getActiveChats que se puede apreciar en el diagrama 4-29. El trabajo de esta función es obtener todos las conversaciones cuyos chats se encuentre activos y adicionalmente incluir una propiedad llamada lastMessage que posee el último mensaje de cada conversación para poder mostrarlo al usuario en la lista de conversaciones.

Diagrama 4-29 Función obtener los chats activos

```
func getUserActiveChats(uid: String, handler: @escaping (_ response: [Dictionary<String, AnyObject>]) -> Void){
    var chats:[Dictionary<String, AnyObject>] = []
    self.userIsAdmin(uid: uid) { (status) in
        if(status){
            self.REF_ADMINS.child(uid).child("activeChats").observeSingleEvent(of: .value, with: { (snapshot) in
                if snapshot.exists() {
                    if let chatsSnapshot = snapshot.children.allObjects as? [DataSnapshot] {
                        let group = DispatchGroup()
                        for chatSnapshot in chatsSnapshot {
                            group.enter()
                            var chatDict = chatSnapshot.value as? Dictionary<String, AnyObject>
                            self.getLastMessage(chatId: chatDict!["chatId"] as! String, handler: { (lastMessage) in
                                chatDict?.updateValue(lastMessage as AnyObject, forKey: "lastMessage")
                                group.leave()
                                chats.append(chatDict!)
                            })
                        }
                        group.notify(queue: .main) {
                            print("Finished all requests.")
                            handler(chats)
                        }
                    }
                }
            })
        }else{
            self.REF_USERS.child(uid).child("activeChats").observeSingleEvent(of: .value, with: { (snapshot) in
                if snapshot.exists() {
                    if let chatsSnapshot = snapshot.children.allObjects as? [DataSnapshot] {
                        let group = DispatchGroup()
                        for chatSnapshot in chatsSnapshot {
                            group.enter()
                            var chatDict = chatSnapshot.value as? Dictionary<String, AnyObject>
                            self.getLastMessage(chatId: chatDict!["chatId"] as! String, handler: { (lastMessage) in
                                chatDict?.updateValue(lastMessage as AnyObject, forKey: "lastMessage")
                                group.leave()
                                chats.append(chatDict!)
                            })
                        }
                        group.notify(queue: .main) {
                            print("Finished all requests.")
                            handler(chats)
                        }
                    }
                }
            })
        }
    }
}
```

Elaborado Por: David Salgado

Cuando el usuario hace click en un contacto se reconoce si la conversación con ese usuario es nueva o si previamente ya existía, en caso de no existir se pasa un al controlador de conversaciones el cual es igual al uid de cada usuario separado con un guión como se puede ver en la función del diagrama 4-28. Este key posteriormente será usado para crear un objeto en la colección de “chats” en caso de enviar un mensaje.

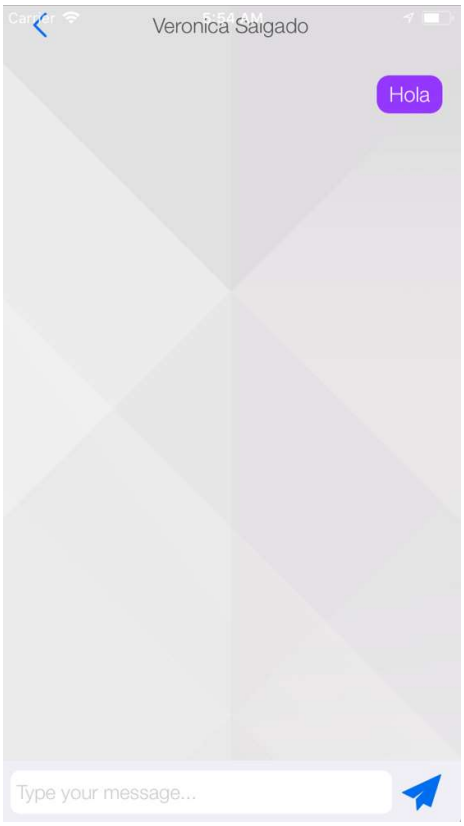
Diagrama 4-30 Función obtener el key único para identificar a la conversación

```
func getUniqueChatId(senderId: String, receiverId: String) -> String{

    var idArr = [senderId, receiverId]
    idArr = idArr.sorted(by: {$0 < $1 })

    return idArr[0] + "_" + idArr[1]
}
```

Elaborado Por: David Salgado

La pantalla de la conversación consta de 3 componentes, una tabla cuyas celdas van a contener cada mensaje que la conversación contenga, un campo de texto para visualizar el mensaje que se está escribiendo y un botón de enviar como se puede ver en el diagrama 4-31.

Diagrama 4-31 Interfaz de la pantalla del chat



Elaborado Por: David Salgado

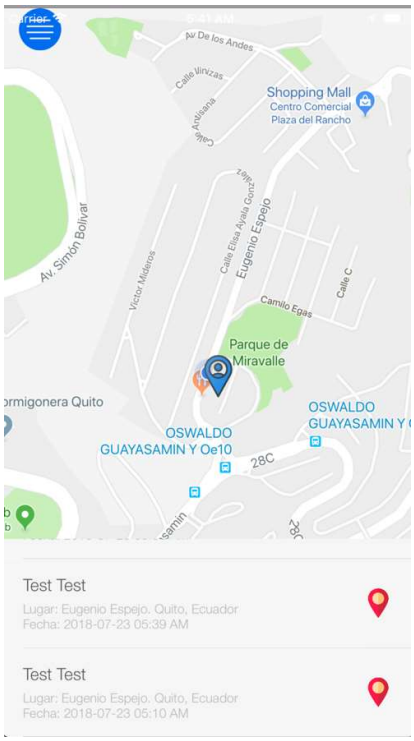
En este controlador se realizan dos procesos fundamentales, el primero es inicializar un observador que esté escuchando cualquier cambio en el chat y si es así vuelve a cargar la lista de mensajes desplegada y segundo un proceso de envío el cual escribe un objeto tipo chat en la colección chats, bajo el key que anteriormente se creó así se asegura que cada mensaje se encuentre en la conversación a la que corresponde.

4.11. Función de registro (Check In)

Este módulo es complementario al módulo de georeferencia y hace uso de las coordenadas actuales del dispositivo para crear un registro de dónde estuvo el usuario, a qué hora fue y notificarlo al administrador. Los registros se los puede observar en una tabla alojada debajo del mapa como se

puede ver en el diagrama 4-32 si se es usuario de tipo administrador, en cada celda se van a cargar los registros que cada miembro de la red realice.

