UNIVERSIDAD DE LAS AMÉRICAS

FACULTAD DE INGENIERÍAS Y CIENCIAS AGROPECUARIAS

DESARROLLO DE UNA APLICACIÓN MÓVIL PARA EL IPHONE, UTILIZANDO METODOLOGÍA EXTREMA (XP), PARA EL SISTEMA PÚBLICO DE TRANSPORTE BICIQ.

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

Profesor Guía
Ing. Francisco Tixi

Autor
Juan José de la Torre León

Año
2014

DECLARACIÓN DEL PROFESOR GUÍA

"Declaro haber dirigido este trabajo a través de reuniones periódicas con el estudiante, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación"

.....

Juan Francisco Tixi Topón

Ingeniero en Sistemas

C.I. 1713623666

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

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

Juan José de la Torre León

C.I. 1717943748

AGRADECIMIENTOS

"Agradezco a mi familia y a todas las personas que estuvieron pendientes".

DEDICATORIA

"A mis padres, a mi hermana y a mis amigos".

RESUMEN

El presente proyecto de grado tiene como objetivo mostrar el proceso de análisis, diseño y desarrollo de una aplicación móvil para el dispositivo iPhone 4 o superior, usando una metodología ágil. Esta aplicación esta enfocada para mejorar la experiencia en el uso del servicio de bicicletas públicas biciQ.

Se eligió desarrollar la aplicación usando el lenguaje de programación Objective-C, el cual permite crear aplicaciones nativas para dispositivos iOS. Se usó Xcode para: editar el código fuente, crear las interfaces, probar el código en un simulador de iPhone, crear pruebas unitarias y realizar la instalación en un dispositivo iPhone. También se evidencia el uso de servicios web basados en una API de tipo REST y la interacción con la red social Facebook a través de su API pública.

La aplicación móvil cuenta con 5 módulos principales. El primero permite la autenticación del usuario y la creación de una nueva cuenta, la información de ambos módulos es procesada a través del servicio REST. El segundo módulo muestra datos informativos sobre el servicio biciQ de una forma dinámica. El tercer módulo muestra las últimas entradas de la página oficial de biciQ en Facebook. El cuarto módulo muestra cada estación de la biciQ en un mapa y permite crear una ruta entre las dos estaciones o entre la ubicación actual del usuario y una estación. El último módulo permite cambiar los datos de la cuenta del usuario o eliminar la misma.

A lo largo del proceso de análisis, diseño y desarrollo de la aplicación, se evidencia el uso de la metodología ágil de desarrollo XP, dicho conjunto de procesos metodológicos permite adoptar un desarrollo basado en pruebas, crear un código simple, fácil de entender y escalable. Permitiendo crear un aplicación de alta calidad en corto tiempo.

ABSTRACT

This thesis aims to show the process of analysis, design and development of a mobile application for the iPhone 4 or higher, using an agile methodology. This application is focused to enhance the experience when using the public bicycle service biciQ.

The application is developed using Objective-C as a programming language, which allows to create native applications for iOS devices. Xcode was used to: edit the source code, create interfaces, test an iPhone simulator, create automatized unit tests and for the deployment on an iPhone device. It also demonstrates the use of web services based on a REST API and the interaction with the social network Facebook through its public API.

The mobile app has 5 main modules. The first allows user authentication and the creation of a new account; the information in both modules is processed through the REST service. The second module displays information about biciQ service in a dynamic way. The third module displays the latest posts from biciQ official page on Facebook. The fourth module shows each biciQ station on a map and allows the user to create a route between two stations or between the user's current location and one station. The last module allows the user to modify his account details or delete the same.

Throughout the process of analysis, design and development of the application, gets demonstrated the use of the agile development methodology XP, this set of methodological processes permits to adopt a test-driven development, writing simple code, easy to understand and scalable. Allowing creating a high quality application in a short time.

ÍNDICE

Introducción

Antecedentes

biciQ es una nueva propuesta de transporte público en bicicleta por parte del Municipio de la ciudad de Quito. Este nuevo sistema fue inaugurado el 31 de Julio del año 2012, impulsando el uso de bicicletas para el transporte urbano, reduciendo costos de movilización al hacerlo de una manera ágil y ecológica.

El sistema cuenta con 425 bicicletas distribuidas en 25 estaciones, ubicadas estratégicamente en lugares cercanos a los puntos de mayor afluencia, atracción o interés comercial, bancario, turístico o estudiantil. Dichas estaciones están distribuidas a lo largo del perímetro de la Estación Norte del Trolebús-La 'Y', hasta la Plaza de Santo Domingo, en el Centro Histórico.

El servicio funciona todos los días del año de 07:00 a 19:00. La inscripción y la emisión de la tarjeta no tiene ningún costo. Toda persona que desee usar este servicio necesita seguir el siguiente procedimiento:

1. Llenar el formulario de inscripción a través de la pagina web: http://www.biciq.gob.ec o solicitando el formulario de inscripción en las distintas estaciones o acercándose a las oficinas.

2. Entregar los documentos que solicita biciQ en una de sus oficinas, 72 horas después recibirá una llamada o un correo electrónico de confirmación.

3. Acercarse a las oficinas para que se le otorgue al usuario su credencial del servicio.

4. Ya con la credencial de usuario, se puede acercar a cualquier estación de biciQ para usar una de las bicicletas disponibles. (la credencial se la presenta en la estación de inicio y en la estación de llegada)

En el diario "El Comercio", la Redacción Quito (2012) señala que, al cabo de dos meses de inauguración biciQ ya contaba con 1078 usuarios carnetizados, que en total generaron 523 viajes cortos en bicicleta diarios. La cantidad de subscriptores aumenta cada día, gracias a las propagandas y anuncios mostrados en los diferentes medios de comunicación masiva.

Este atractivo servicio del Municipio de Quito cuenta con un sitio web plenamente informativo (http://www.biciq.gob.ec), dentro de el se encuentra información detallada sobre el proceso de inscripción, políticas y reglamentos, rutas, estaciones, noticias y novedades sobre este servicio.

biciQ no cuenta con algún software para los usuarios que disponen de un teléfono móvil. Una aplicación para este tipo de usuarios conllevará un gran valor agregado para este servicio, ya que dispondrán ágilmente de información importante y relevante para el uso diario de este servicio.

Marco Referencial

Hoy en día la tendencia de las personas que disponen de teléfonos inteligentes, es buscar y usar aplicaciones móviles que faciliten el uso y mejoren la experiencia de los servicios que normalmente utilizamos a diario, tales como: el transporte, alojamiento, información, alimentación, médico, bancario, comunicación y muchos más.

A nivel mundial el uso de teléfonos inteligentes está creciendo rápidamente, esto se ve reflejado con la amplia gama de *Smartphones* que existen en el mercado y sus varios modelos. Los siguientes datos estadísticos, obtenidos del sitio web http://www.mashable.com, Murphy, S. (2012)., demuestran el crecimiento móvil a nivel mundial:

- De 7 billones de habitantes 5.9 billones tienen un Smartphone.
- La venta de estos dispositivos aumento un 63.1% desde el 2010.

- Más de 300.000 aplicaciones móviles han sido desarrolladas en los últimos 3 años.
- En un 64%, los *smartphones* son usados en el transporte público.
- 95% de los *smartphones* son utilizados para hacer búsquedas locales (restaurantes, mapas, sitios de interés, etc.)

Esta tendencia se ve reflejada en el Ecuador, el modelo de negocio de las grandes marcas y empresas está enfocándose a los smartphones, como por ejemplo, la aplicación móvil de la farmacia Pharmacys, la de Multicines, Diario el Comercio, aplicaciones de entidades bancarias, entre otras.

Dentro de las lista de las aplicaciones móviles para iPhone y iPod Touch más descargadas en el Ecuador, obtenidas desde el sitio web http://www.distimo.com, (Distimo, s.f), se encuentra la aplicación móvil Waze. La funcionalidad principal de esta aplicación es ayudar al usuario a ubicarse en un mapa y crear rutas para optimizar sus trayectos en la ciudad que se encuentre. Dado esto, se puede deducir que gran cantidad de personas que poseen un *smartphone* en el país están usando esta aplicación y están acostumbrados a usar este tipo de software para facilitar y solventar los problemas que surgen cuando queremos llegar de un lugar a otro por medios de transporte público o privado.

Considerando la tendencia de crecimiento de smartphones en el país y la familiaridad que poseen los dueños de estos dispositivos al usar aplicaciones móviles que optimicen su movilización diaria en una ciudad; es un momento oportuno para desarrollar una aplicación móvil para el sistema público de transporte biciQ, el cual solvente las preguntas más comunes que tienen los usuarios de este servicio, tales como:

- ¿Cómo puedo utilizar el servicio y cómo me registro?
- ¿Cuáles son los horarios y reglamentos?
- ¿Dónde están las paradas y sus rutas?
- ¿Cómo puedo llegar en la bicicleta a 'x' lugar?

Esto supone una gran ayuda y valor agregado para los usuarios de la biciQ, ya que dispondrán de información elemental de dicho servicio en todo momento y de una manera muy rápida y amigable para visualizarla.

Alcance

El alcance de este proyecto es el análisis, diseño, desarrollo e implementación de una aplicación móvil para dispositivos iPhone con sistema operativo iOS 7, enfocado al sistema público de transporte biciQ, utilizando metodología ágil de desarrollo (XP). Esta aplicación tendrá los siguientes módulos:

- **Login/Registro App:** Si el usuario no se ha registrado a la app, tendrá la opción que completar un pequeño formulario de 4 a 5 campos como máximo, con información personal.

- **Información:** La información más relevante sobre uso, políticas, horarios, restricciones, procesos, historia de la empresa y datos de contacto.

- **Noticias:** Dentro de este módulo se podrá visualizar las publicaciones de la página de Facebook de la biciQ (http://www.biciq.gob.ec/web/index.php/servicios/noticias/).

- **Mapa dinámico e interactivo:** Esta módulo, mostrará las rutas y paradas de la biciQ, con funcionalidad de ruteo (Ir de un punto A a un punto B) y puntos de interés referentes a la biciQ (oficinas, lugares de mantenimiento, etc.)

- **Redes Sociales (Facebook):** El objetivo de este módulo es integrar la red social Facebook, para dar a conocer la aplicación y compartir mensajes de recorrido por día.

Justificación

En la actualidad, las aplicaciones móviles están incorporándose más a nuestro país con el fin de optimizar ciertos aspectos de nuestra vida. Dado esto, desarrollar una aplicación móvil para el servicio de la biciQ, llegaría a ser un proyecto muy innovador para los usuarios, facilitando el uso de este servicio.

Disponer un mapa interactivo con toda la información de las paradas y rutas, conllevará una inmensa ayuda para el ciclista. Este aspecto es fundamental, ya que será un valor agregado que se le ofrecerá al usuario final de la biciQ, y en si facilitaría y mejoraría el uso de este gran servicio público.

biciQ es un servicio muy joven, lleva operando alrededor de un año, hoy en día ya cuenta con más de 10000 usuarios y cada día crece más. Gran parte de la comunidad quiteña conoce de este servicio, ya que hay mucho apoyo por parte de las entidades encargadas y estas están generando gran cantidad de publicidad en medios radiales e internet. Dado esto, es un momento adecuado para desarrollar una aplicación móvil que mejore y facilite el uso de este servicio.

Objetivo General

Desarrollar una aplicación móvil para el sistema público de transporte biciQ usando código nativo (Objective-C) para dispositivos iOS.

Objetivos Específicos

- Usar metodología XP (Extreme Programming) para la implementación de la aplicación móvil.
- Probar la aplicación en dispositivos iPhone versión 4 con sistema operativo iOS 7.x.
- Diseñar la interfaz gráfica utilizando principios de las guías de diseño y usabilidad de Apple.
- Codificar la aplicación móvil usando estándares para el lenguaje Objective-C.
- Crear un ambiente de pruebas.

Metodología a utilizar

Siendo el objetivo principal de la tesis el desarrollo de una aplicación móvil nativa, el proceso investigativo comprenderá el conocimiento del lenguaje de programación para dispositivos iOS (Objective-c), los recursos y funcionalidades que ofrece el hardware, sistema operativo en el cual será instalado la aplicación (iPhone y iOS) y la abstracción y recopilación de información del sistema de transporte publico biciQ para implementar las funcionalidades que ofrecerá la aplicación móvil a todos los usuarios.

Planteado lo anterior, la metodología a utilizarse será la exploratoria descriptiva, la cual ayudará a encontrar, analizar y abstraer todos los detalles y funcionalidades de la aplicación móvil, basados en fenómenos y situaciones reales.

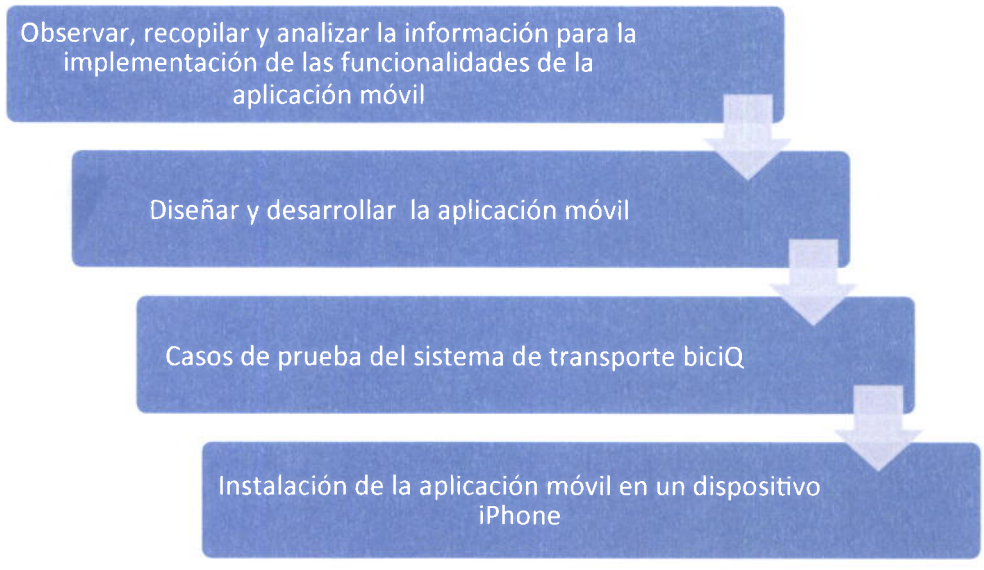


Figura 1. Diagrama de proceso sobre la metodología a utilizar en el trabajo de titulación.

1. Marco Teórico

1.1. Sistema de Transporte Público "biciQ"

biciQ es un proyecto del Distrito Metropolitano de la ciudad de Quito inaugurado en Julio del año 2012. Este es un sistema basado en el concepto internacional de bicicletas públicas o de bicicletas compartidas, cuyo principal objetivo es el de ofrecer un medio de transporte alternativo a una cierta ciudad o región a un bajo costo.

Apegándose al concepto de bicicletas públicas, biciQ se encuentra implementado en la ciudad Quito, cuyas 25 estaciones están repartidas desde el Centro hasta el Centro-Norte de la ciudad de Quito.

Figura 2. Mapa de la Ciclo-Vías y Estaciones de la biciQ, Sector Centro. Tomado de (BiciQ, 2012)

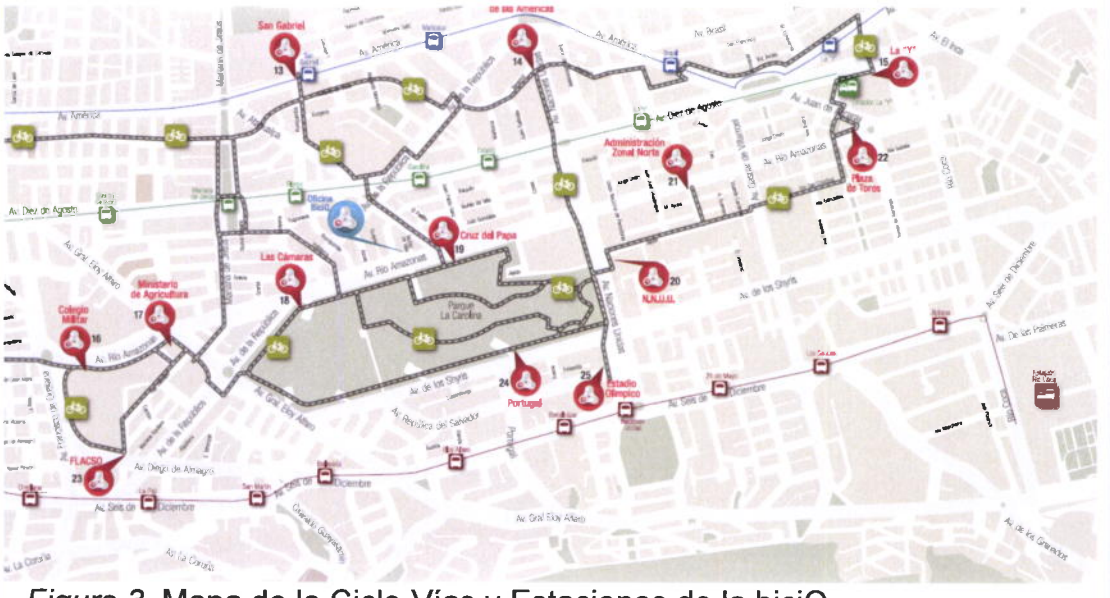


Figura 3. Mapa de la Ciclo-Vías y Estaciones de la biciQ, Sector Centro-Norte. Tomado de BiciQ, 2012

El ciudadano que desee utilizar este novedoso servicio deberá llenar un formulario de inscripción y finalmente obtendrá una credencial de usuario que deberá presentar en las estaciones que retire o deje la bicicleta pública.