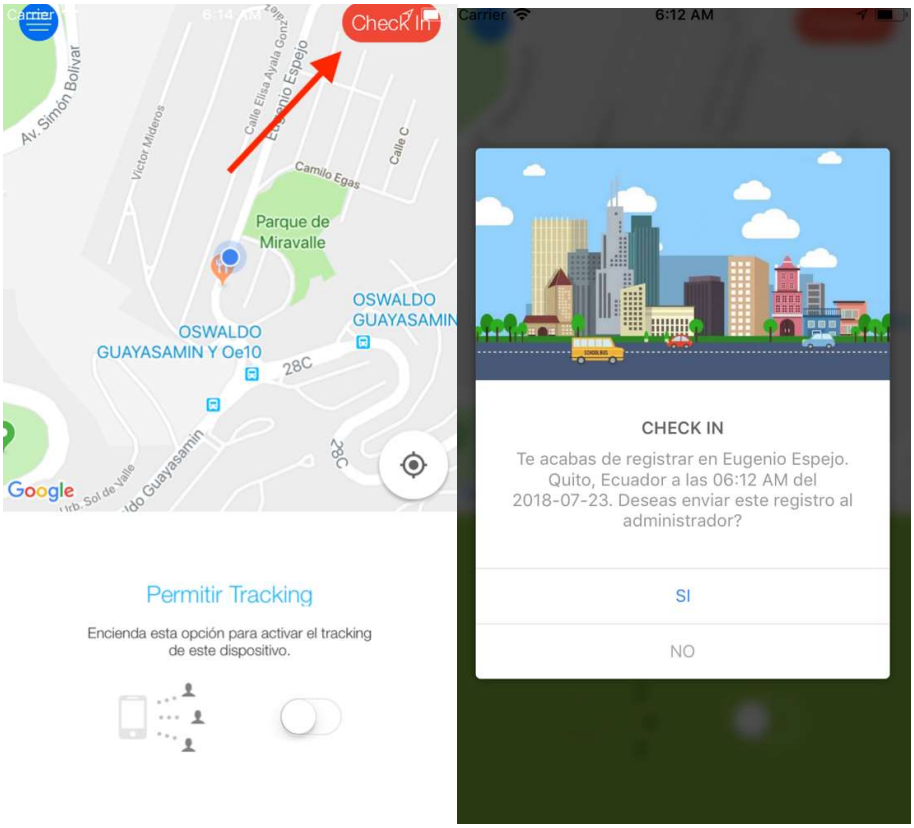
Diagrama 4-32 Interfaz de Check In para administradores



Elaborado Por: David Salgado

Para los usuarios normales el interfaz es distinto ya que cuenta con un botón de Check In en la parte superior derecha y al momento de aplastarlo se presenta una pantalla de confirmación como se puede ver en el diagrama 4-33.

Diagrama 4-33 Interfaz para usuarios normales usado para el Check In



Elaborado Por: David Salgado

Al momento que el usuario miembro de la red presiona el botón de “Check In” por detrás, se captura la posición actual del usuario y se la pasa a la función checkInButtonWasPressed que se puede observar en el diagrama 4-34. Esta función se encarga de transformar estas coordenadas usando la librería de GooglePlaces a una dirección legible, además se registra la hora actual en la que se está realizando el registro y se muestra todo esto en un diálogo.

Diagrama 4-34 Función llamada al presionar el botón de Check In

```
@IBAction func checkInButtonWasPressed(_ sender: Any) {

    geocoder.reverseGeocodeCoordinate(self.currentLocation, completionHandler: { (response, error) in

        if error != nil {
            print(error?.localizedDescription ?? "")
        }

        if let address = response?.firstResult() {
            let lines = address.lines! as [String]

            let fullAddress = lines.first! + ". " + address.locality! + ", " + address.country!

            let now = Date()
            let dateFormatter = DateFormatter()
            dateFormatter.timeZone = TimeZone.current
            dateFormatter.dateFormat = "yyyy-MM-dd"

            let timeFormatter = DateFormatter()
            timeFormatter.timeZone = TimeZone.current
            timeFormatter.dateFormat = "hh:mm a"

            let dateString = dateFormatter.string(from: now)
            let timeString = timeFormatter.string(from: now)

            self.showCheckInPopUpDialog(address: fullAddress, time: timeString, date: dateString)

        }
    })

}
```

Elaborado Por: David Salgado

Si el usuario acepta crear el registro se escribe un objeto de tipo checkIn en la base de datos. Esto dispara un trigger de las funciones en la nube. Llamando a la función que se puede observar en el diagrama 4-35.

Diagrama 4-35 Función disparada al escribir un Check In en la colección de Users

```
exports.sendCheckInNotification = functions.database.ref('/admins/{userUid}/checkIns/{checkInId}').onWrite((change, context) => {
  const userUid = context.params.userUid;
  const checkInId = context.params.checkInId;

  console.log('We have a new check in from UID:', userUid);
  console.log('Check In ID:', checkInId);

  // Get message.
  const getMessagePromise = admin.database().ref(`/users/${userUid}/checkIns/${checkInId}`).once('value');

  // Get the list of device notification tokens.
  const getDeviceTokensPromise = admin.database().ref(`/users/${userUid}/notificationTokens`).once('value');

  return Promise.all([getMessagePromise, getDeviceTokensPromise]).then(results => {
    const message = results[0].val();
    const tokensSnapshot = results[1];

    console.log('User: ', message.username, ' Message: ', message.message);

    // Check if there are any device tokens.
    if (!tokensSnapshot.hasChildren()) {
      return console.log('There are no notification tokens to send to.');
    }
    console.log('There are', tokensSnapshot.numChildren(), 'tokens to send notifications to.');

    // Notification details.
    const payload = {
      notification: {
        title: `${message.username} se registró!`,
        body: `${message.message}`,
        sound: 'default',
        icon: 'http://www.iconninja.com/files/334/253/85/marker-navigation-map-pin-location-icon.png'
      }
    };

    // Listing all tokens.
    const tokens = Object.keys(tokensSnapshot.val());
    console.log(tokens)
    // Send notifications to all tokens.
    return admin.messaging().sendToDevice(tokens, payload).then(response => {
      // For each message check if there was an error.
      const tokensToRemove = [];
      response.results.forEach((result, index) => {
        const error = result.error;
        if (error) {
          console.error('Failure sending notification to', tokens[index], error);
          // Cleanup the tokens who are not registered anymore.
          if (error.code === 'messaging/invalid-registration-token' ||
              error.code === 'messaging/registration-token-not-registered') {
            tokensToRemove.push(tokensSnapshot.ref.child(tokens[index]).remove());
          }
        }
      });
      return Promise.all(tokensToRemove);
    });
  });
});
```

Elaborado Por: David Salgado

Esta función recepta cada elemento que se escribe en la propiedad checkIns de cada usuario, leyendo a su vez la propiedad notificationTokens. Se pasea por cada token enviando una notificación push, notificando así al administrador respectivo cuando este no se encuentre con la aplicación abierta.