La credencial es la herramienta que utiliza la biciQ como seguridad y control sobre las bicicletas, ya que al momento que un abonado desee usar una bicicleta deberá presentar la credencial al encargado de la estación en la cual retirará la bicicleta, esta persona desliza el carnet en un equipo PDA y entrega la credencial al abonado; el mismo procedimiento es realizado al momento de entregar la bicicleta en otra estación. El PDA está enlazado con un sistema remoto de gestión y control sobre las bicicletas prestadas de la biciQ, dicho sistema permite conocer en que momento del día, que persona, en que estación, retiró una bicicleta y si es que esta fue devuelta o no en el tiempo que está permitido usar una bicicleta.

Según diario "El Comercio", la Redacción Quito (2012) señala que, existen 425 bicicletas públicas, distribuidas alrededor de las paradas o estaciones. Las

bicicletas que se encuentran disponibles son de color azul o rojo, estas poseen asientos regulables, canasta frontal, timbre y 5 marchas. La estructura o marco de la bicicleta es diseñada para que el usuario pueda manejarla de una manera fácil y cómoda, a su vez tiene guardafangos que protegen la vestimenta del usuario cuando el clima no es favorable.

Figura 4. Bicicleta Roja biciQ.

Tomado de Chalmeida, 2012

En otros países ya se ha implementado sistemas de bicicletas compartidas, cuyo efecto a sido muy positivo para el medio ambiente y para los usuarios. En la siguiente tabla se resume los sistemas de bicicletas compartidas más importantes a nivel mundial.

Tabla 1. Sistemas de bicicletas compartidas a nivel mundial

Nombre	País	Ciudad	Abonados	Año	Sitio Web	Aplicación Móvil
Bicing	España	Barcelona	110.488	Marzo 2007	http://www.bicing.cat	http://itunes.apple.com/es/app/bicing/id303950687
Vélib'	Francia	Paris	250.000	Julio 2007	http://www.velib.paris.fr/	http://itunes.apple.com/fr/app/velib-2012/id577807727
Bicicletas Públicas Hangzhou	China	Hangzhou	8.700.000[a]	Mayo 2008	http://www.hzzxc.com.cn/	https://play.google.com/store/apps/details?id=com.sinaapp.hzpubike

Bike Rio	Brasil	Rio de Janeiro	600 bicicletas	Octubre 2011	http://www.mobilicidade.com.br/	https://itunes.apple.com/us/app/bike-rio/id474679114

[a] Cantidad de usuarios anuales.

1.2. Metodología de Desarrollo

Extreme Programming o comúnmente llamada metodología de desarrollo XP, es un conjunto de procedimientos o metodologías cuyo objetivo principal es desarrollar software de calidad a pesar de los cambios que puedan surgir durante las diferentes etapas de un proyecto de software, de esta manera obviando el falso paradigma que indica que los cambios en un proyecto disminuyen las ganancias y a veces comprometen la calidad del producto.

Esta metodología pretende solucionar esta falsa concepción frente a cambios ejerciendo una planificación, codificación y pruebas en el código de una manera reiterada en las diferentes etapas del proyecto, enfatizando la simplicidad, la retro-alimentación, el coraje y la comunicación en todo momento. De esa manera es posible mantener el costo de hacer cambios en el software y que estos no se eleven demasiado.

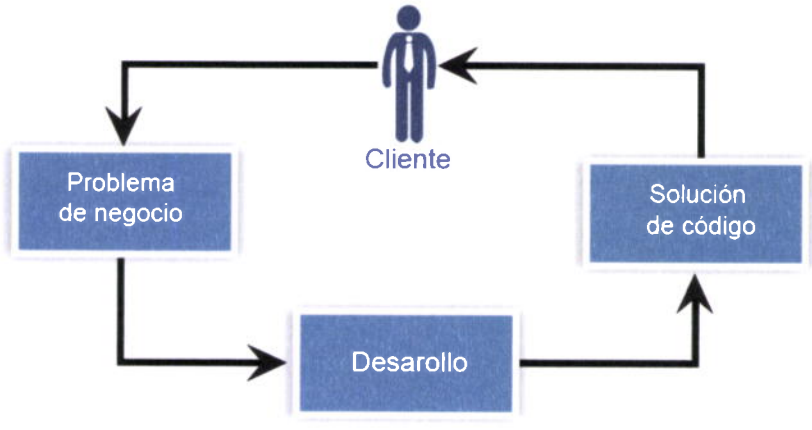


Figura 5. Ciclo XP.

Tomado de Chromatic, 2009, p. 5

XP, como muchas otras, gestiona un proyecto de software basado en 4 variables: **tiempo, alcance, recursos y calidad**.

Todas estas variables son factores limitantes e influyen entre si al momento de desarrollar un proyecto de software. La metodología XP sugiere establecer el nivel de estas variables desde el inicio del proyecto, siendo estas conocidas y aprobadas por el equipo de desarrollo como por cliente. XP recomienda que la variable tiempo y recursos deban permanecer constantes a lo largo de todo el proyecto, acordar el nivel de calidad del proyecto entre ambas partes y evaluar el nivel de alcance regularmente o diariamente, de acuerdo a las necesidades, nuevos requerimientos o limitaciones que surjan en el transcurso del proyecto.

Monitoreando estas variables constantemente, el software puede ser planeado, desarrollado y entregado más deliberada y predeciblemente. Esto se resume en una de las hipótesis que indica esta metodología:

> "Exponer la ventajas y desventajas de los cambios en el software, conduce a un menor número de sorpresas y un desarrollo más fluido" (Chromatic, 2009, p. 6).

La metodología XP y todas sus prácticas se basa en 4 valores principales: **comunicación, retro-alimentación, simplicidad y coraje**.

La comunicación es esencial en cualquier tipo de proyecto y más aún en uno de software donde el equipo y el cliente deben estar enterados de todos los detalles y del estado actual del proyecto. De esta manera el proyecto se flexibiliza frente a cambios, en vez de complicarse y detenerse.

XP se caracteriza por obtener una rápida y frecuente retro-alimentación, en cada iteración, por parte del grupo de desarrolladores y del cliente, por lo tanto los cambios podrán ser resueltos de una manera más eficaz y sin que cause un severo impacto económico.

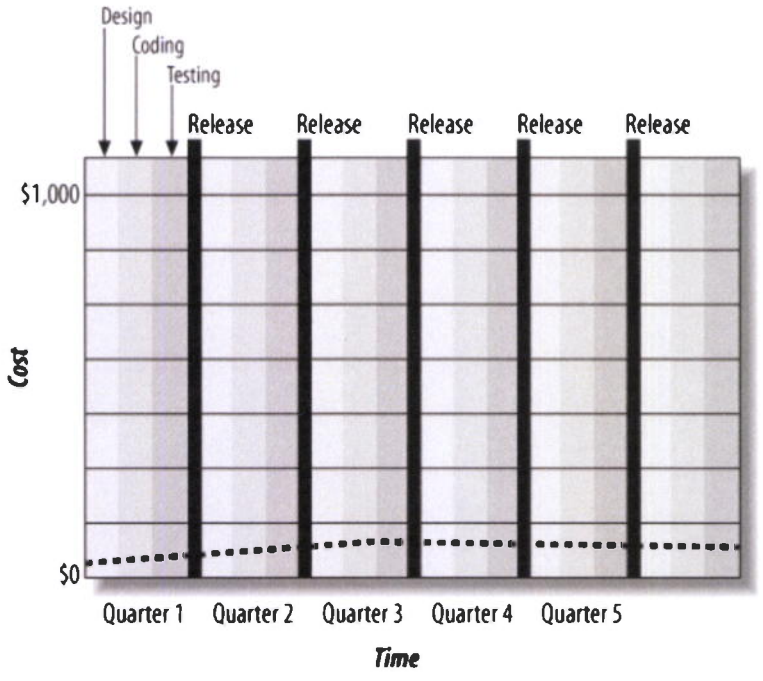


Figura 6. Curva de costo-cambio en XP.

Tomado de Chromatic, 2009, p. 9

> "Retro-alimentación significa hacer preguntas y aprender de las respuestas. La única manera de saber lo que el cliente realmente quiere es preguntárselo a él. La única manera de saber si el código hace lo que realmente debe hacer es probarlo. Entre más pronto se obtiene retro-alimentación, más tiempo tiene para reaccionar ante esto" (Chromatic, 2009, p. 7).

La simplicidad es el valor de desarrollar un sistema que cumpla solo las necesidades que el cliente espera, resolver los problemas que suscitan en el momento del desarrollo y no complicarse en funcionalidades futuras, ya que puede que nunca existan y estas elevan el costo de desarrollo.

Ser valiente o tener coraje en un proyecto de desarrollo de software implica tomar decisiones difíciles cuando sea necesario y hacerse responsable por los aspectos negativos que conlleva la gestión de un proyecto.

Todas estas 4 virtudes: comunicación, retro-alimentación, simplicidad y coraje, deben ser aplicadas en conjunto en un proyecto de desarrollo de software,

llegando a una sinergia entre estas virtudes, se eliminarán muchos contratiempos en el desarrollo de un proyecto, mejorando así la calidad del producto que entregará en un tiempo prudente y satisfaciendo al cliente.

Por último la metodología XP asume suficiencia en varios aspectos para mejorar el proceso de desarrollo de un proyecto.

- **Tiempo Suficiente:** Comprende trabajar a un ritmo normal y constante, implica que frente a cambios solicitados por el cliente, el desarrollador pueda implementarlos fácilmente. Produciendo software valioso para el tiempo y recursos invertidos.
- **Recursos Suficientes:** XP permite tener la cantidad de recursos suficientes para un proyecto. Ajustando el alcance y cantidad de funcionalidad a entregar en cada iteración.
- **Costo de Cambios Constante:** XP asegura que el costo por cambios se mantendrá en el tiempo. Invirtiendo en recursos y tiempo donde estos produzcan mejores resultados. Se llega a esta constancia, mediante frecuentes retro-alimentaciones y lanzamientos de versiones, así como el hecho de producir código flexible y sustentable, enfatizando siempre en la simplicidad y verificación.
- **Efectividad de los desarrolladores:** Un desarrollo efectivo requiere de buenos programadores, con amplios conocimientos y productivos. XP toma ventaja de las prácticas de desarrollo que establece para mejorar la calidad del software y la productividad del equipo de programadores.
- **Libertad para Experimentar:** Todos los involucrados en un proyecto de desarrollo de software deben sentirse libres de experimentar maneras alternativas de como mejorar el proceso del desarrollo del producto.

1.2.1 Las 12 Prácticas de la Metodología XP

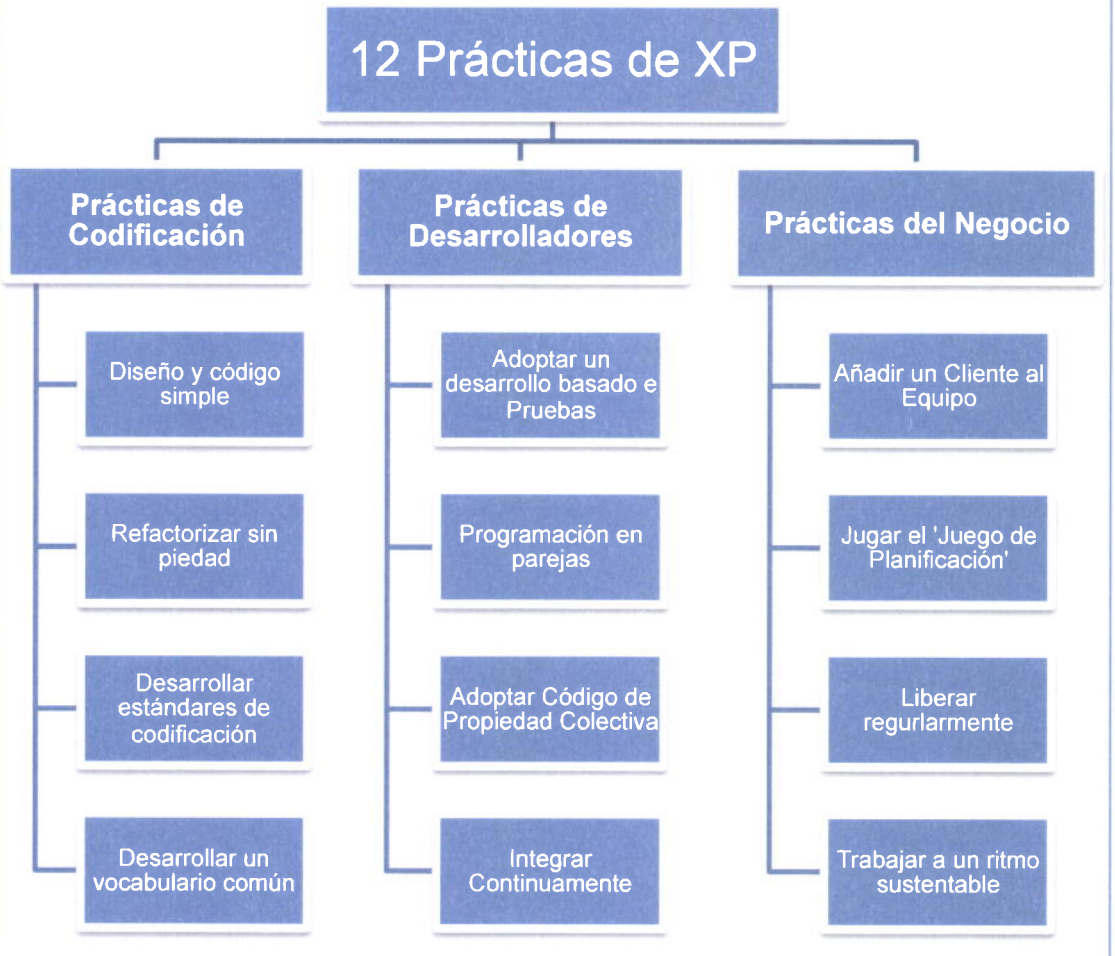


Figura 7. Las 12 Prácticas de la Metodología XP.

1.2.1.1 Diseño y Código Simple.

Objetivo: Producir código que será fácil de cambiar.

Diseñar y codificar de una manera simple significa resolver los requerimientos actuales del cliente y resistirse a considerar en el diseño o código requerimientos futuros. XP practica 3 frases que rigen a la simplicidad en el código y diseño:

Realizar lo más simple que, posiblemente funcione.

No se va a necesitar eso.

Una y solo una vez.

> "La flexibilidad es la meta. La simplicidad es el medio para ese fin. Los diseños simples son más fáciles de entender y de explicar. Código simple es más fácil de probar, mantener y cambiar" (Chromatic, 2009).

1.2.1.2 Refactorizar sin Piedad.

Objetivo: Encontrar el diseño óptimo del código.

Refactorizar no es más que mejorar el diseño del código actual sin cambiar su comportamiento, haciéndolo más simple, conciso y más flexible. La refactorización debe ser empleada regularmente, ya sea después de una tarea, de una prueba, etc., dejando un código fácil de entender y de modificar.

1.2.1.3 Desarrollar Estándares de Codificación.

Objetivo: Comunicar ideas de manera clara a través del código.

El código es la principal forma de comunicación en el proyecto, son convenciones o lineamientos que debe seguir el equipo de desarrollo desde el inicio del proyecto e ir evaluándolos según sea necesario con la finalidad de mejorarlos y clarificarlos. Seguir estándares asume un gran ahorro de tiempo para todo el equipo de desarrollo.

1.2.1.4 Desarrollar un Vocabulario Común.

Objetivo: Comunicar ideas sobre el código de manera clara.

Desarrollar un vocabulario común para describir el proyecto a medida que evoluciona. Dicho vocabulario debería describir cada componente importante del proyecto de una forma expresable y accesible ya que será compartida con los clientes.

1.2.1.5 Adoptar un Desarrollo Basado en Pruebas.

Objetivo: Probar que el código funciona como debería.

Adoptar esta mentalidad implica crear pruebas que fallen y código que pase estas pruebas de una manera cíclica. El automatizar las pruebas y refactorizar el código mejoraran los resultados de estas y provocan una rápida retro-alimentación.

1.2.1.6 Programación en Parejas.

Objetivo: Difundir conocimientos, experiencias e ideas.

El hecho de programar en parejas es muy beneficioso para el proyecto y para los programadores que practican esto. Por los siguientes motivos:

- La persona con el teclado (el conductor) se centra en los detalles de la tares, mientras que el otro programador (el navegador) mantiene el enfoque global del proyecto en la mente.
- Tanto el conductor como el navegador intercambian ideas, de esta manera se comparte y se generan nuevos conocimientos.
- Si existe complicación en resolver una tarea, el cambio de roles entre los programadores es la manera más adecuada de resolverlo rápida y eficazmente.
- El código generado en parejas encaja de mejor manera en el proyecto como un todo, siguiendo todos los lineamientos propuestos por el equipo.

1.2.1.7 Adoptar Código de Propiedad Colectiva.

Objetivo: Difundir la responsabilidad del código con todo el equipo.

El código del proyecto pertenece a todo el equipo, todos los desarrolladores son libres de hacer cambios en el código para completar sus tareas. De esta manera el equipo se siente libre y tranquilo de poder continuar con tareas o código que otro desarrollador lo empezó.

1.2.1.8 Integrar Continuamente.

Objetivo: Reducir el impacto al agregar nuevas características.

Integrar continuamente, fusionar *(merge)* las nuevas tareas y probar en el principal repositorio de código fuente tan pronto como sean completadas. Ejecutar pruebas para verificar que el nuevo código se ajusta bien al sistema. Corregir los errores - comprobar el código en el repositorio sólo cuando el conjunto de pruebas pasa totalmente. Comprobar la última versión del código con frecuencia.

1.2.1.9 Añadir un Cliente al Equipo.

Objetivo: Resolver los problemas de negocio con precisión y de manera directa.

XP denomina `*the whole team*` al hecho de añadir un cliente al equipo. El cliente proporciona un punto de vista comercial como un usuario real del software. La comunicación regular, fiable y rápida entre los técnicos y los clientes mejora la confianza, reduce las dudas, las equivocaciones, y produce los resultados deseados con mayor rapidez.

1.2.1.10 Jugar el 'Juego de Planificación'.

Objetivo: Planificar el trabajo más importante.

XP utiliza el 'Juego de Planificación' para describir el proceso de decidir qué funciones implementar y en qué orden. El objetivo del juego es la planificación para maximizar el valor de las funciones producidas.

1.2.1.11 Liberar Regularmente.

Objetivo: Devolver la inversión del cliente a menudo.

Liberar el código regularmente provoca una rápida retro-alimentación por parte del cliente, sobre las funcionalidades principales del software. Es menos severo liberar código frecuentemente, de esta manera las pruebas se agilitan y el proceso se simplifica.

1.2.1.12 Trabajar a un Ritmo Sustentable.

Objetivo: Ir a casa cansado, pero no exhausto.

Cada persona tiene un nivel de productividad natural, tratar de exceder ese nivel es muy contra-productivo para el desarrollador y para el o los proyectos involucrados.

1.2.2 Eventos de la Metodología XP

1.2.2.1 Planificación de la Iteraciones

El propósito principal de planificar iteraciones es el producir software de alta calidad en tiempos prudentes, priorizando funcionalidades en cada sub etapa de un proyecto, o en cada iteración, según los lineamientos dados por el cliente.

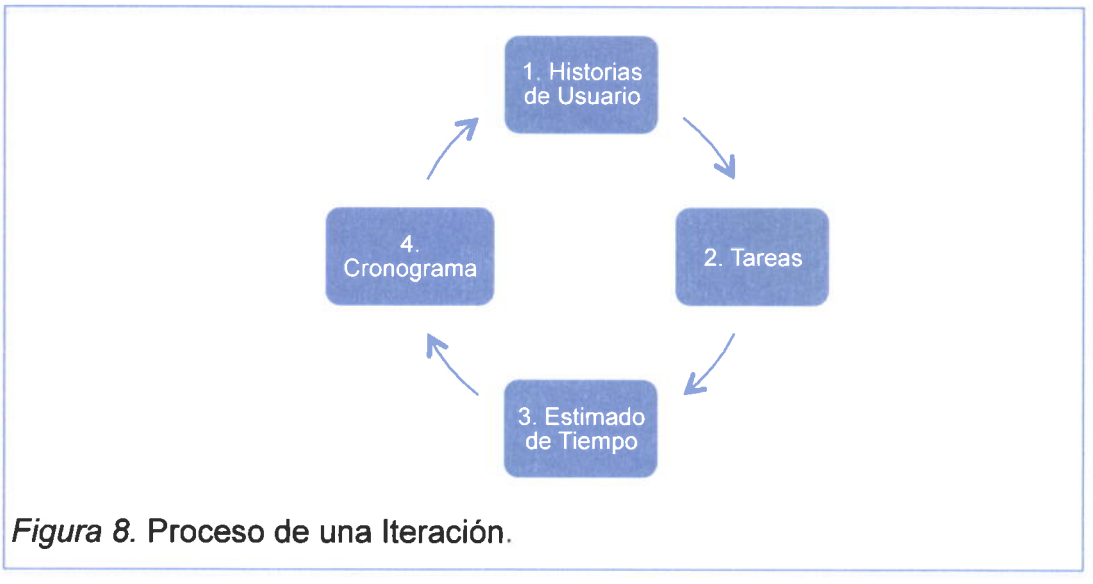


Figura 8. Proceso de una Iteración.

La planificación de una iteración comienza con una reunión dirigida por el cliente, en la cual él expresa las funcionalidades que espera del sistema y estas son registradas físicamente en tarjetas llamadas '*Historias de Usuarios*'. Estas tarjetas comunican al equipo de desarrollo las especificaciones del

proyecto a desarrollarse. Finalmente el equipo de desarrollo usa toda esta información para estimar el tiempo que tomará cada *historia de usuario* y se creará tareas por cada paso para implementar dichas historias.

1.2.2.2 La Iteración

El evento de iteración es el que comprende el desarrollo y retro-alimentación sobre las funcionalidades que el cliente espera en un tiempo no muy largo. De esta manera el cliente planifica el cronograma del proyecto según la velocidad de desarrollo de cada iteración y a su vez el equipo de desarrollo analiza cada tarea para modificar su cronograma de iteraciones o entregas.

1.2.2.3 Liberar

Liberar el software regularmente, es un evento muy importante dentro de la metodología XP, ya que de esta manera el equipo de desarrollo y el cliente analizan los resultados positivos y negativos de la iteración actual para poder mejorar y optimizar la siguiente. A su vez, el cliente contando ya con un software funcional, puede tomar decisiones acerca de este que influyan en su negocio.

1.2.3 Artefactos de la Metodología XP

XP adopta el término 'viajar ligero', el cual se refiere al hecho de preferir conversaciones, código funcional y pruebas sobre documentación y especificaciones voluminosas. Para esto XP utiliza ciertos artefactos físicos, tales como **Tarjetas de Historias, Tarjetas CRC** y ***bullpen*** o ***war room***.

1.2.3.1 Tarjetas de Historias

> "La herramienta más importante del cliente son las tarjetas de historias. Estas responden a la pregunta ¿qué se debe hacer? Cada tarjeta describe una característica deseada del proyecto de software en forma de historia en una o dos frases desde la perspectiva del cliente" (Chromatic, 2009, p. 55).

La planificación del proyecto empieza con las Tarjetas de Historias. El cliente comunica información del negocio a través de estas durante las reuniones de planificación de proyectos o iteraciones de tal manera que cada tarjeta de historia sea una funcionalidad esperada del proyecto a desarrollarse. Después estas tarjetas son analizadas por los desarrolladores quienes estiman la cantidad de tiempo que se demorarán por cada una de estas. Los desarrolladores solo pueden sugerir la creación o la modificación de historias, pero el cliente es el que toma la decisión de las sugerencias. Finalmente el cliente organiza todas las tarjetas de historias para crear un cronograma de trabajo basado en iteraciones y prioridades.

Se recomienda usar el siguiente formato para escribir una historia de usuario: "Como (rol), yo deseo (algo) para (beneficio)". Ejemplo: Como ciclista yo deseo conocer las estaciones de la biciQ, para saber hasta dónde puedo usar el servicio.

Tabla 2. Tarjeta de Historia

	IDENTIFICADOR
HISTORIA	
CONSIDERACIONES	
Usuario Solicitante:	
Fecha:	**Prioridad:**
Estimado:	*horas*
Estado:	*iniciado / en proceso / finalizado*

1.2.3.2 Tarjetas CRC

Las tarjetas CRC (clase, responsabilidad y colaboración) son una metodología para el diseño de software orientado por objetos creada por Kent Beck y Ward Cunningham.

El objetivo de la misma es hacer, mediante tarjetas, un inventario de las clases que vamos a necesitar para implementar el sistema y la forma en que van a interactuar, de esta forma se pretende facilitar el análisis y discusión de las mismas por parte de varios actores del equipo de proyecto con el objeto de que el diseño sea lo más simple posible verificando las especificaciones del sistema.

Un esquema típico de tarjeta CRC puede ser aquel en el que se indiquen los siguientes datos:

- Nombre de la clase.
- Nombre de las superclases y subclases (si procede).
- Las responsabilidades de la clase.
- Las clases con las que va a colaborar para poder realizar las responsabilidades indicadas.
- Autor, fecha, etc...

Tabla 3. Tarjeta CRC.

CLASE	
RESPONSABILIDAD	**COLABORACION**

1.2.3.3 El *Bullpen*

> "El trabajo en equipo eficaz, requiere un ambiente de trabajo propicio para la colaboración. Un buen equipo puede ser productivo en instalaciones mediocres, pero va a ser excelente en buenas instalaciones" (Chromatic, 2009, p. 56).

XP denomina *Bullpen* o *war room* al lugar donde el equipo de desarrolladores trabaja e interactúa entre si. Dicha instalación debería tener las siguientes características:

- Lugar amplio y abierto.
- Varias sillas, mesas, pizarrones y carteleras.
- Escritorios amplios (para programación en parejas)
- Pequeños cubículos u oficinas para tareas individuales o personales.
- Iluminación adecuada. (luces incandescentes o lámparas de mesa)
- Abundante espacios en las paredes.

1.2.4 Desarrollando al estilo de XP

Los objetivos de la metodología son simples. El software debe ser bien comprobado y expresivo, no debe tener funcionalidades innecesarias, los desarrolladores deben confiar en su habilidad para satisfacer las necesidades futuras y los clientes deben recibir el software que ellos realmente necesitan.

Los desarrolladores XP por lo general repiten estas tres frases para recordar el objetivo de la flexibilidad a través de la simplicidad.

"Hacer lo más simple que posiblemente funcione"

Balancear las funcionalidades con simplicidad. El código debe ser diseñado e implementado lo más simple posible, mientras pase todas las pruebas. Desarrollar con simplicidad involucra al desarrollador enfocarse más en las funcionalidades que el cliente solicito. Un código simple es fácil de entender, mantener, probar y mejorar.

> "XP produce sistemas que pueden cambiar fácilmente, no por ser lo suficientemente complejos para manejar todas las posibilidades, sino por ser lo suficientemente simples para ser cambiados" (Chromatic, 2009, p. 65).

"No vas a necesitar eso"

Hacer el trabajo de ahora, hoy, y el trabajo de mañana, mañana. Es tentador desarrollar funcionalidades futuras, pero es más importante y menos costoso trabajar en funcionalidades que son necesarias para el proceso actual.

> "Confía en el cliente para identificar las características necesarias cuando se necesiten. Resolver los problemas de hoy, hoy. Invertir en pruebas, desarrollo y en funcionalidades refactorizadas que se pagan hoy. Un código simple y bien desarrollado, deja opciones abiertas en el futuro. Agregar características cuando se las necesite, refactorizar estas dentro del diseño como si siempre hubieran estado allí" (Chromatic, 2009, p. 66).

"Una y solo una vez"

Esta frase hace acuerdo a los desarrolladores de refactorizar el código. De esta manera se elimina lo repetido, mejorando el mantenimiento y la reutilización de dicho código.

1.3. Herramientas

1.3.1 Xcode

Xcode es la principal herramienta de software para desarrollar aplicaciones para dispositivos Apple y el más completo y eficaz entorno de desarrollo integrado (*IDE, por sus siglas en ingles*).

Esta herramienta está disponible solo para equipos con sistema operativo MAC OS X, y se lo distribuye de forma gratuita, actualmente Xcode está en la versión 5 y cuenta con las siguientes características:

- **Editor Asistente:** esta característica permite dividir en dos el área principal de trabajo, ubicando en una lado de la vista el archivo con el que estamos trabajando y en el otro lado un archivo relevante o complementario a este.
- **Editor de Código:** esta característica permite escribir código utilizando un editor profesional con un avanzado terminador de código, plegado de código, resaltado de sintaxis, y mensajes de alerta que muestran advertencias, errores, y otra información contextual en la línea donde se generó la equivocación.
- **Diseñador de interfaces incorporado:** esta característica permite diseñar y probar interfaces de usuario sin escribir una línea de código, creación de prototipos en cuestión de minutos, para después ser conectadas gráficamente con el código fuente.
- **Simulador de dispositivos iOS:** esta característica permite instalar, ejecutar y depurar aplicaciones en un simulador de iOS.
- **Sistema Integrado de Compilación:** esta característica permite manejar las más complejas compilaciones, maximizando el poder de las Macs multi-core, y automáticamente permite firmar e instalar aplicaciones en un iPhone o iPad conectado.
- **Compiladores:** Dentro de Xcode o la línea de comando, dispone de un conjunto completo de código abierto de compiladores para C, C++ y

Objective-C optimizados por Apple y escalables para procesadores multi-core

- **Organizador:** El organizador es una ventana única para la gestión de proyectos, bases de SCM, archivos, aplicaciones y dispositivos iOS. El organizador es también el lugar donde se envía una aplicación a la App Store.
- **Instantáneas:** esta característica permite crear instantáneas o pequeñas copias de un proyecto, con la finalidad de ser restauradas si el proyecto se corrompe o si se desea volver a un estado anterior.
- **Refactoring:** permite reestructurar la aplicación de Objective-C en una sola operación, el cambio de las jerarquías de objetos o nombres de todas las ocurrencias dentro de su código y de diseños de interfaz de usuario.
- **Editor de Versionamiento:** soporte completo para Subversion y Git sistemas de control de código fuente (SCM).
- **Problemas en directo:** resalta los errores de ortografía y sintaxis, destaca errores comunes de codificación, sin la necesidad de compilar.
- **Ayuda Rápida:** permite obtener información sobre la API de desarrollo, dentro de la vista de código fuente.
- **Arreglos de código automático.**
- **Completa Documentación.**
- **Configuración automática:** Permite la configuración automática de aplicaciones que hagan uso de iCloud, Passbook o Game Center.
- ***Test Navigator***: Facilita y mejora el desarrollo basado en pruebas.
- **Bots para integración continua**: Traslada la carga de compilar y correr pruebas de una aplicación a un servidor Mac en la red local, de esta manera optimizamos el tiempo para el desarrollo.
- **Diseño**: Permite crear interfaces de usuario que se ajusten automáticamente al tamaño de la pantalla, orientación y localización.
- **Administrador de imágenes**: Simplifica la búsqueda y uso de todos los recursos de tipo imágenes que se usa en un proyecto de Xcode, junto con herramientas que permiten editar estos recursos.

- **Depurador de Código visual e intuitivo.**
- **Alertas y diagnóstico de consumo del *hardware* de la aplicación.**

1.3.2 Objective–C 2.0

Objective-C es el principal lenguaje de programación que se utiliza para desarrollar software para OS X, iOS y sus respectivas API's Cocoa y Cocoa Touch. Es un *superset* del lenguaje de programación C, proporciona capacidades de programación orientada a objetos y ejecución de rutinas dinámicas. Objective-C hereda la sintaxis, los tipos primitivos y las sentencias de control de flujo de C y añade sintaxis para la definición de clases y métodos.