Si se es usuario administrador, el aplicativo realiza otro proceso al momento de cargar el controlador, se llama a una función que instancia un observador sobre los checkIns para así

mantener actualizada la tabla de CheckIns como se puede ver en el diagrama 4-36.

Diagrama 4-36 Función para actualizar la tabla de registros

```
func initCheckInsTable(){

    self.checkInsTableView.dataSource = self
    self.checkInsTableView.delegate  = self

    DataService.instance.REF_ADMINS.child(self.currentUserId!).child("checkIns").queryOrdered(byChild: "tsc").observe(.value) { (snapshot) in
        if snapshot.exists() {
            let checkIns = snapshot.value as? Dictionary<String, Any>
            self.checkIns = []
            for (key,_) in checkIns! {
                self.checkIns.append(checkIns![key] as! Dictionary<String, Any>)
            }


            self.checkInsTableView.reloadData()

        }
    }
}
```

Elaborado Por: David Salgado

Esta función recepta cada cambio y actualiza el arreglo que es usado para cargar la tabla, una vez realizado este proceso se recarga la tabla con los nuevos datos.

4.12. Módulo de reportes

Este módulo se encarga de mostrar al usuario todos los recorridos realizados usando el aplicativo. En caso de abrir esta pantalla con un usuario de tipo administrador este puede visualizar los recorridos de todos los miembros de su red, caso contrario un usuario normal visualiza solo sus recorridos. Para obtener estos resultados usamos la función initReportsTable que se puede apreciar en el diagrama 4-37, la cual se encarga de consultar todos los reportes correspondientes.

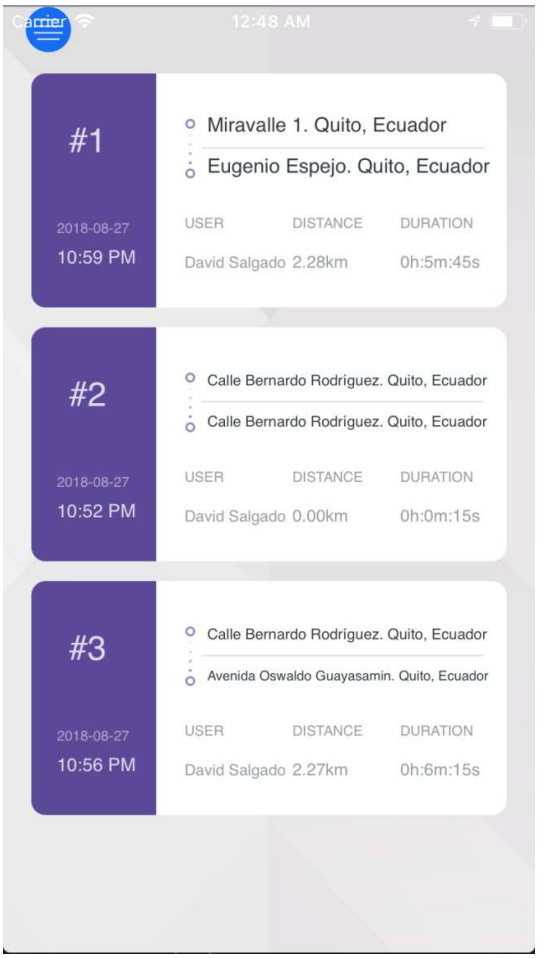
Diagrama 4-37 Listado de trackings

```
func initReportsTable(){

    self.tableView.dataSource = self
    self.tableView.delegate  = self

    DataService.instance.REF_USERS.child(self.currentUserId!).child("reports").observeSingleEvent(of: .value, with: { (snapshot) in

        if snapshot.exists() {
            let reports = snapshot.value as? Dictionary<String, Any>
            self.reports = []
            for (key,_) in reports! {
                self.reports.append(reports![key] as! Dictionary<String, Any>)
            }
            self.tableView.reloadData()
        }
    })

}
```

Elaborado Por: David Salgado

Con esta función se obtiene la fecha, punto de partida, punto de llegada, duración y distancia recorrida de cada viaje otorgando a los usuarios una mejor percepción de cada viaje. Estos datos se los carga en una lista para ser presentados al usuario como se puede ver en el diagrama 4-38.