Objective-C para organizar el código fuente y modularizar la aplicación a desarrollar, define las siguientes extensiones para los diferentes tipos de archivos:

Tabla 4. Extensiones de archivos de Objective-C

Extensión	Tipo
.h	Archivos de cabecera (*header files,* en inglés). Estos archivos contienen la declaración de las clases, tipos, funciones y constantes. Es la parte conocida como *interface de una clase.*
.m	Archivos fuentes (*source files, en inglés).* El código fuente e implementación de los métodos se encuentran en estos tipos de archivos. Es la parte conocida como la *implementación de una clase.*

Tomado de Apple Inc., 2012

Las clases en este lenguaje, como en la gran mayoría, proveen una estructura básica para encapsular propiedades, funciones, constantes, variables, etc. y así poder interactuar entre varios objetos. Los objetos son instancias de una clase en tiempo de ejecución, estos reservan un espacio de memora para alojar variables de instancias como también punteros a los métodos, ambos declarados en la clases.

Una clase puede declarar dos tipos de métodos: métodos de instancia y métodos de clase. Un método de instancia es un método cuya ejecución es de

ámbito a una instancia particular de la clase. En otras palabras, para llamar a métodos de instancia primero debemos crear una instancia de una clase particular. A diferencia de los métodos de instancia, los métodos de clase no necesitan ser llamados desde una instancia de una clase particular.

En Objective–C cuando se desea llamar a un método se conoce como enviar *un mensaje* a un objeto. *Un mensaje* es la definición del método junto con los parámetros que este debe recibir. Todos los mensajes que se envían al objeto despachados dinámicamente, facilitando así el comportamiento polimórfico de las clases de dentro de este lenguaje de programación.

Objective-C soporta variables tipadas y no tipadas (dinámicas) para contener objetos, variables tipadas son aquellas que debe incluir el nombre de la clase en la declaración del tipo de variable. Variables no tipadas son aquellas que usan el tipo de variable *id* en su declaración, este tipo de variables se usa por lo general cuando la variable que contiene el objeto puede ser reusada en algún bloque posterior por otro objeto de otra clase, de esta forma se tiene un código muy dinámico y flexible.

La versión estable y más utilizada actualmente es la 2.0, incluye las características antes mencionadas y presenta algunas nuevas como:

- Modernizado colector de variables no usadas.
- Mejoras en la sintaxis.
- Mejoras en el rendimiento.
- Soporte para arquitecturas de 64-Bits.
- Conteo referencial automático (*ARC, por sus* siglas en ingles).
- Propiedades declaradas y sintetizadas
- *Dot Syntax.*
- Enumeraciones rápidas.
- Métodos de protocolos opcionales.
- Atributos de clase/métodos/protocolos.
- Extensiones de clase.

Dado todas estas características, Objective-C es un lenguaje muy robusto y escalable, lo cual es aprovechado por Xcode para crear grandes aplicaciones para dispositivos Apple.

1.3.3 SQLite

SQLite en una pequeña librería de código que implementa un motor SQL de base de datos **auto-contenido, sin-servidor, cero-configuración y transaccional**. El código de esta librería es de dominio público, por lo tanto es libre para cualquier propósito, comercial o privado. SQLite se encuentra actualmente en la versión 3 y es usada por empresas reconocidas mundialmente como: Adobe, Airbus, Apple, Dropbox, Firefox, General Electric, Google, entre otros.

1.3.3.1 SQLite es auto-contenido.

Es a gran medida auto-contenido. Requiere soporte muy mínimo de bibliotecas externas o del sistema operativo. Esto hace que sea muy adecuado para su uso en dispositivos integrados que carecen de la infraestructura de soporte de un computador. Esto también hace que SQLite sea apropiado para aplicaciones que necesiten funcionar sin modificaciones en una amplia variedad de equipos de diferentes configuraciones.

Está escrito en ANSI-C y debe ser fácilmente codificado por cualquier compilador C estándar. Hace un uso mínimo de la biblioteca estándar de C.

1.3.3.2 SQLite no requiere servidor y es cero-configuración.

SQLite no requiere de un proceso aparte o de un proceso intermedio de comunicación, como TCP/IP, para enviar y recibir datos, lo hace directamente al archivo de base de datos que se encuentra en el disco duro del dispositivo.

No hay ningún proceso de servidor intermedio, por lo tanto no se basa en la conocida arquitectura cliente-servidor.

La principal ventaja de trabajar bajo esta arquitectura es que no hay un proceso separado para instalar, configurar, inicializar, administrar y solucionar errores. Dado esto SQLite es un motor de base de datos que necesita cero-configuración. Aplicaciones que usan esta base de datos no requieren soporte administrativo para configurar e instalar este motor de base de datos antes de ejecutar la aplicación. Cualquier aplicación o programa que sea capaz de acceder al disco duro, es capaz de usar una base de datos SQLite.

Por otro lado, un motor de base de datos que utiliza un servidor puede proporcionar una mejor protección contra errores en la aplicación de cliente. Y debido a que un servidor es un solo proceso persistente, es capaz de controlar el acceso a base de datos con más precisión, permitiendo solo bloqueos necesarios y una mejor concurrencia.

1.3.3.3 SQLite es transaccional.

SQLite implementa transacciones serializables que son atómicas, consistentes, aisladas y durables (*ACID, siglas en ingles*), incluso si la transacción es interrumpida por un fallo del programa, una caída del sistema operativo o un corte de energía hacia el dispositivo. De esta manera SQLite cumple los pilares fundamentales a lo que refiere una transacción.

1.3.3.4 Características Principales.

- Dentro de las bases de datos que no requieren servidor, SQLite es el único que permite múltiples aplicaciones accedan a la misma base de datos al mismo tiempo.
- No usa archivos de configuración.

- No hay procesos en el servidor que necesiten ser iniciados, detenidos o configurados.
- No hay necesidad que un administrador cree nuevas instancias de una base de datos, ni asigne permisos a usuarios.
- Implementa gran cantidad de sentencias del estándar *SQL92*.
- Una base de datos es alojada en un solo archivo multi-plataforma y multi-arquitectura.
- Permite bases de datos de hasta 2 *Terabytes* de tamaño y campos tipo cadena de caracteres y *BLOB* de varios Gigabyte.
- La librería ocupa muy poco espacio: menos de *400 Kibibytes* totalmente configurado o menos de *250 Kibibytes* con características opcionales omitidas.
- SQLite es más rápido que cualquier motor de base de datos de tipo cliente/servidor.
- API de fácil uso.
- Código fuente comentado y muy probado por la comunidad.
- Multi-Plataforma: Unix (Linux, Mac OS-X, Android, iOS) y Windows (Win32, WinCE, WinRT) son soportados por defecto.

1.3.4 PHP

PHP (*Hypertext Pre-processor, en ingles*) es un lenguaje de programación de código abierto, de alto nivel, no tipado y de uso general. Se compila del lado del servidor. Principalmente se usa para crear sistemas web dinámicos, ya que es un lenguaje fácil de aprender, su sintaxis es semejante a C, Java y Pearl, es muy dinámico, flexible y la característica fundamental es que se incrusta en páginas HTML, por lo tanto se puede crear aplicaciones web dinámicas en un tiempo prudente.

PHP es un lenguaje de programación muy conocido y difundido a nivel mundial, gran cantidad de sitios web visitados a diario son desarrollados en PHP. Los desarrolladores optan por este lenguaje, gracias a que es muy escalable,

multi-plataforma, soportado por la mayoría de empresas que alquilan servidores web y, es fácil acoplarlo con distintos componentes o arquitecturas.

1.3.4.1 Características Principales.

- Generar código del lado del servidor para posteriormente ser incrustado en archivos del lado del cliente de una manera transparente.
- Crear código a nivel de línea de comandos para después ser ejecutados en el mismo servidor como tareas programadas (*cron jobs*).
- Soporta el desarrollo de aplicaciones con interfaces graficas usando la extensión PHP-GTK.
- Multi-plataforma, se puede instalar en Linux, variantes de Unix, Windows, Solaris, Windows, MAC OS X, RISC OS.
- Soporta gran variedad de servidores web como: Apache, IIS, lighttpd, nginx, etc.
- PHP soporta programación estructurada, programación orientada a objetos o la posibilidad de mezclar ambas.
- Creación de imágenes, archivos PDF e incluso películas Flash (usando libswf y Ming como extensiones del lenguaje).
- Puede presentar resultados en archivos XHTML y cualquier otro tipo de archivos XML.
- Soporta gran cantidad de base de datos.
- Soporta cURL y sockets.
- PHP también cuenta con soporte para comunicarse con otros servicios usando protocolos tales como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows) y muchos otros.
- PHP posee útiles características de procesamiento de texto, las cuales incluyen Expresiones Regulares Compatibles de Perl (PCRE), muchas extensiones, y herramientas para el acceso y análisis de documentos XML.

1.3.5 MySQL Community Edition 5.5.X

MySQL es un sistema de gestión de base de datos relacional (RDBMS), multi-hilo, multi-usuario y basado en una arquitectura cliente/servidor. Es una de las base de datos de código abierto más usada a nivel mundial, posee diferentes tipos de licenciamiento, hoy en día pertenece a *Oracle Corporation*.

Entre la gran cantidad de características y funcionalidades, las siguientes son las más representativas:

- Arquitectura modular del motor de almacenamiento.
- Múltiples motores de almacenamiento: InnoDB, MyISAM, NDB (MySQL Cluster), Memory, Merge, Archive, CSV y muchos otros.
- Replicación de base de datos: para mejorar el rendimiento de las aplicaciones y la escalabilidad.
- Particionamiento de entidades de base de datos: para mejorar el rendimiento y la gestión de aplicaciones de bases de datos grandes.
- Soporta procedimientos almacenados, para mejorar la productividad del desarrollador y mejorar la seguridad de la aplicación.
- Desencadenadores: para hacer cumplir las reglas de negocio complejas a nivel de base de datos.
- Vistas: para asegurar que información sensible no sea comprometida.
- Esquema de Rendimiento: para monitoreo de consumo de recursos a nivel de aplicación o usuarios.
- Esquema de Información: para proveer un fácil acceso a la *metadata*.
- Conectores MySQL: (ODBC, JDBC, .NET, etc.) para crear aplicaciones en múltiples lenguajes.
- *MySQL Workbench*: Utilidad de administración, modelado y desarrollo de SQL.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Posibilidad de selección de mecanismos de almacenamiento que ofrecen diferente velocidad de operación, soporte físico, capacidad, distribución geográfica, transacciones.

- Transacciones y claves foráneas.
- Conectividad segura.
- Búsqueda e indexación de campos de texto.

1.3.6 REST

REST (*Representational State Transfer*), es un estilo de arquitectura de software para el diseño de aplicaciones de red. Es una arquitectura bastante simple y eficiente, ya que está basada en llamadas HTTP para realizar la trasferencia de datos entre el cliente y servidor, tal como lo hace la *World Wide Web* (www).

REST es muy versátil, ya que la información esta publicada o es accedida a través de entidades llamadas recursos. Por recursos se entiende la representación de un concepto de negocio (ej. Empleado, Usuario, Canción, etc.). Cada recurso posee un estado interno, que no puede ser accedido directamente desde el exterior. Lo que sí es accesible desde el exterior es una o varias representaciones de dicho estado. Por representación se entiende un formato de datos concreto usado para la transferencia de una copia del estado público del recurso entre el cliente y el servidor (ej. HTML, XML y JSON). Tanto los recursos REST como sus diferentes representaciones son accedidos mediante un identificador único de recurso (URI).

Las aplicaciones REST o *RESTful* utilizan solicitudes u operaciones HTTP para inter-operar con los recursos publicados dentro de esta arquitectura y son los siguientes:

- **POST:** Permite crear nuevos recursos.
- **GET:** Permite la lectura de recursos o búsqueda.
- **PUT:** Permite la actualización de recursos.
- **DELETE:** Permite la eliminación de recursos.

Esta arquitectura de comunicación actualmente es una de las preferidas por sus grandes ventajas, como:

- Al seguir los principios de HTTP, se puede aprovechar toda la infraestructura que este ofrece, como: *cache*, *proxies*, corta-fuegos, compresión, etc.
- Independiente del sistema operativo y de lenguaje de programación del cliente y del servidor.
- Múltiples representación de datos o recursos.
- La implementación y mantenimiento del código es bastante simple.
- Recursos y acciones auto-descubribles, no es necesario un WSDL.

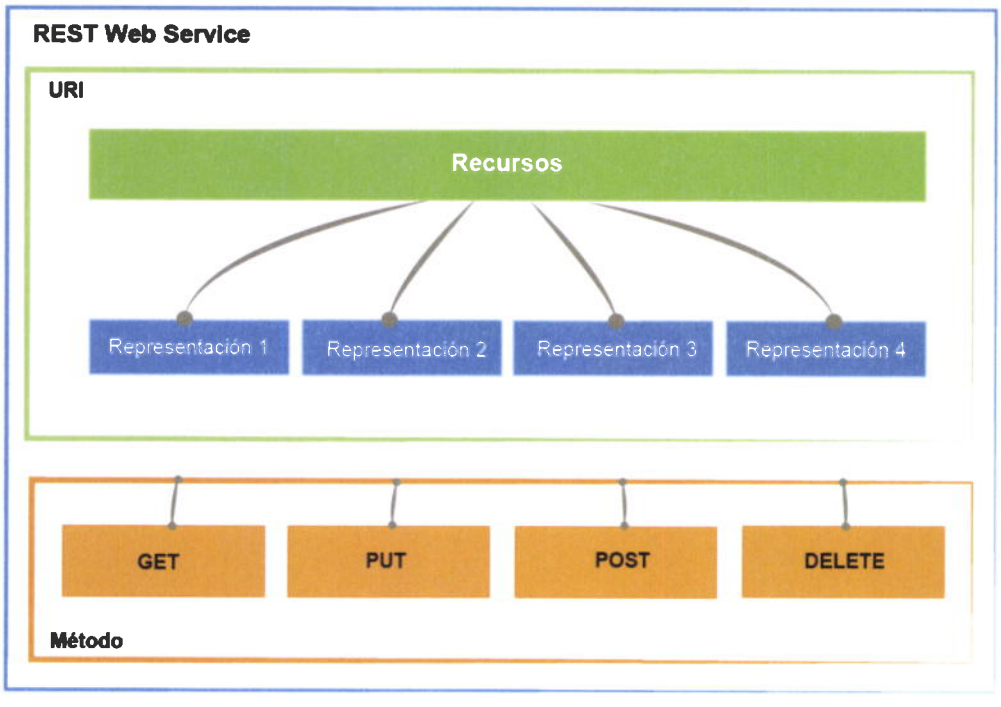


Figura 9. Arquitectura REST

Tomado de Bluk, 2009

1.4. Plataformas

1.4.1 iOS 7.x

Sistema operativo de altas prestaciones desarrollado por Apple, para el dispositivo iPad, iPhone, y iPod Touch. Este sistema operativo no permite la instalación en *hardware* de terceros.

iOS es muy parecido al sistema operativo MAC OS X, sistema operativo que está instalado en *laptops* y *Desktops* Apple, ya que está basado en este, por lo tanto iOS es un sistema operativo basado en Unix (distribución Darwin BSD).

Ya que este sistema operativo está diseñado y probado solo para dispositivos modernos Apple con pantalla táctil, involucra que este *software* sea muy intuitivo y a la medida, aprovechando al máximo la capacidad del hardware del dispositivo. Entre las características más relevantes se encuentran las siguientes:

- Siri: módulo de control por voz.
- iCloud: almacenamiento y administración de respaldos en la *nube.*
- Actualización del sistema operativo *on-the-fly*
- Mapas: indicaciones paso a paso (voz y texto), información de tráfico en tiempo real, vistas en 3D, alta resolución, etc.
- Facebook: totalmente integrado entre todas las aplicaciones del sistema.
- Passbook: aplicación del sistema para gestión de entradas y boletos digitales de servicios de terceros.
- FaceTime: videoconferencia por redes de datos o red de inalámbrica de área local.
- Teléfono, Mail y Safari.
- Soporte para accesibilidad.
- Cámara con soporte modalidad panorámica.
- Soporte para más de 30 idiomas.
- Funcionalidad de comprar desde una aplicación.

- AirDrop: Compartir documentos, imágenes y otros recursos con varios dispositivos que soporten AirDrop.
- Optimizado para multi-tareas y arquitecturas de 64-bits.