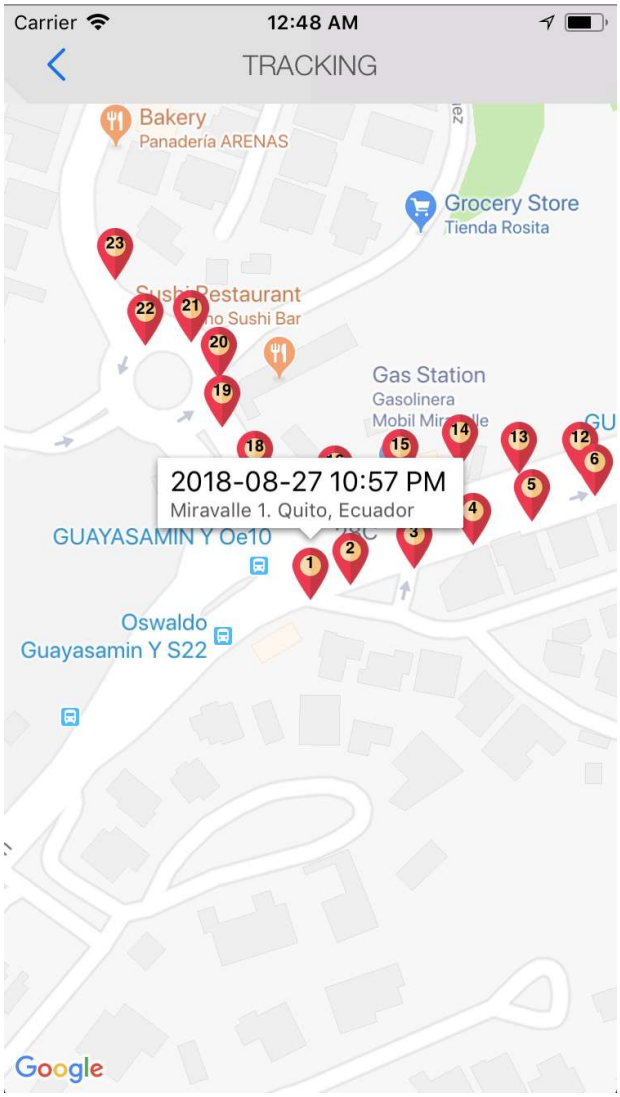
Diagrama 4-38 Listado de trackings



Elaborado Por: David Salgado

Al dar un click en la celda se despliega el trayecto del viaje seleccionado, en la base de datos en cada objeto de la clase reports se tiene una propiedad llamada logs, esta contiene un arreglo que guarda las coordenadas del viaje que el usuario ha realizado. Cada coordenada se despliega en un mapa con un marcador, trazando así la ruta realizada como se puede ver en el diagrama 4-39. Adicionalmente, al hacer un click en cada puntero se puede visualizar la fecha y dirección del punto visitado, estos punteros están numerados para facilitar el seguimiento de la ruta visitada por el usuario.

Diagrama 4-39 Listado de trackings



Elaborado Por: David Salgado

Capítulo V: Pruebas

En el siguiente capítulo se realizará una serie de evaluaciones y análisis realizando una serie de pruebas al software desarrollado para el presente proyecto, mismo software que ha sido sometido a escenarios reales y críticos (pruebas Beta), con el fin de encontrar variables de medición del consumo de datos y del rendimiento del hardware del dispositivo utilizado.

5.1. Pruebas aplicación móvil

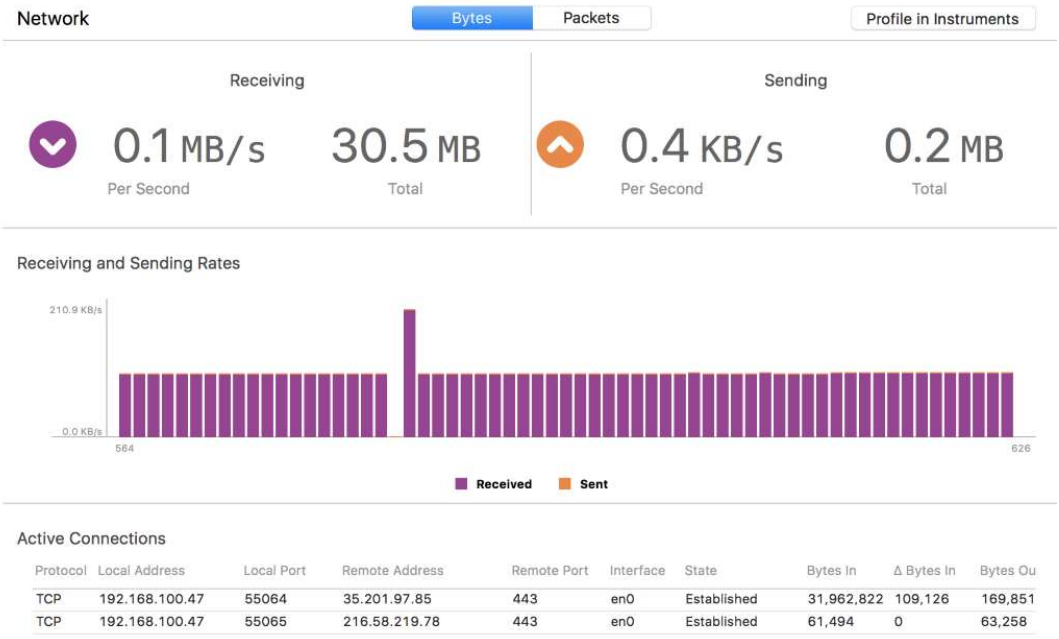
Tipo de Evaluación: La presente evaluación analiza el consumo de datos con respecto a las transferencias de información de posiciones y escritura del reporte desde el dispositivo móvil en la base de datos, los gráficos del consumo son provistos por el IDE desarrollo Xcode.

Periodo de tiempo: 30 minutos.

Condiciones: Es necesario tomar en cuenta que la evaluación ha sido realizada en un dispositivo iPhone 5, con sistema operativo iOS 10.

5.1.1. Coordenadas en tiempo real y reporte cada 5 segundos

Diagrama 5-1 Gráfico del consumo de datos con coordenadas en tiempo real y reporte cada 5 segundos

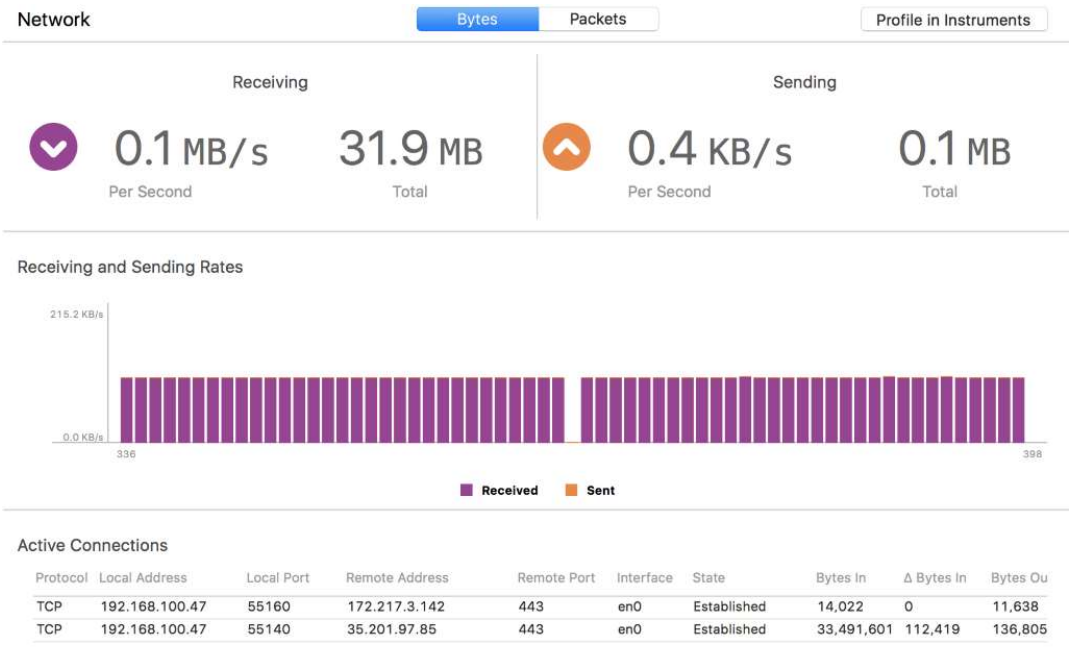


Elaborado Por: David Salgado

Análisis: En el diagrama 5-1 se considera que la frecuencia de escritura del reporte se lo realiza cada 5 segundos, la frecuencia de escritura de coordenadas se lo realiza en tiempo real durante un periodo de 5 minutos, durante este proceso se ha generado un total de 30.5 MB en descarga de datos, siendo equivalente a un consumo de 0.08mbps.