1.4.2 iPhone 4

Es uno de los teléfonos inteligentes más avanzados y usados en el mundo, este dispositivo es fabricado por Apple y corre bajo el sistema operativo iOS, desarrollado por la misma empresa. El primer modelo de esta teléfono inteligente fue lanzado a mediados del año 2007, causando gran conmoción a todo el mundo, por las características muy innovadores que presentaba en ese entonces. El iPhone 4 fue lanzado en el año 2010, tuvo una enorme acogida, según el Blog *AppleInsider* (AppleInsider Staff, 2010) se vendieron 1.7 millones de dispositivos iPhone 4 solo 3 días después de su lanzamiento.

Figura 10. iPhone 4 de color negro.

Tomado de Miller, 2010

1.4.2.1 Características Principales

- Peso: 137 gramos.
- Soporta las siguientes tecnologías celulares: UMTS/HSDPA/HSUPA (850, 900, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz); CDMA EV-DO Rev. A (800, 1900 MHz)
- Soporta tecnología inalámbrica: Wi-Fi 802.11b/g/n (solo 802.11n de 2.4 GHz) y tecnología *wireless* Bluetooth 2.1 + EDR.
- Soporta servicio localización basado en los siguientes tipos de fuentes: GPS asistido, Brújula digital, Wi-Fi y tecnología celular.
- Dispone de pantalla LCD Retina *multi-touch* de 3.5 pulgadas y con resolución de 960x480 pixels con una densidad de 326 puntos por pixel.
- *CPU* de 800Mhz modelo Apple A4-ARM Cortex A8 y cuenta con una *GPU* PowerVR SGX 535.
- Memoria RAM de 512MB.
- Dispone de una cámara de fotos trasera de 5 Megapixeles de resolución y una cámara trasera de 0.3 Megapixeles de resolución.
- Soporta grabación de video en calidad HD (720p) desde la cámara trasera y calidad VGA desde la cámara frontal.
- Incluye varios tipos de sensores: Giroscopio de 3 ejes, acelerómetro, sensor de proximidad y sensor de luz ambiental.

1.4.3 Servidor Web Apache

Nació y se ha mantenido como un proyecto de desarrollo de software para la comunidad cuyo primordial objetivo es desarrollar y mantener un servidor HTTP de código abierto para sistemas operativos modernos, incluyendo UNIX y Windows NT. Proporcionando un servidor seguro, eficiente y extensible que ofrezca servicios HTTP que se encuentren ya en los estándares HTTP actuales.

Apache httpd ha sido el servidor web más popular en Internet desde abril del año 1996. Sitios web como Youtube, Wikipedia, Craiglist, AVG, Ask, entre otros han decidido optar por este servidor HTTP. Según *W3Techs, Web Technology Surveys*, 63.5% de los Servidores encuestados por este sitio web, están instalados alguna versión de Apache.

1.4.3.1 Características Principales

- Es un potente y flexible servidor Web que soporta HTTP/1.1.
- Implementa los últimos protocolos HTTP incluyendo HTTP/1.1 (RFC2616).
- Es altamente configurable y extensible con módulos de terceros.
- Existe la posibilidad de crear módulos personalizados utilizando el API de desarrollo para módulos.
- Proporciona el código fuente completo y viene con una licencia sin restricción alguna.
- Soporta Windows 2000, Netware 5.x y superior, OS/2 y la mayoría de versiones Unix, así como varios otros sistemas operativos.
- Hay bastante participación de la comunidad y de desarrolladores que aportan con sus conocimientos para mantener y mejorar el código fuente.
- Alienta a los usuarios a generar nuevas ideas, informes de errores y parches.
- Implementa las siguientes tecnologías o módulos:
    - Base de datos DBM como base de datos relacional y el soporte para autenticación por medio del protocolo LDAP.
    - Permite configurar fácilmente páginas protegidas con contraseña con un enorme número de usuarios autorizados, sin atascos en el servidor.
    - Repuestas personalizadas frente a errores y problemas.

- Cantidad ilimitada de alias y redirecciones que se asocian a la URL.
- Negociación de Contenido.
- *Hosts* Virtuales.

1.5 Pruebas

El enfoque de la metodología XP frente a pruebas establece que si el código se prueba pocas veces, se podrá eliminar pocas fallas, pero si se prueba extensivamente se eliminaran un gran porcentaje de fallas. Realizando pruebas continuas se puede decidir si ciertas funcionalidades del producto están bien o si necesitan de cambios. Este tipo de decisiones basados en pruebas continuas supone un gran ahorro en el presupuesto del proyecto.

Las pruebas unitarias determinan si una característica establecida funciona como esta prevista. Las pruebas se realizan para romper el código, si todas las pruebas resultan satisfactorias, entonces la codificación esta completa y se debe proceder a codificar la siguiente característica.

Las pruebas de aceptación verifican que los requerimientos entendidos por el equipo de desarrollo satisfagan los requerimientos actuales del cliente.

1.5.1 Pruebas Unitarias

Las pruebas unitarias son un pilar fundamental en la metodología XP. Estas estipulan que primero, se debería usar una librería o *framework* en el código que permitan la creación de suites de pruebas automatizadas. Segundo, se debería probar todas las clases en el sistema.

Estas pruebas deberían ser creadas antes que el código para ser liberadas conforme se desarrolla los módulos a probar. El código que no disponga de

estas pruebas no debe ser liberado, al menos que se cree las respectivas pruebas unitarias en ese momento.

Crear pruebas unitarias automatizadas conlleva un gran ahorro de tiempo y de codificación, como resultado ofrece muchas mas ventajas que el tiempo gastado en crear dichas pruebas.

Este tipo de pruebas proporciona una red segura de pruebas para que el equipo de desarrollo pueda refactorizar e integrar efectivamente el código. El hecho de crear las pruebas antes que el código ayuda aun más a solidificar los requerimientos y a mejorar el enfoque del equipo.

1.5.2 Pruebas de Aceptación

Las pruebas de aceptación se crean a partir de las historias de usuario, dichas historias son seleccionadas en las iteraciones. El cliente especifica el escenario para realizar estas pruebas y verificar que se haya implementado correctamente. Una historia de usuario pueden tener las pruebas de aceptación necesarias para asegurar que la funcionalidad a probarse funcione.

Estas pruebas pertenecen a la categoría de pruebas de caja negra. Por caja negra se entiende al tipo de pruebas en las cuales la persona que va a probar la aplicación, debe conocer la información a ser ingresada como también el resultado de dicho ingreso, sin la necesidad de conocer como el resultado fue procesado internamente. Este tipo de pruebas se ejecuta en programas compilados, en el cual el desarrollador es el único que puede hacer alteraciones a través de una interface expuesta.

Cada prueba de aceptación representa algún resultado que se espera de la aplicación. Los clientes son responsables de verificar la exactitud de las pruebas de aceptación y revisión de resultados de las mismas, para decidir que prueba fallida tiene la mas alta prioridad.

Una historia de usuario no es considerada completa hasta que todas sus pruebas de aceptación relacionas hayan sido ejecutadas y verificadas correctamente.

1.5.3 Pruebas de Caja Blanca

Son también conocidas como pruebas estructurales o de caja de cristal. Este tipo de pruebas permite al evaluador tener acceso a los métodos y estructuras internas de la aplicación a ser probada. Esta persona mira el código y usa toda la información extra obtenida, para asegurarse que la aplicación funciona como debería. El objetivo de este tipo de pruebas, es generalmente, encontrar defectos específicos de la implementación, cuyos errores no podrán se detectados por pruebas de caja negra.

Estas pruebas permiten una evaluación mucho mas detallada, eso significa que las pruebas están dictaminadas por el diseño del código. Dado la prolijidad, este tipo de pruebas están enfocadas en pequeños componentes cuya verificación es particularmente importante.

2. Análisis y Diseño

2.1. Planificación Inicial

El objetivo principal de la planificación inicial es detallar: los recursos humanos, tecnológicos y el tiempo que involucrará el desarrollo de la aplicación móvil, basado en los requerimientos iniciales o en el alcance.

Recursos Humanos: Un desarrollador y un diseñador de interfaces/experiencias (UI/UX).

Recursos Tecnológicos: Computador con sistema operativo MAC OS X 10.8 y un dispositivo móvil iPhone 4 (iOS 7).

Tiempos:

Tabla 5. Planificación Inicial

ETAPA	TIEMPO	DESARROLLADOR	DISEÑADOR
Diseño Interfaces	20 horas		x
Diseño de Experiencias	6 horas	x	x
Módulo Registro/Login App	8 horas	x	
Módulo Información	10 horas	x	
Módulo Noticias	12 horas	x	
Módulo Mapa Dinámico e interactivo	48 horas	x	
Módulo Redes Sociales (Facebook)	12 horas	x	
Módulo Configuración de Cuenta	5 horas	x	
Servicios Web	6 horas	x	
TOTAL HORAS		**107 HORAS**	**26 HORAS**

2.2 Historias de Usuario

Esta herramienta o artefacto de la metodología XP, se utiliza para documentar las funcionalidades que se espera que otorguen a los usuarios finales, basadas en las ideas del propio cliente.

Cada historia de usuario presenta los siguientes campos:

- **ID**: Identificador único de cada historia de usuario.
- **Historia**: Descripción de la funcionalidad en palabras del cliente.
- **Consideraciones**: Detalles secundarios sobre la historia de usuario.
- **Usuario Solicitante**: Nombre y Apellidos del cliente que escribió la historia de usuario.
- **Fecha**: Fecha en la cual la historita de usuario fue escrita.
- **Prioridad:** El nivel de importancia que el cliente desea frente a esta historia.
- **Estimado:** Cantidad de horas en la que el cliente espera que se complete la historia de usuario.
- **Estado:** El estado que el equipo de desarrollo define.

Estas historias fueron redactadas por un cliente.

Tabla 6. Historia de Usuario 1

	ID: 1
HISTORIA: Quisiera que la aplicación sea compatible con el sistema operativo de un iPhone	
CONSIDERACIONES: -Gratuito -Que no sea pesada -Que se vea bien en clima soleado	
Usuario Solicitante:	Ramiro de la Torre Freire
Fecha: 26-Jun-13	**Prioridad:** *alta* / *media* / *baja*
Estimado:	horas
Estado:	*iniciado* / *en proceso* / *finalizado*

Tabla 7. Historia de Usuario 2

	ID: 2
HISTORIA: El acceso sea a través de un usuario y una clave	
CONSIDERACIONES: - usuario: nombre corto - clave: 2 dígitos	
Usuario Solicitante:	Ramiro de la Torre Freire
Fecha: 26-Jun-13	**Prioridad:** *alta* / *media* / *baja*
Estimado:	horas
Estado:	*iniciado* / *en proceso* / *finalizado*

Tabla 8. Historia de Usuario 3

	ID: 3
HISTORIA: Crear una cuenta directamente desde el smartphone, considerando nombres, apellidos, correo electrónico y una contraseña	
CONSIDERACIONES: *Posibilidad de enlazar con Facebook personal. *El e-mail debería ser la cuenta del usuario.	
Usuario Solicitante:	Ramiro de la Torre Freire
Fecha: 26-Jun-13	**Prioridad:** *alta / media / baja*
Estimado:	horas
Estado:	*iniciado / en proceso / finalizado*

Tabla 9. Historia de Usuario 4

	ID: 4
HISTORIA: Tener información de uso de la aplicación biciQ, si es posible bien detallada (referencias, fotos, videos)	
CONSIDERACIONES: - sistemas operativos compatibles, servicios disponibles, costos, contactos	
Usuario Solicitante:	Ramiro de la Torre Freire
Fecha: 26-Jun-13	**Prioridad:** *alta / media / baja*
Estimado:	horas
Estado:	*iniciado / en proceso / finalizado*

Tabla 10. Historia de Usuario 5

	ID: 5
HISTORIA: Estar enlazado con la página biciQ de Facebook	
CONSIDERACIONES: últimas 20 publicaciones resumen	
Usuario Solicitante:	Ramiro de la Torre Freire
Fecha: 26-Jun-13	**Prioridad:** *alta / media / baja*
Estimado:	horas
Estado:	*iniciado / en proceso / finalizado*

Tabla 11. Historia de Usuario 6

	ID: 6
HISTORIA: Las paradas deberían estar representadas en un mapa.	
CONSIDERACIONES: - Paradas bien identificadas - Direcciones exactas - Referencias	
Usuario Solicitante:	Ramiro de la Torre Freire
Fecha: 26-Jun-13	**Prioridad:** *alta / media / baja*
Estimado:	horas
Estado:	*iniciado / en proceso / finalizado*

Tabla 12. Historia de Usuario 7

	ID: 7
HISTORIA: Que me permita crear una ruta entre paradas o entre mi ubicación actual y una parada, en el mapa	
CONSIDERACIONES: - Que se presente la ruta en color llamativo - Velocidad promedio, tiempo estimado y distancia faltante. Cuando me movilizo en una ruta creada	
Usuario Solicitante:	Ramiro de la Torre Freire
Fecha: 26-Jun-13	**Prioridad:** *alta / media / baja*
Estimado:	horas
Estado:	*iniciado / en proceso / finalizado*

Tabla 13. Historia de Usuario 8

		ID: 8
HISTORIA: Quisiera que la ruta recorrida, se pueda publicar en el FB personal		
CONSIDERACIONES: - Opcional		
Usuario Solicitante:	Ramiro de la Torre Freire	
Fecha: 26-Jun-13	**Prioridad:** *alta / media / baja*	
Estimado:		
Estado:	*iniciado / en proceso / finalizado*	

Tabla 14. Historia de Usuario 9

	ID: 9
HISTORIA: Una vez ingresado a la aplicación, permitir: • Eliminar cuenta • Cambio de contraseña • Cambio de correo electrónico	
CONSIDERACIONES: - Botón de opciones visible y de fácil acceso	
Usuario Solicitante:	Ramiro de la Torre Freire
Fecha: 26-Jun-13	**Prioridad:** alta / *media* / *baja*
Estimado:	horas
Estado:	*iniciado* / *en proceso* / *finalizado*

2.3 Planificación de Iteraciones

Nombre de tarea	Duración	Comienzo	Fin
ITERACIÓN 1	5 días	mar 01/10/13	lun 07/10/13
Quisiera que la aplicación sea compatible con el sistema operativo de un iPhone	1 día	mar 01/10/13	mar 01/10/13
Crear una cuenta directamente desde el smartphone, considerando nombres, apellidos, correo electrónico y una contraseña	2 días	mié 02/10/13	jue 03/10/13
El acceso sea a través de un usuario y una clave	2 días	vie 04/10/13	lun 07/10/13
ITERACIÓN 2	12 días	mar 08/10/13	mié 23/10/13
Tener información de uso de la aplicación bicIQ, si es posible bien detallada (referencias, fotos, videos)	4 días	mar 08/10/13	vie 11/10/13
Estar enlazado con la página bicIQ de Facebook	5 días	lun 14/10/13	vie 18/10/13
Una vez ingresado a la aplicación, permitir: Eliminar la cuenta, cambio de contraseña, cambio de correo electrónico	3 días	lun 21/10/13	mié 23/10/13
ITERACIÓN 3	16 días	jue 24/10/13	jue 14/11/13
Las paradas deberían estar representadas en un mapa	4 días	jue 24/10/13	mar 29/10/13
Que me permita crear una ruta entre paradas o entre mi ubicación actual y una parada, en le mapa	10 días	mié 30/10/13	mar 12/11/13
Quisiera que la ruta recorrida, se pueda publicar en el FB personal	2 días	mié 13/11/13	jue 14/11/13

Figura 11. Iteraciones

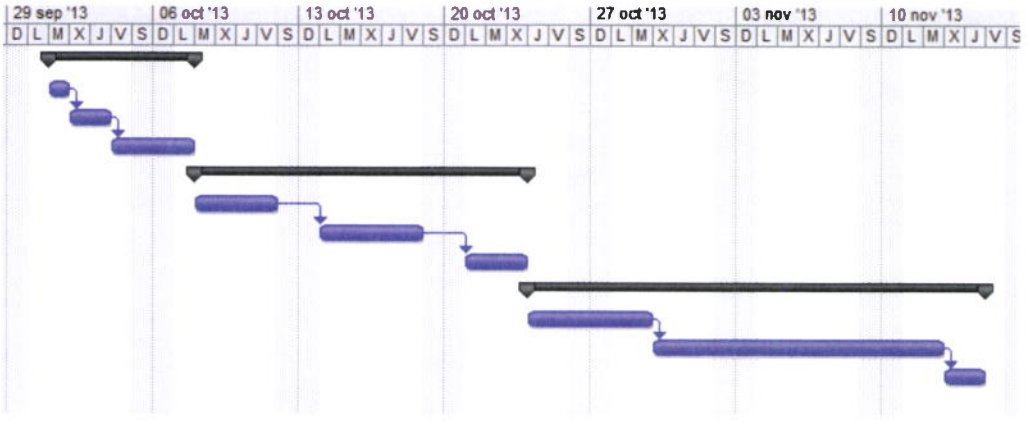


Figura 12. Diagrama de Gantt - Iteraciones

2.4 Tarjetas CRC

Esta herramienta o artefacto grupal de la metodología XP, se utiliza para documentar los nombres de las clases, sus responsabilidades y cómo colaboran con otras clases para realizar tareas; cada tarjeta representa una clase en el sistema que a su vez formaran parte en el diagrama de clases.

Además las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos y que el equipo completo contribuya en la tarea del diseño.

Las principales características de esta herramienta son:

- Identificación de clases y asociaciones que participan del diseño del sistema.
- Obtención de las responsabilidades que debe cumplir cada clase.
- Establecimiento de cómo una clase colabora con otras clases para cumplir con sus responsabilidades.

Contienen los siguientes campos:

- **Nombre de la clase**
- **Responsabilidades de la clase**: describen el propósito de la existencia de la clase, normalmente una clase no debe tener más de tres o cuatro responsabilidades. Si tiene más, habría que plantearse describirla de forma más concisa.
- **Colaboradores de la clase**: Clases que ayudan a ejecutar una responsabilidad.

Tabla 15. Tarjeta CRC USUARIO

USUARIO	
RESPONSABILIDAD	**COLABORACION**
Crear Usuario	
Editar Usuario	
Autentificar Usuario	
Compartir Ruta	Ruta
	FacebookAPI

La tarjeta CRC USUARIO, permite la creación, edición y autenticación de un usuario en la aplicación móvil, así como la interacción con la red social Facebook.

Tabla 16. Tarjeta CRC INFORMACION BICIQ

INFORMACIONBICIQ	
RESPONSABILIDAD	**COLABORACION**
Recuperar la información de la biciQ	

La tarjeta CRC INFORMACIONBICIQ, recupera los datos generales e informativos sobre el servicio biciQ desde la base de datos.

Tabla 17. Tarjeta CRC NOTICIAFACEBOOK

NOTICIAFACEBOOK	
RESPONSABILIDAD	**COLABORACION**
Recuperar entradas	FacebookAPI

La tarjeta CRC NOTICIAFACEBOOK, recupera las entradas publicadas en la pagina principal de la biciQ en Facebook.

Tabla 18. Tarjeta CRC ESTACION

ESTACION	
RESPONSABILIDAD	**COLABORACION**
Obtener información de la estación	Estacion
Listar estaciones	Estacion

La tarjeta CRC ESTACION, permite visualizar las diferentes estaciones del servicio biciQ en el mapa de la aplicación móvil.

Tabla 19. Tarjeta CRC RUTA

RUTA	
RESPONSABILIDAD	**COLABORACION**
Calcular ruta	Usuario
	Estacion
Graficar ruta	
Calcular distancia total de la ruta	Estacion
Calcular tiempo total que tomará al usuario completar ruta	Estacion
	Usuario
Calcular tiempo restante que le toma al usuario completar la ruta	Estacion
	Usuario

La tarjeta CRC RUTA, permite el cálculo y la visualización de una ruta con datos interesantes para el usuario.

2.3 Diagramas de Clases UML

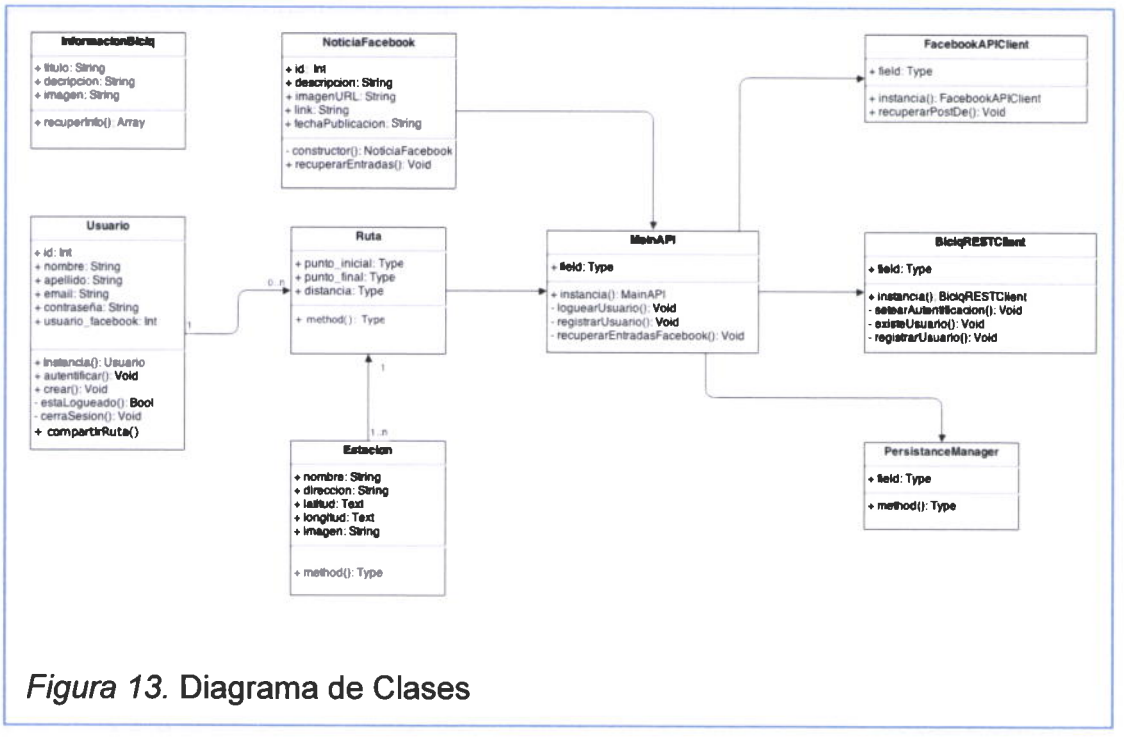


Figura 13. Diagrama de Clases

2.4 Diagrama de Base de Datos

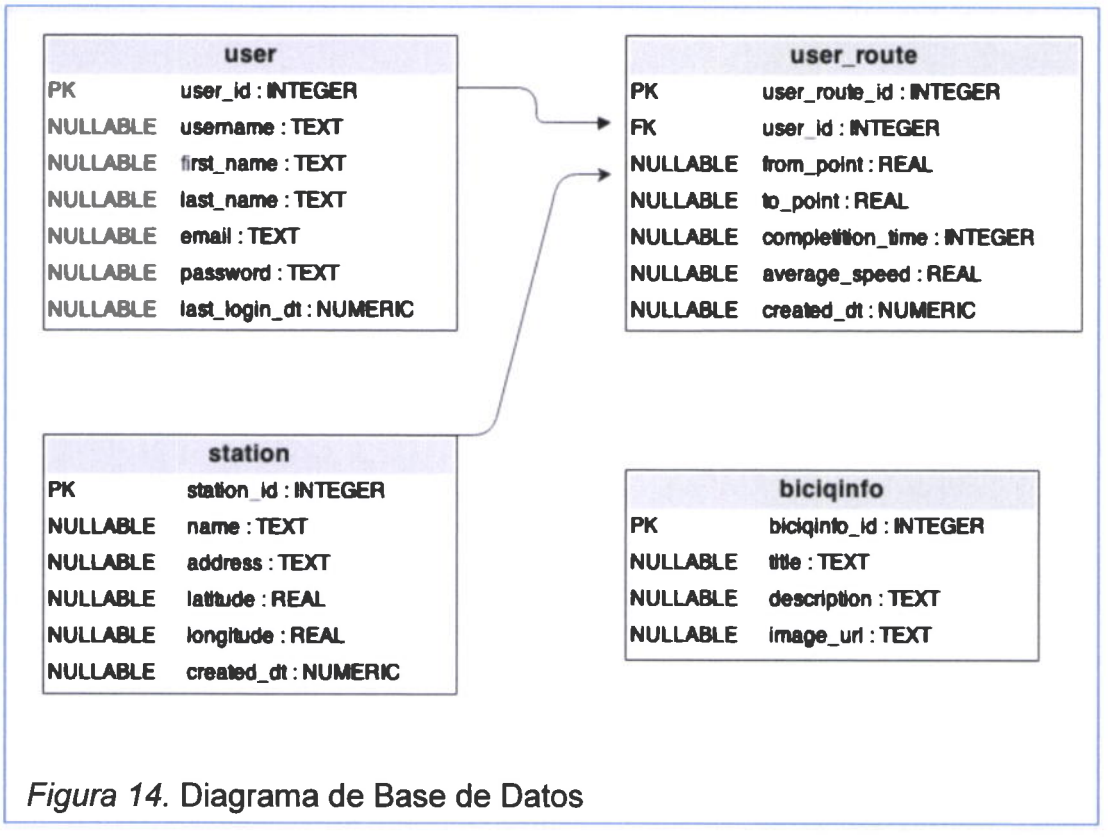


Figura 14. Diagrama de Base de Datos

2.5 Descripción Servicio Web RESTful

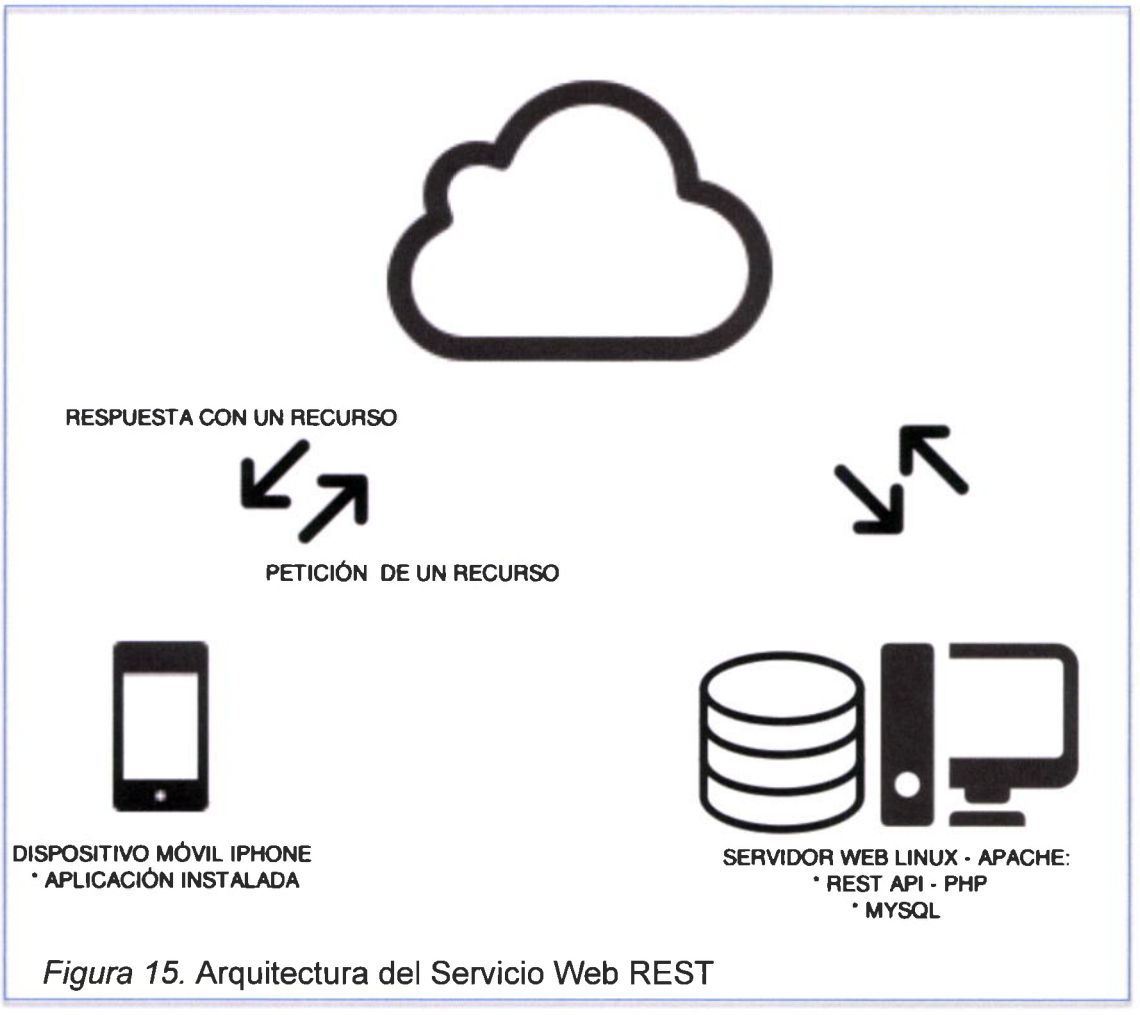


Figura 15. Arquitectura del Servicio Web REST

Tabla 20. Servicio REST POST usuario/autenticar

POST usuario/autenticar	
URL	http://rest.biciq.com/usuario/autenticar
Formato de Respuesta	JSON
Objeto de Respuesta	Usuario
Método HTTP	POST
Tipo Autenticación	Básico (usuario y contraseña)
PARAMETROS	
email	Email del usuario a ser autenticado (requerido)
password	Contraseña del usuario a ser autenticado (requerido)
CODIGOS DE RESPUESTA	

200 (OK)	Usuario encontrado y se devuelve un objeto de tipo Usuario
404 (Not Found 400 (Bad Request)	Usuario no encontrado, se devuelve un objeto de tipo usuario vacío.

Este método se utiliza para autentificar al usuario desde la aplicación móvil.

Ejemplo de Petición:

POST: http://rest.biciq.com/usuario?email=juanjdlt@gmail.com&password=12345

```
[
    {
        "id": "1"
        "nombres": "Juan José",
        "apellidos": "De la Torre León",
        "email": juanjdlt@gmail.com,
        "facebook": "1"
    }
]
```

Tabla 21. Servicio REST POST usuario

POST usuario	
URL	http://rest.biciq.com/usuario
Formato de Respuesta	JSON
Objeto de Respuesta	Usuario
Método HTTP	POST
Tipo Autenticación	Básico (usuario y contraseña)
PARAMETROS	
nombres	Nombres del usuario a ser creado (requerido)
apellidos	Apellidos del usuario a ser creado (requerido)
email	Email del usuario a ser creado (requerido)
password	Contraseña del usuario a ser creado (requerido)
facebook	Campo de tipo entero, si es 1 el usuario creado podrá compartir las rutas que recorra.

CODIGOS DE RESPUESTA	
201 (Created)	Usuario registrado y devuelve un objeto de tipo Usuario con los datos del usuario creado.
500 (Internal Server error) 400 (Bad Request)	Usuario no registrado, se devuelve un objeto de tipo usuario vacío.

Este método se utiliza para realizar el registro de un nuevo usuario desde la aplicación móvil.

Ejemplo de Petición:

POST: http://rest.biciq.com/usuario/

POST DATA:

nombres=JuanJosé&
apellidos=De La Torre&
email=juanjdlt@gmail.com
&password=123456
&facebook=0

```
[
    {
        "id": "1"
        "nombres": "Juan José",
        "apellidos": "De la Torre",
        "email": juanjdlt@gmail.com,
        "facebook": "0"
    }
]
```

Tabla 22. Servicio REST PUT usuario

PUT usuario	
URL	http://rest.biciq.com/usuario/<id>
Formato de Respuesta	JSON
Objeto de Respuesta	Usuario
Método HTTP	PUT
Tipo Autenticación	Básico (usuario y contraseña)

PARAMETROS	
nombres	Nombres del usuario a ser modificado (requerido)
apellidos	Apellidos del usuario a ser modificado (requerido)
email	Email del usuario a ser modificado (requerido)
password	Contraseña del usuario a ser modificado (requerido)
facebook	Campo de tipo entero, si es 1 el usuario creado podrá compartir las rutas que recorra.
CODIGOS DE RESPUESTA	
200 (OK)	Usuario modificado. Devuelve un objeto de tipo Usuario con los datos del usuario actualizado.
500 (Internal Server Error) 400 (Bad Request) 404 (Not Found)	Usuario no modificado, se devuelve un objeto de tipo usuario vacío.

Este método se utiliza para realizar la modificación de un usuario desde la aplicación móvil.

Ejemplo de Petición:

PUT: http://rest.biciq.com/usuario/3

PUT DATA:

nombres=JuanJosé&
apellidos=De La Torre León&
email=juanjdlt@gmail.com
&password=123456
&facebook=1


```
[
    {
        "id": "1"
        "nombres": "Juan José",
        "apellidos": "De la Torre León",
        "email": juanjdlt@gmail.com,
        "facebook": "1"
    }
]
```

Tabla 23. Servicio REST DELETE usuario

DELETE usuario	
URL	http://rest.biciq.com/usuario/<id>
Formato de Respuesta	JSON
Objeto de Respuesta	Confirmación (booleano)
Método HTTP	DELETE
Tipo Autenticación	Básico (usuario y contraseña)
CODIGOS DE RESPUESTA	
200 (OK)	Usuario eliminado y se devuelve un objeto de tipo confirmación
404 (Not Found 500 (Internal Server Error)	Usuario no eliminado, se devuelve un objeto de tipo confirmación.