5.1.2. Coordenadas en tiempo real y reporte cada 15 segundos

Diagrama 5-2 Gráfico del consumo de datos con coordenadas en tiempo real y reporte cada 15 segundos

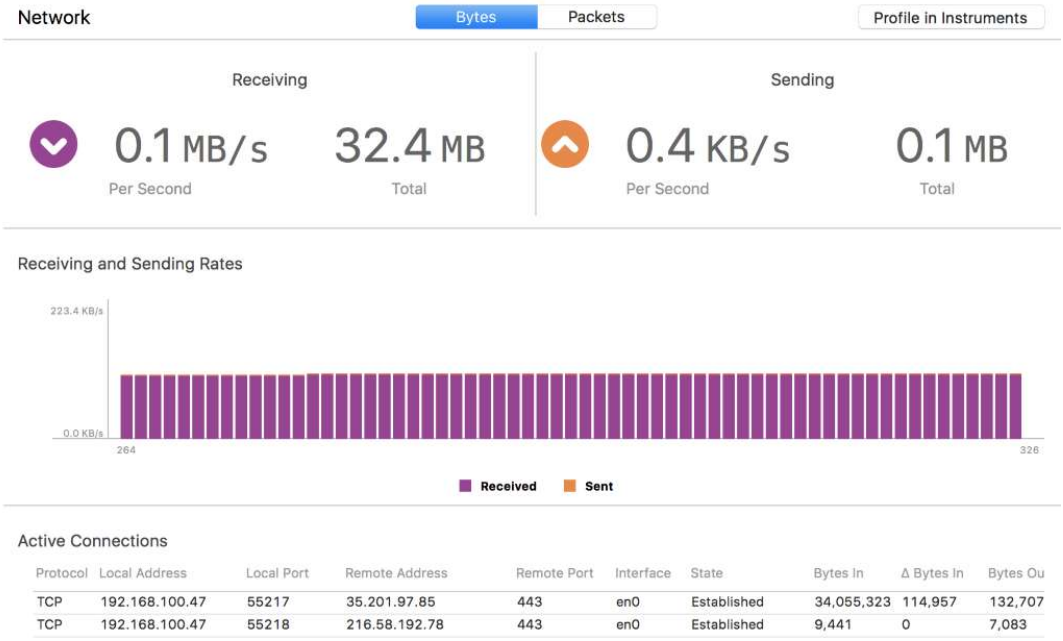


Elaborado Por: David Salgado

Análisis: En el diagrama 5-2 se considera que la frecuencia de escritura del reporte se lo realiza cada 15 segundos, la frecuencia de escritura de coordenadas se lo realiza en tiempo real durante un periodo de 5 minutos, durante este proceso se ha generado un total de 31.9MB de descarga de datos, siendo equivalente a un consumo de 0.106 mbps.

5.1.3. Coordenadas en tiempo real y reporte cada 30 segundos

Diagrama 5-3 Gráfico del consumo de datos con coordenadas en tiempo real y reporte cada 30 segundos

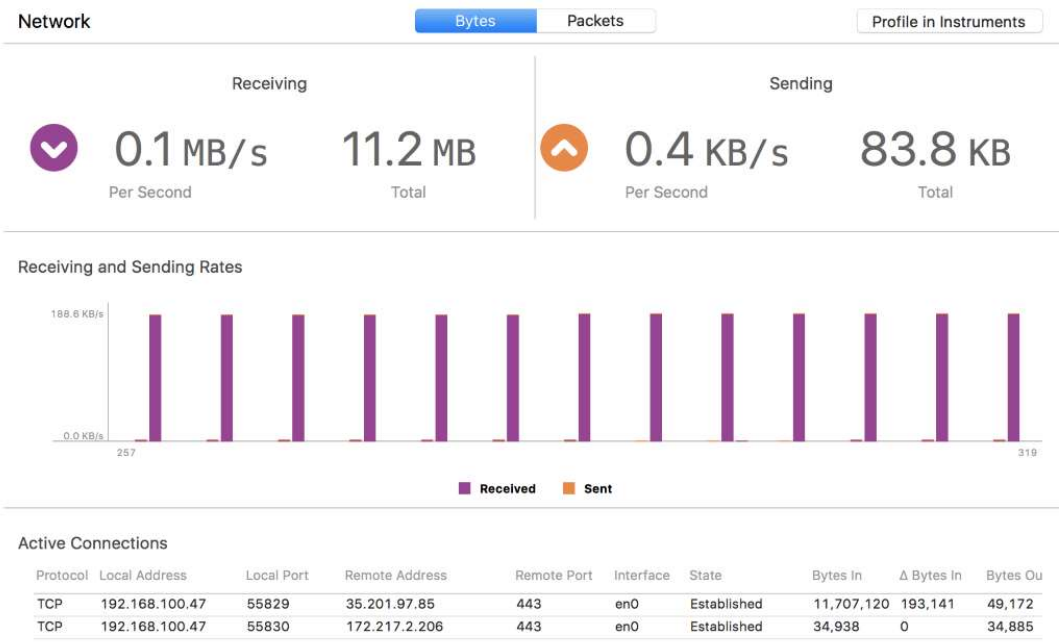


Elaborado Por: David Salgado

Análisis: En el diagrama 5-3 se considera que la frecuencia de escritura del reporte se lo realiza cada 30 segundos, la frecuencia de escritura de coordenadas se lo realiza en tiempo real durante un periodo de 5 minutos, durante este proceso se ha generado un total de 32.4MB de descarga de datos , siendo equivalente a un consumo de 0.108 mbps.