Este método se utiliza para eliminar lógicamente el usuario desde la aplicación móvil.

Ejemplo de Petición:

DELETE: http://rest.biciq.com/usuario/5

```
[
    {
        "deleted": true
    }
]
```

2.6 Estándares de Codificación

Los lineamientos mencionados a continuación se establecieron con la finalidad de obtener un código claro, organizado y consistente. Unas de las principales ventajas de crear y utilizar una guía de codificación, como esta, es optimizar el mantenimiento del código y que para un nuevo desarrollador o equipo de desarrollo se le facilite la comprensión del mismo.

Sintaxis de *dot-sintax*

Dot-sintax se debe utilizar siempre para acceder y mutar las propiedades de una clase o objeto. Es recomendado usar la notación-por-corchetes para todos los demás casos.

Uso Correcto:

```
view.colorDeFondo = [UIColor orangeColor];
[UIApplication sharedApplication].delegate;
```

Uso Incorrecto:

```
[view setColorDeFondo:[UIColor orangeColor]];
UIApplication.sharedApplication.delegate;
```

Espaciado

- Usar tabulador, no espacios.
- Terminar un archivo con una nueva línea.
- Se recomienda usar espaciado vertical para separar el código en bloques lógicos.
- Debe haber exactamente una línea en blanco entre métodos para ayudar en la claridad visual y la organización. Los espacios en blanco dentro de los métodos deben separar la funcionalidad.

Estructuras de Control

- Las llaves de las estructuras de control (if, else, switch, for, etc.) se deben abrir en la misma línea que la declaración.
- Se debe colocar un solo espacio después de las palabras clave y antes de su paréntesis.
- No usar espacios entre los paréntesis y su contenido.
- El operador ternario (?:) deberá ser usado solo cuando vaya a incrementar la claridad o limpieza del código. Y se deberá evaluar solo una condición.

Uso correcto:

```
if (variable == nil) {
    // hacer algo
} else {
    // hacer otra cosa
}

if (!error) {
    return resultado;
}

resultado = a > b ? x : y;
```

Uso incorrecto:

```
if (variable == nil)
{
    // hacer algo
}
else {
    // hacer otra cosa
}

if (!error)
    return resultado;

if (!error) return resultado;


resultado = a > b ? x = c > d ? c : d : y;
```

Excepciones y Manejo de Errores

- No usar excepciones como parte de la lógica.
- Usar excepciones únicamente para indicar errores del programador.
- Cuando los métodos devuelven un parámetro de error por referencia, se debe evaluar el valor devuelto, no la variable de error.

Uso correcto:

```
NSError *error;
if (![self metodoConError:&error]) {
    // Procesar el error
}
```

Uso incorrecto:

```
NSError *error;
[self metodoConError:&error];
if (error) {
   // Procesar el error
}
```

Métodos

- El nombre de los métodos es de tipo *camelCase*.
- Usar un nombre único con respecto a la clase que lo contiene que sea claro y muy explicativo.
- Se debe evitar usar abreviaturas en el nombre, al menos que se utilice una abreviatura común y conocida. Como las listadas en esta dirección de internet: http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CodingGuidelines/Articles/APIAbbreviations.html#//apple_ref/doc/uid/20001285
- Documentar los métodos que realmente necesiten explicación.
- No usar salto de línea en la declaración de un método.
- En las firmas de los métodos debe haber un espacio después de la declaración del ámbito (símbolo +/-) y un espacio entre cada segmento.

```
- (void)setTextoEj:(NSString *)texto image:(UIImage *)imagen;
```

- A menos que se especifique lo contrario, el método accesor *getter* debe utilizar el mismo nombre que la propiedad, excepto cuando las propiedades son de tipo booleanas, en dicho caso el método accesor *getter* debe empezar con la palabra '*is*' seguido por el nombre de la propiedad, por ejemplo: `isVisible`.
- El método accesor *setter* deber empezar con la palabra '*set*' seguido por el nombre de la propiedad, por ejemplo: `setFirstName`.

Variables, Propiedades y Constantes:

- Las variables deben ser nombradas de la manera más descriptiva posible. Nombres con una sola letra deben ser evitados, excepto en las estructuras de control *for()*.
- Los asteriscos que indican punteros, pertenecen a la variable, excepto en el caso de constantes.

Uso Correcto:

```
NSString *texto
NSString * const MI_CONSTANTE;
```

Uso Incorrecto:

```
NSString* texto
NSString * texto
```

- La definición de propiedades deben ser usados en lugar de variables de instancia cortas. El acceso directo a las variables de instancia deben evitarse, excepto en métodos de inicialización (`init`, `initWithCoder:`, `etc.`), métodos `dealloc` y dentro de métodos personalizados de seteo y acceso (accesores).

Uso Correcto:

```
@interface NYTSeccion: NSObject
@property (nonatomic) NSString *titulo;
@end
```

Uso Incorrecto:

```
@interface NYTSection: NSObject {
        NSString *titulo;
}
```

- El nombre de las variables debe basarse en la estructura *camelCase*.
- Siempre declarar la semántica para el manejo de memoria, incluso en propiedades de tipo `readonly`.
- Declarar las propiedades como `readonly`, si estas han sido seteadas una sola vez en `-init`.
- Declarar las propiedades como `copy`, si estas devuelven objetos inmutables y nunca son mutadas a lo largo de la implementación.
- No usar `@synthesize`, al menos que el compilador lo requiera.
- Las variables de instancia deben ser precedidas por un guion bajo. (al igual que cuando se sintetiza implícitamente).
- Las variables locales deben ser únicas dentro del mismo ámbito.
- Las variables globales deben ser precedidas por la letra 'g'.
- Variables locales no deben contener guión bajo en su nombre.
- Al usar propiedades o variables de instancia, estas siempre deben ser accedidas o mutadas usando la palabra clave `self`.
- Utilizar mayúsculas para definir el nombre de una constante.
- Constantes deben ser declaradas como constantes de tipo `static`.
- Siempre usar un prefijo de tres letras global o relativo a la clase para el nombre de las constantes. Ejemplo: `MAP`

Clases y Objetos

- El nombre de una clase debe ser único con respecto a todo el proyecto, debe ser un sustantivo y debe contener un prefijo de 3 letras relativo al proyecto o al nombre de la empresa que esta creando el código.
- Para crear una instancia de una clase (un objeto), se deben seguir ciertas convenciones:

```
NSMutableArray *arreglo = [[NSMutableArray alloc] init];
```

ó

```
NSMutableArray *arreglo = [NSMutableArray new];
```

ó

```
NSMutableArray *arreglo = [NSMutableArray array]; //método
factory de la clase
```

Literales

- Los literales `NSString`, `NSDictionary`, `NSArray`, y `NSNumber` deben ser usados siempre al momento de crear instancias inmutables de dichos objetos.
- No pasar valores `nil` a una instancia de tipo `NSArray` o `NSDictionary`, ya que esto genera errores.

Uso correcto:

```
NSArray *nombres = @[@"Andres", @"Juan", @"Jose", @"Alex",
@"Paul"];
NSDictionary *productManagers = @{@"iPhone" : @"Kate", @"iPad" :
@"Kamal", @"Mobile Web" : @"Bill"};
NSNumber *debeUserLiterales = @YES;
NSNumber *CodigoPostal = @10018;

NSArray *nada = @[
    @"Cadena de texto larga...",
    [unaInstanciadeClase too],
    @"texto texto .. ."
];


NSDictionary *diccionarioConIndice = @{
    @"this.key": @"corresponde a este valor",
```

```
    @"ptra.llave": @"remoteData.payload",
    @"mucho": @"algo",
    @"JSON": @"indices",
    @"otros": @"cosas",
};
```

Uso incorrecto:

```
NSArray *nombres = [NSArray arrayWithObjects:@"Andres", @"Juan",
@"Jose", @"Alex", @"Paul", @"Luis", nil];
NSDictionary *productManagers = [NSDictionary
dictionaryWithObjectsAndKeys: @"Kate", @"iPhone", @"Kamal",
@"iPad", @"Bill", @"Mobile Web", nil];
NSNumber *debeUserLiterales = [NSNumber numeroConBooleano:YES];
NSNumber *ZIP Code = [NSNumber numberWithInteger:10018];
```

Comentarios

- Cuando los comentarios son necesarios, deben ser usados para explicar porque un bloque de código en particular hace algo.
- Cualquier comentario que se haya usado debe mantenerse actualizado o debe ser borrado.
- Los comentarios de bloque deben ser generalmente evitados, ya que el código debe ser los mas auto-documentado posible, con solo la necesidad de comentar algo en pocas líneas.

`init` and dealloc

- Los métodos `dealloc` deben ser colocados en la parte superior de la implementación de la clase, inmediatamente después de las sentencias `@synthesize` y `@dynamic`.
- El método `init` debe ser colocado inmediatamente debajo de los métodos `dealloc` de cualquier clase y debe ser estructurado de la siguiente manera:

```
- (instancetype)init {
    self = [super init]; // llama al constructor designado
    if (self) {
        // Inicialización personalizada
    }
    return self;
}
```

Sistema de Archivos

- La imágenes deben ser nombradas usando *camelCase,* y su nombre deber hacer referencia a su propósito, clase o propiedad que modifica y estado.
- Imágenes que se utilizan para un similar propósito deben ser agrupadas en una carpeta de imágenes.

Ejemplo:

```
BotonParaRefrescar / BotonParaRefrescar@2x
BotonParaRefrescarSeleccionado /
BotonParaRefrescarSeleccionado@2x
```

- La estructura que se mantiene en Xcode, como los grupos creados para alojar clases y archivos, deben ser reflejados en el sistema de archivos, para mantenerlos organizados, claros y congruentes.
- Por convención el nombre base para los archivos .h y .m de una categoría es el nombre de la clase seguido por el nombre de la categoría.

2.7 Interfaces Gráficas

2.7.1 Caja de Texto

Se utiliza para el ingreso de información.

Figura 16. Caja de Texto

2.7.2 Botones/Hipervínculos

Se utiliza para activar una acción o evento.

Figura 17. Botón de propósito general

Figura 18. Hipervínculo

2.7.3 Etiqueta

Se utiliza para presentar información al usuario.

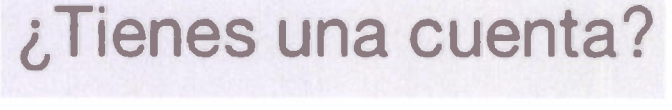


Figura 19. Etiqueta

2.7.4 Tablas

Se utiliza para mostrar una secuencia de datos sobre un mismo tema. Opcionalmente consta de una imagen o botón en los laterales dentro de cada ítem de la lista.

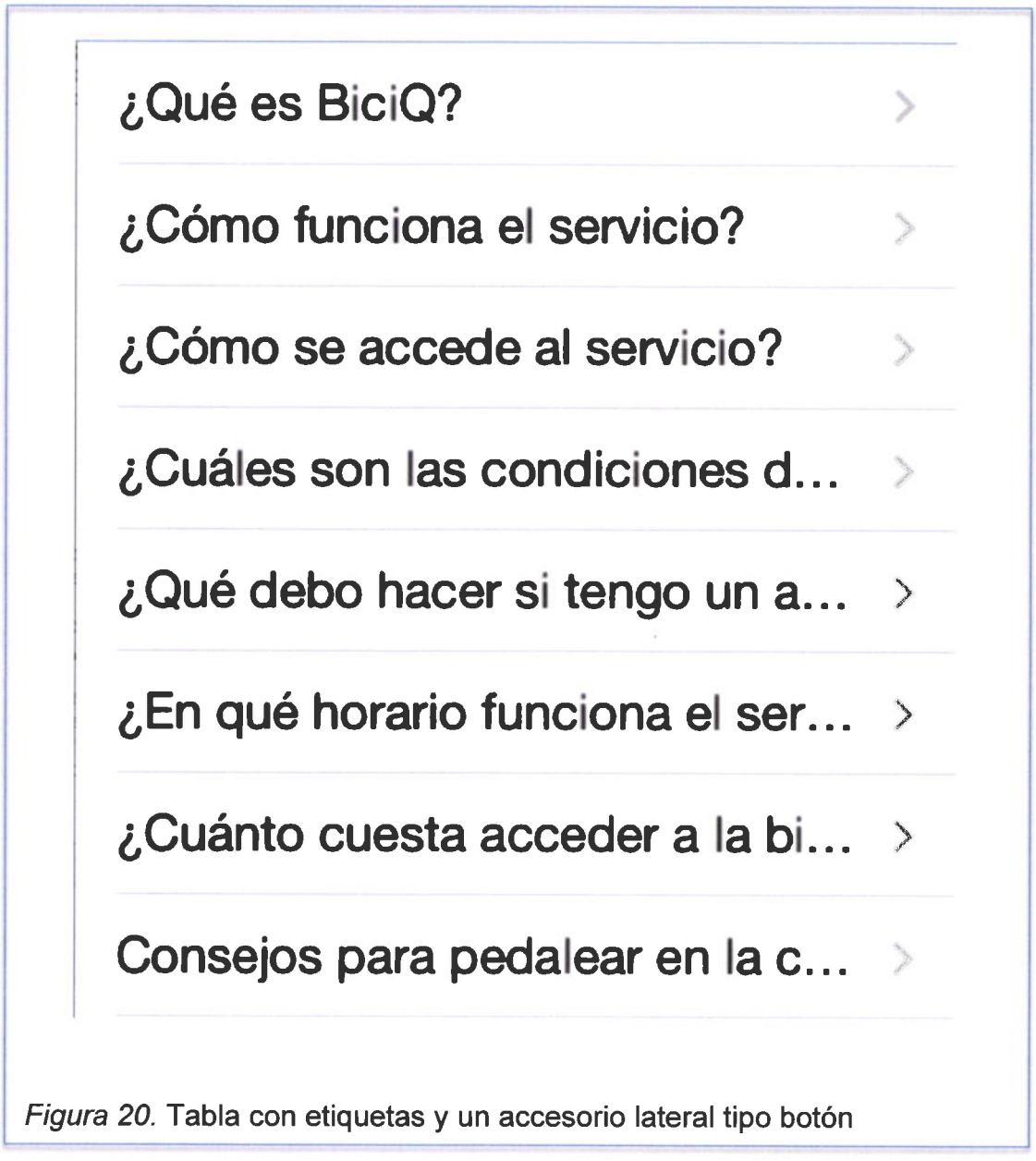


Figura 20. Tabla con etiquetas y un accesorio lateral tipo botón

Figura 21. Tabla con etiquetas y accesorios laterales de tipo imagen y botón.