5.1.4. Coordenadas y reporte cada 5 segundos

Diagrama 5-4 Gráfico del consumo de datos con coordenadas y reporte cada 5 segundos

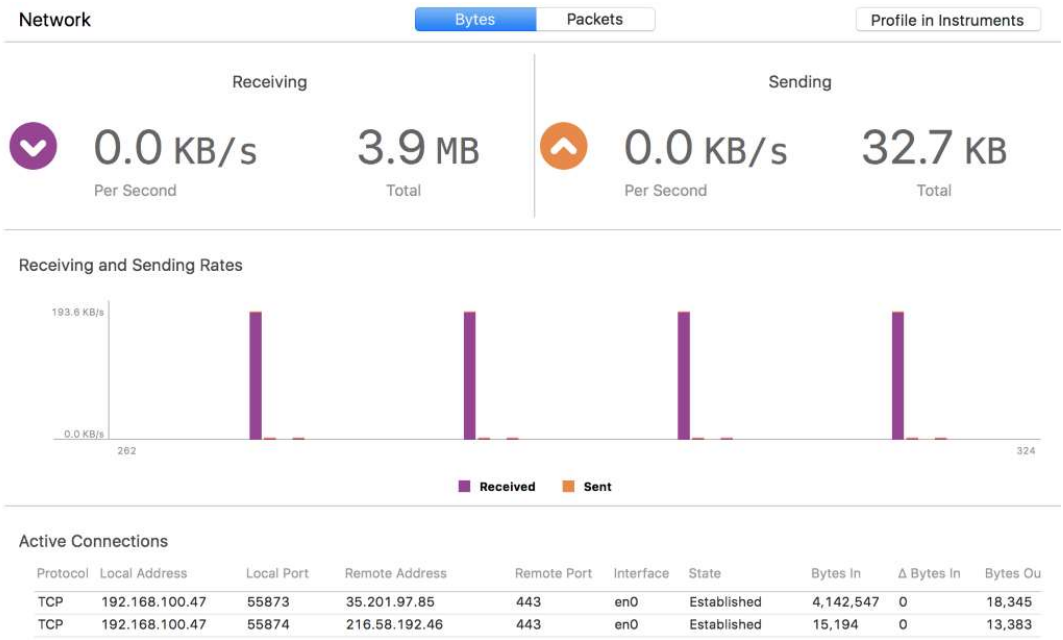


Elaborado Por: David Salgado

Análisis: En el diagrama 5-4 se considera que la frecuencia de escritura del reporte se lo realiza cada 5 segundos, la frecuencia de escritura de coordenadas se lo realiza cada 5 segundos durante un periodo de 5 minutos, durante este proceso se ha generado un total de 11.2MB de descarga de datos, siendo equivalente a un consumo de 0.037 mbps.

5.1.5. Coordenadas y reporte cada 15 segundos

Diagrama 5-5 Gráfico del consumo de datos con coordenadas y reporte cada 15 segundos

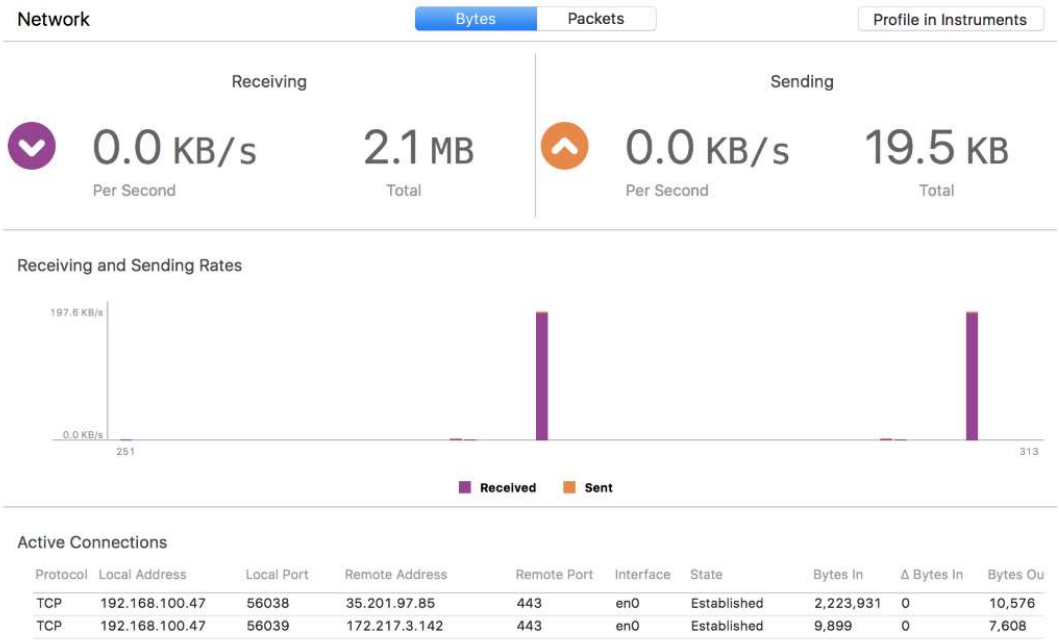


Elaborado Por: David Salgado

Análisis: En el diagrama 5-5 se considera que la frecuencia de escritura de reporte se lo realiza cada 15 segundos, la frecuencia de escritura de coordenadas se lo realiza cada 15 segundos durante un periodo de 5 minutos, durante este proceso se ha generado un total de 3.9MB de descarga de datos, siendo equivalente a un consumo de 0.013 mbps.

5.1.6. Coordenadas y reporte cada 30 segundos

Diagrama 5-6 Gráfico del consumo de datos con coordenadas y reporte cada 30 segundos



Elaborado Por: David Salgado

Análisis: En el diagrama 5-6 se considera que la frecuencia de escritura de reporte se lo realiza cada 30 segundos, la frecuencia de escritura de coordenadas se lo realiza cada 30 segundos durante un periodo de 5 minutos, durante este proceso se ha generado un total de 2.1MB de descarga de datos, siendo equivalente a un consumo de 0.0073 mbps.

5.2. Consumo de batería

Tipo de evaluación: La presente evaluación analiza el consumo de batería con respecto a los sensores (hardware) utilizados por el dispositivo.

Periodo de Tiempo: 30 minutos.

Condiciones: Se debe tomar en cuenta que la evaluación ha sido realizada en un dispositivo IPhone 7 con un ciclo de vida de batería mayor a los 2 años.

Sensores y hardware utilizados

- **Red LTE:** Envía datos de posición GPS a la base de datos, en caso de no tener acceso a internet por WiFi.
- **Pantalla (Brillo 65%):** Se debe tomar en cuenta el consumo de batería, ya que este sensor consumo gran parte de esta.
- **Sensor GPS:** Calcula la latitud y longitud para obtener una posición en el plano, este hardware también realiza un consumo considerable de batería.

5.2.1. Consumo de batería en primer plano LTE + GPS

Tabla 5-1 Tabla de consumo de batería en primer plano

Estado	Tiempo de Actividad (Min)	Descarga de Batería (%)
OK	5	5
OK	10	3
OK	15	2
OK	20	2
OK	25	3
OK	30	3
	Total	18

Elaborado Por: David Salgado

Análisis: Teniendo en cuenta los sensores activados se realiza una evaluación durante 30 minutos, en los cuales se ha

realizado 6 puntos de controles, al final se determina un desgaste de la batería del dispositivo móvil de un 18%.

5.2.2. Consumo de batería en segundo plano LTE + GPS

Tabla 5-2 Tabla de consumo de batería en segundo plano

Estado	Tiempo de Actividad (Min)	Descarga de Batería (%)
OK	5	3
OK	10	2
OK	15	1
OK	20	1
OK	25	2
OK	30	2
	Total	11

Elaborado Por: David Salgado

Análisis: Teniendo en cuenta los sensores activados se realiza una evaluación durante 30 minutos, en los cuales se ha realizado 6 puntos de controles, al final se determina un desgaste de la batería del dispositivo móvil de un 11%. Cabe recalcar que en esta ocasión la aplicación estuvo cerrada, es decir, el proceso están segundo plano.

5.3. Evaluación de la precisión del GPS

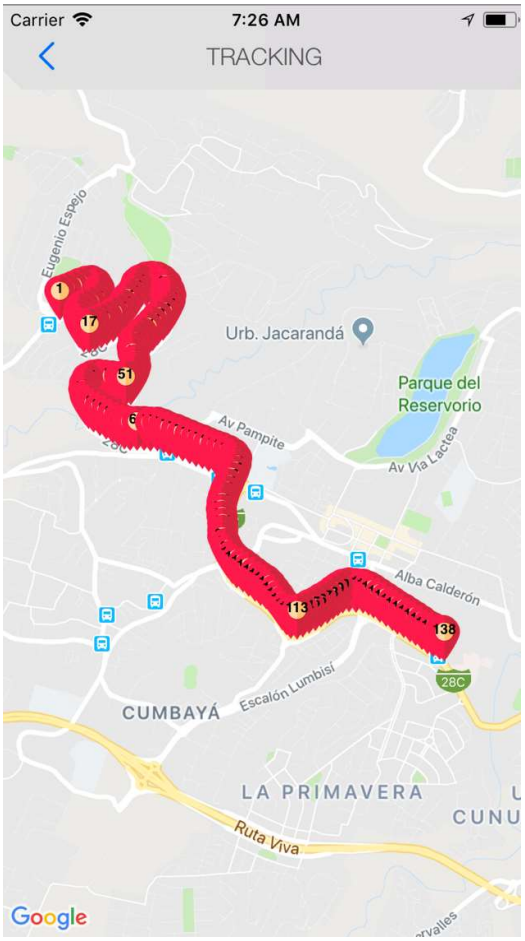
Tipo de evaluación: Esta prueba se centra en analizar la precisión de la posición obtenida por los sensores del dispositivo móvil versus la posición real del usuario.

Periodo de tiempo: 30 minutos.

Condiciones: Esta evaluación se realiza usando el GPS integrado del dispositivo, la prueba se realiza usando un iPhone 5 el cual está conectado a una red LTE para poder enviar las coordenadas a la base.

5.3.1. Prueba trayecto # 1

Diagrama 5-7 Visualización del trayecto realizado en la prueba # 1



Elaborado Por: David Salgado

Análisis: El dispositivo GPS resultó muy preciso en esta prueba, el trayecto duró aproximadamente 35 minutos y se recorrió una distancia de 16 km, permite fácilmente detectar el rumbo que siguió la persona en una autopista y se detectó un total de 138 puntos.

5.3.2. Prueba trayecto # 2

Diagrama 5-8 Visualización del trayecto realizado en la prueba # 2

Elaborado Por: David Salgado

Análisis: El rastreo al igual que en la prueba número uno resultó muy precisa, cabe recalcar que durante este trayecto se atravesó por un túnel, la señal se perdió en la boca del túnel pero se reanudó al salir del mismo sin perder la precisión. Este trayecto

duro aproximadamente 33 minutos en donde se recorrió una distancia de 10 km y se detectó un total de 133 puntos.

6. Conclusiones

- Al desarrollar un aplicativo móvil que brinde la posibilidad de rastrear a un usuario en tiempo real hay que considerar aspectos muy importantes como el consumo de batería, consumo de datos y configuración del hardware mediante código ya que estos puntos afectan directamente al desempeño de un aplicativo haciendo que está pueda ser una herramienta viable para el uso en escenarios reales o no. (Referencia 5.1 y 5.2)
- Con la ayuda de Firebase se agilita mucho el proceso de desarrollo ya que ahorra mucho tiempo al no tener que realizar configuraciones, montar un servidor, autenticación de usuarios y al proveer de un mecanismo sencillo de acceso a los datos, es una herramienta eficaz y sólida para proyectos de pequeña y mediana escala, además de ahorrar costos ya que es una arquitectura que tiene todos sus servicios en la nube. (Referencia 2.1.1.9, 4)
- El GPS integrado de los dispositivos móviles iOS es una herramienta muy útil la cual se puede configurar y gracias a su alta precisión que nos provee permite crear aplicativos cuyo objetivo es el rastreo de usuarios o emplearlo en aplicaciones relacionadas que hagan uso de geolocalización. (Referencia 5.3).

7. Recomendaciones

- Al iniciar el desarrollo de un aplicativo móvil una vez definido los requerimientos del sistema es importante realizar una investigación previa de librerías o herramientas que puedan aportar al proyecto en el diseño del interfaz o en las funcionalidades base del aplicativo, esto ahorra tiempo de desarrollo ya que muchas veces nos encontramos con contenido libre que se puede reutilizar y acoplarlo a nuestras necesidades. Firebase es un ejemplo claro en este proyecto una librería que se ve usada concurrentemente durante todo el desarrollo (Referencia 4), de igual manera para la creación de los interfaces RAMPaperSwitch para el interruptor que activa el rastreo, InteractiveSideMenu para el menú lateral y PopupDialog para el interfaz del diálogo de Check-In (Referencia 4.2, 4.5 y 4.9)
- Si se desea realizar un aplicativo móvil que use el GPS integrado, investigar todas las configuraciones que éste posee y adecuarlas según lo requerido, esto ayuda a crear un aplicativo más óptimo y preciso. (Referencia 4.7.2)
- Cuando se trabaje con bases no relacionales es importante crear una estructura que facilite las búsquedas de datos relevantes, en ciertos casos es mejor repetir información en ciertos nodos que se guardan para evitar realizar búsquedas complejas. Un ejemplo claro en este proyecto se aprecia en el módulo Registro de Usuarios y de Check-In (Referencia 4.6 y 4.9)

8. Glosario

GPS: Sistema que permite conocer la posición de un objeto o de una persona gracias a la recepción de señales emitidas por una red de satélites.

Hardware: Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático.

Software: Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

API: Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

SDK: Sus siglas vienen de Software Development Kit. Es un kit de desarrollo de software que reúne un grupo de herramientas que permiten la programación de sistemas, aplicaciones tanto de escritorio, web, y móviles.

Librería: Conjunto de implementaciones funcionales, codificadas en un lenguaje de programación para ser utilizada por otros programas, independientes y de forma simultánea.

NoSQL: Clase de sistemas de gestión de datos (mecanismos para el almacenamiento y recuperación de datos) que difieren, en aspectos importantes, del modelo clásico de relaciones entre entidades (o tablas) existente en los sistemas de gestión bases de datos relacionales, siendo el más destacado el que no usan SQL como lenguaje principal de consulta.

JSON: Formato de texto ligero para el intercambio de datos.

CRUD: Acrónimo de "Crear, Leer, Actualizar y Borrar" (Create, Read, Update, Delete).

Backend: Alude a la parte encargada de procesar y almacenar la información.

Frontend: Hace referencia al interfaz de usuario que presenta y/o recibe la información.

UID: Identificador único.

9. Bibliografía

Aho, A., Ullman, J., Sethi, R., & Lam, M. S. (1998). *Compiladores: Principios Técnicas y Herramientas.* Pearson Educación.

Angulo, R. (2013). *DEBATES IESA.* Obtenido de Tecnología Móvil: http://cursa.ihmc.us/rid=1NTQ9NMKD-R1SKBP-24M4/Aplicaciones%20moviles%20hibridas-%20lo%20mejor%20de%20dos%20mundos.pdf

Baz Alonso, A., Ferreira Artime, I., Álvarez Rodríguez, M., & García Baniello, R. (s.f.). *Dispositivos Móviles.* Obtenido de uniovi: http://isa.uniovi.es/docencia/SIGC/pdf/telefonia_movil.pdf

Delia, L., Galdamez, N., Thomas, P., & Pesado, P. (s.f.). *sedici*. Obtenido de Un Analisis Experimental de Tipo de Aplicaciones para Dispositivos Móviles: http://sedici.unlp.edu.ar/bitstream/handle/10915/32397/Documento_completo.pdf?sequence=1

Google. (s.f.). *Cloud Messaging*. Obtenido de Firebase Google: https://firebase.google.com/docs/cloud-messaging/concept-options

Google. (s.f.). *Firebase*. Obtenido de Fireabse Authentication: https://firebase.google.com/docs/auth/

Google. (s.f.). *Firebase*. Obtenido de Firebase Google: https://firebase.google.com/docs/database/?gclid=Cj0KCQjwvqbaBRCOARIsAD9s1XA8Ctow4DSbXnx9E-EZ0DuTZSlvuWJrhoCpYoIxpBjmbP6CEZ2HOukaAvZLEALw_wcB

Google. (s.f.). *Firebase*. Obtenido de Firebase.Google: https://firebase.google.com/docs/database/?gclid=Cj0KCQjwvqbaBRCOARIsAD9s1XA8Ctow4DSbXnx9E-EZ0DuTZSlvuWJrhoCpYoIxpBjmbP6CEZ2HOukaAvZLEALw_wcB

Google. (s.f.). *Google Maps Platform*. Obtenido de Cloud Google: https://cloud.google.com/maps-platform/maps/

Google. (s.f.). *Google Maps Platform*. Obtenido de Cloud Google: https://cloud.google.com/maps-platform/places/

http://developer.apple.com. (s.f.). *WWDC18*. Obtenido de developer Apple: http://developer.apple.com

NodeJS. (s.f.). *nodejs*. Obtenido de about NodeJS: https://nodejs.org/en/about/

Reinman, A. (s.f.). *NodeMailer*. Obtenido de Outfunnel: https://nodemailer.com/about/

Wells, J. D. (2001). *sourceforge.* Obtenido de Aplicación de eXtreme programming en ONess: http://oness.sourceforge.net/proyecto/html/index.html

SMAK, M. (17 de Marzo de 2012). *SlideShare.* Obtenido de https://www.slideshare.net/MrSMAk/extreme-programming-12047889

Beck, K., & Gamma, E. (2000). *Extreme programming explained: embrace change*. Addison-Wesley Professional. Obtenido de http://index-of.co.uk/Etc/Extreme%20Programming%20Explained.%20Embrace%20Chan.pdf

Ponce, D., & Prado, J. (2017). *Estudio de las redes GPS-WI-FI y su aplicación en la monitorización de dispositivos móviles* (Tesis de pregrado). Pontificia Universidad Católica del Ecuador. Quito, Ecuador.

Sánchez Rodríguez, D. C., & Cano Castañeda, N. D. (2014). *Tecnología e Informática.* Obtenido de fie.sanjo: http://files.sanjo2014.webnode.es/200000001-c34cac445e/INTRODUCCION%20A%20LA%20PROGRAMACION.pdf

Sanchez, A. (19 de Septiembre de 2014). *prezi.* Obtenido de Entorno de Desarrollo Integrado IDE: https://prezi.com/7wmx8_d6ertl/entorno-desarrollo-integrado-ide/

Stevenson, D. (7 de Febrero de 2017). *Quora*. Obtenido de Is Google Firebase realtime database really a good solution for a

messenger app? (requires core data as well?): https://www.quora.com/Is-Google-Firebase-realtime-database-really-a-good-solution-for-a-messenger-app-requires-core-data-as-well

Wolf, G., Ruiz, E., Bergero, F., & Meza, E. (29 de Agosto de 2015). *sistop Fundamentos OS.* Obtenido de sistop: https://sistop.org/pdf/sistemas_operativos.pdf

Araujo, A. (Abril del 2016) ORACLE. Obtenido de ¿Qué es una Base de Datos NoSQL?: https://blogs.oracle.com/spain/qu-es-una-base-de-datos-nosql

Ventura, V. (6 de Marzo del 2013) Acadacual. Obtenido de 6

Frontend – Backend. Aclarando conceptos: https://www.acadacual.es/concepto-frontend-backend.

Letelier, P., & Penadés, M. C. (2012). Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).