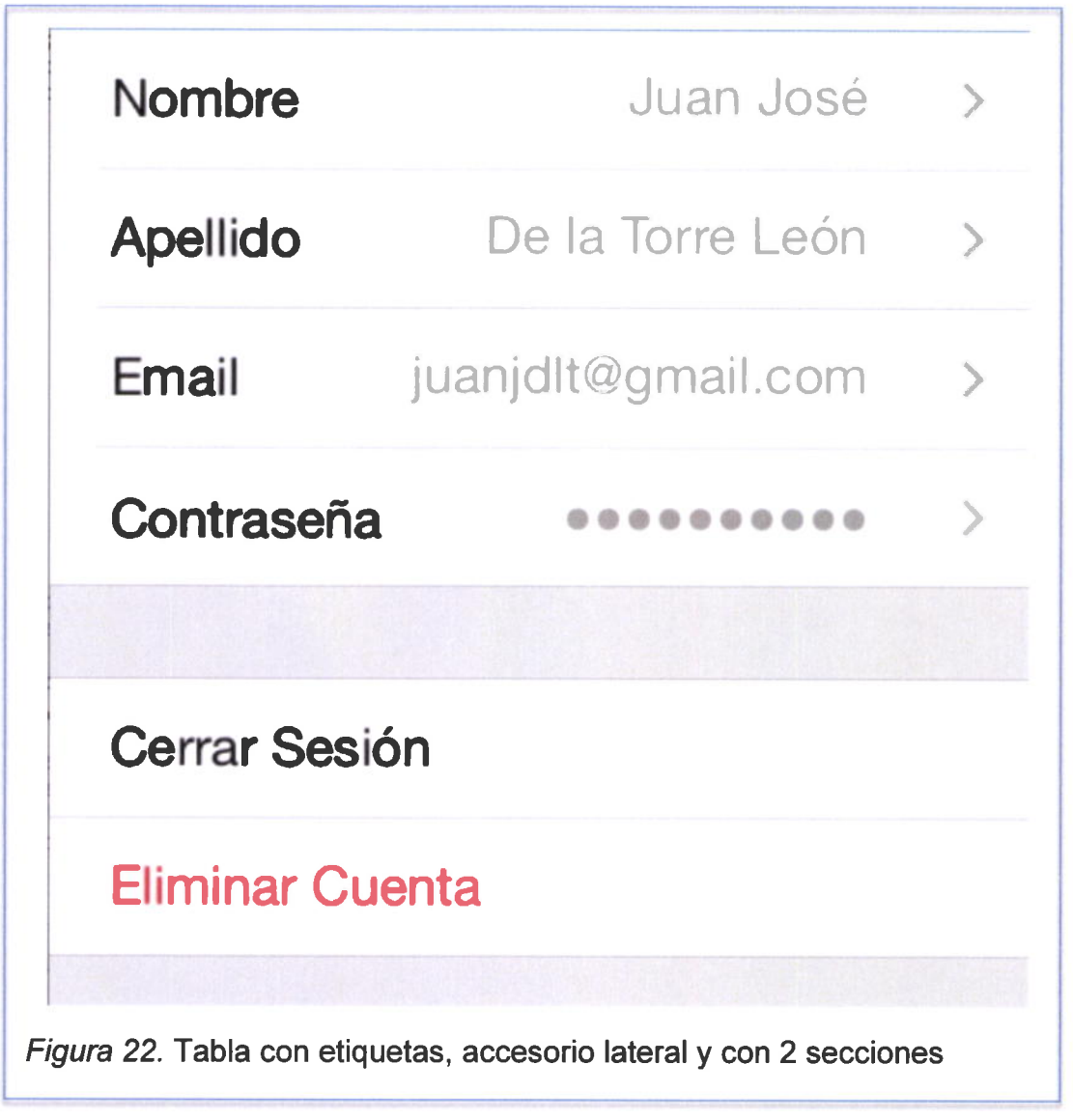


Figura 22. Tabla con etiquetas, accesorio lateral y con 2 secciones

2.7.5 Mapa

Se utiliza para mostrar información geográfica.

Figura 23. Mapa

2.7.6 Mensaje emergente de un ítem del mapa

Se utiliza para presentar mas información o acciones sobre un ítem del mapa.

Figura 24. Mensaje emergente en el mapa

2.7.7 Indicador de una estación de biciQ en el mapa

Se utiliza para mostrar la ubicación en el mapa de una estación biciQ .

Figura 25. Indicador de estación de la biciQ

2.7.8 Barra de Navegación

Se utiliza en todas las interfaces que comprenden una jerarquía de vistas. Opcionalmente muestra 2 botones en sus laterales.

Figura 26. Barra de navegación con botones a los laterales

2.7.9 Barra de Menú

Se utiliza como contendor para cada ítem del menú.

Figura 27. Barra de menú

3. Implementación y Pruebas

3.1. Programación en parejas

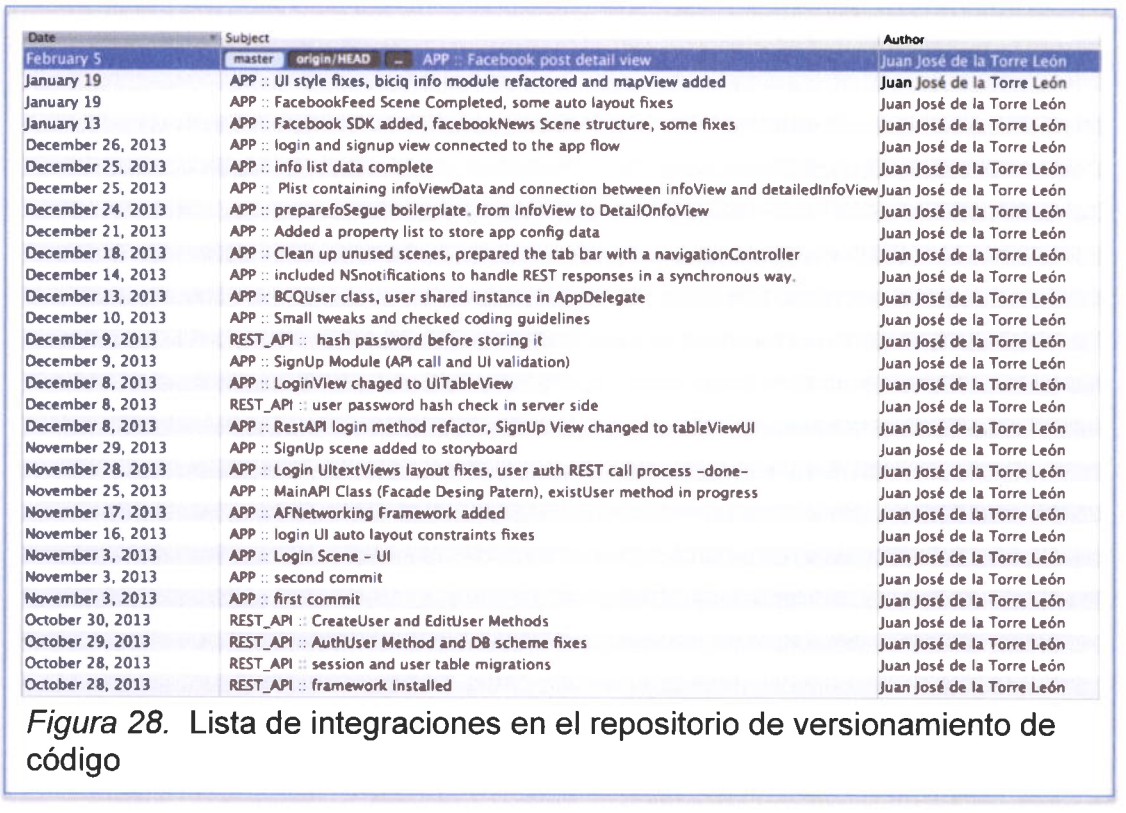
Esta técnica que introduce la metodología de programación extrema no se aplicó, debido a que el desarrollo de la aplicación fue elaborado por una sola persona.

3.2. Adoptar código de propiedad colectiva

Esta técnica que introduce la metodología de programación extrema no se aplicó, debido a que el desarrollo de la aplicación fue elaborado por una sola persona y no por un equipo de desarrolladores. En definitiva se adoptó la propiedad de código de forma individual.

3.3. Integrar continuamente

Esta técnica fue aplicada de manera individual y siguiendo todos los fundamentos establecidos por la metodología. Se usó un repositorio de versionamiento de código local, en el cual el código fue integrado cada vez que un módulo fue terminado, probado y refactorizado.



Date	Subject	Author
February 5	master origin/HEAD APP :: Facebook post detail view	Juan José de la Torre León
January 19	APP :: UI style fixes, biciq info module refactored and mapView added	Juan José de la Torre León
January 19	APP :: FacebookFeed Scene Completed, some auto layout fixes	Juan José de la Torre León
January 13	APP :: Facebook SDK added, facebookNews Scene structure, some fixes	Juan José de la Torre León
December 26, 2013	APP :: login and signup view connected to the app flow	Juan José de la Torre León
December 25, 2013	APP :: info list data complete	Juan José de la Torre León
December 25, 2013	APP :: Plist containing infoViewData and connection between infoView and detailedInfoView	Juan José de la Torre León
December 24, 2013	APP :: preparefoSegue boilerplate. from InfoView to DetailOnfoView	Juan José de la Torre León
December 21, 2013	APP :: Added a property list to store app config data	Juan José de la Torre León
December 18, 2013	APP :: Clean up unused scenes, prepared the tab bar with a navigationController	Juan José de la Torre León
December 14, 2013	APP :: included NSnotifications to handle REST responses in a synchronous way.	Juan José de la Torre León
December 13, 2013	APP :: BCQUser class, user shared instance in AppDelegate	Juan José de la Torre León
December 10, 2013	APP :: Small tweaks and checked coding guidelines	Juan José de la Torre León
December 9, 2013	REST_API :: hash password before storing it	Juan José de la Torre León
December 9, 2013	APP :: SignUp Module (API call and UI validation)	Juan José de la Torre León
December 8, 2013	APP :: LoginVIew chaged to UITableView	Juan José de la Torre León
December 8, 2013	REST_API :: user password hash check in server side	Juan José de la Torre León
December 8, 2013	APP :: RestAPI login method refactor, SignUp View changed to tableViewUI	Juan José de la Torre León
November 29, 2013	APP :: SignUp scene added to storyboard	Juan José de la Torre León
November 28, 2013	APP :: Login UItextViews layout fixes, user auth REST call process -done-	Juan José de la Torre León
November 25, 2013	APP :: MainAPI Class (Facade Desing Patern), existUser method in progress	Juan José de la Torre León
November 17, 2013	APP :: AFNetworking Framework added	Juan José de la Torre León
November 16, 2013	APP :: login UI auto layout constraints fixes	Juan José de la Torre León
November 3, 2013	APP :: Login Scene - UI	Juan José de la Torre León
November 3, 2013	APP :: second commit	Juan José de la Torre León
November 3, 2013	APP :: first commit	Juan José de la Torre León
October 30, 2013	REST_API :: CreateUser and EditUser Methods	Juan José de la Torre León
October 29, 2013	REST_API :: AuthUser Method and DB scheme fixes	Juan José de la Torre León
October 28, 2013	REST_API :: session and user table migrations	Juan José de la Torre León
October 28, 2013	REST_API :: framework installed	Juan José de la Torre León

Figura 28. Lista de integraciones en el repositorio de versionamiento de código

3.4. Refactorizar sin piedad

La refactorización del código se llevó a cabo según el criterio y habilidad del programador, teniendo claro el objetivo de encontrar siempre el diseño más optimo, la simplicidad y flexibilidad del código.

Los puntos más importantes al momento de revisar el código para refactorizar, fueron los siguientes:

- Eliminar la repeticiones de código, ya sea en variables o en estructuras de control repetidas alrededor de varias clases o métodos.
- Dividir funciones muy largas en pequeñas.
- Clarificar los nombres de variables , archivos y métodos.
- Abstraer y generalizar las funcionalidades principales.
- Revisar los patrones de diseño así como la arquitectura de la aplicación.

3.5. Codificación

3.5.1 Inicio de Sesión

```
- (IBAction)doLogin:(id)sender
{

    NSString *emailRegex = @"[-0-9a-zA-Z.+_]+@[-0-9a-zA-Z.+_]+.[a-zA-Z]{2,4}";
    NSRange range = [self.username.text rangeOfString:emailRegex options:NSRegularExpressionSearch];

    if (range.location == NSNotFound) {
        BCQ_alertView(@"Error de Validación", @"Ingrese una dirección de correo electrónico válida");
        return;
    }

    if (self.password.text.length == 0 ) {
        BCQ_alertView(@"Error de Validación", @"Ingrese su contraseña");
        return;
    }

    [BCQUser login:self.username.text withPassword:self.password.text];

}
```

Figura 29. Método de la aplicación para autentificar al usuario.

Este método realiza los siguientes procedimientos:

1. Verifica que el correo electrónico ingresado por el usuario se encuentre correcto.
2. Verifica que la contraseña ingresada por el usuario no se encuentre vacía.
3. Si ambos datos son válidos, se realiza una llamada al método *loginWithPassword* de la clase *BCQUser,* enviando a este la información ingresada por el usuario.

```
- (void)loginUser:(NSString *)username withPassword:(NSString *)password
{

    AppDelegate *appDelegate = [[UIApplication sharedApplication]delegate];

    [[BCQBiciqRESTClient sharedBiciqRESTClient] setUsername:[[appDelegate.appConfig objectForKey:@"APIAuth"] objectForKey:@"username"]
                                                andPassword:[[appDelegate.appConfig objectForKey:@"APIAuth"] objectForKey:@"password"]];

    [[BCQBiciqRESTClient sharedBiciqRESTClient] usernameExist:username withPassword:password onCompletion:^(NSDictionary *response) {
        if (! [response objectForKey:@"error"]) {

            BCQUser *user = appDelegate.currentUser;
            user.id = (NSNumber *)response[@"id"];
            user.username = response[@"email"];
            user.email = response[@"email"];
            user.firstName = response[@"nombres"];
            user.lastName = response[@"apellidos"];

            [[NSNotificationCenter defaultCenter] postNotificationName:@"userLoggedIn" object:self];
        }

    }];
}
```

Figura 30. Método de la aplicación para autentificar al usuario a través del servicio REST

Este método es responsable de llamar a la función del servicio REST para verificar la existencia del usuario y así permitir el ingreso a la aplicación.

```
public function authenticate()
{
    $rules = array(
            'email' => 'required|email|',
            'password' => 'required',
            );

    $validator = Validator::make(Input::only('email', 'password'), $rules);

    if (!$validator->fails()) {
        try {
            $app_user = AppUser::where('username', '=', Input::get('email'))->firstOrFail();

            if (Hash::check(Input::get('password'), $app_user->password))
            {
                return Response::json(array(
                    'id'        => $app_user->id,
                    'nombres'   => $app_user->first_name,
                    'apellidos' => $app_user->last_name,
                    'email'     => $app_user->email,
                    'facebook'  => $app_user->is_facebook_permitted,
                    ), 200
                );
            }

            throw new ModelNotFoundException;

        } catch (ModelNotFoundException $e) {
            return Response::json(array($this->empty_user), 404);
        }
    }

    return Response::json(array($this->empty_user), 400);
}
```

Figura 31. Método REST para autentificar al usuario.

Este método REST permite verificar si existe un usuario y que los datos del mismo se encuentren en un formato correcto.

Figura 32. Interface de inicio de sesión

3.5.2 Registro de Usuario

```
- (IBAction)doSignUp:(id)sender
{

    NSString *emailRegex = @"[-0-9a-zA-Z.+_]+@[-0-9a-zA-Z.+_]+.[a-zA-Z]{2,4}";
    NSRange range = [self.email.text rangeOfString:emailRegex options:NSRegularExpressionSearch];

    if (self.name.text.length == 0 || self.lastName.text.length == 0 || self.password.text.length == 0 ) {
        BCQ_alertView(@"Error de Validación", @"Todos los campos son requeridos");
        return;
    }

    if (range.location == NSNotFound) {
        BCQ_alertView(@"Error de Validación", @"Ingrese una dirección de correo electrónico válida");
        return;
    }

    NSDictionary *data =
                        @{
                            @"first_name": self.name.text,
                            @"last_name": self.lastName.text,
                            @"email": self.email.text,
                            @"password": self.password.text,
                            @"is_facebook_permitted": self.isFacebookPermited.isOn ? @"1" : @"0"
                        };

    [BCQUser create:data];
}
```

Figura 33. Método de la aplicación que permite registrar un usuario.

Este método realiza los siguientes procedimientos:

1. Verifica que el nombre, apellido y contraseña ingresados por el usuario no se encuentren vacíos.
2. Verifica que el correo electrónico ingresado por el usuario se encuentre correcto.
3. Si ambos datos son válidos, se realiza una llamada al método *create* de la clase *BCQUser,* enviando a este la información ingresada por el usuario.

```
- (void)registerNewUser:(NSDictionary *)data;
{
    AppDelegate *appDelegate = [[UIApplication sharedApplication]delegate];

    [[BCQbicloRESTClient sharedBicloRESTClient] setUsername:[[appDelegate.appConfig objectForKey:@"APIAuth"] objectForKey:@"username"]
                                                andPassword:[[appDelegate.appConfig objectForKey:@"APIAuth"] objectForKey:@"password"]];

    [[BCQbicloRESTClient sharedBicloRESTClient] registerNewUser:data onCompletion:^(NSDictionary *response) {
        if (! [response objectForKey:@"error"]) {

            BCQUser *user = appDelegate.currentUser;

            user.id = (NSNumber *)response[@"id"];
            user.username = response[@"email"];
            user.email = response[@"email"];
            user.firstName = response[@"nombres"];
            user.lastName = response[@"apellidos"];

            [[NSNotificationCenter defaultCenter] postNotificationName:@"userRegistered" object:self];
        }
    }];
}
```

Figura 34. Método de la aplicación para registrar al usuario a través del servicio REST.

Este método es responsable de llamar a la función del servicio REST para crear una nueva cuenta de usuario.

```
public function store()
{
    $rules = array(
                'nombres'   => 'required',
                'apellidos' => 'required',
                'email'     => 'required|email|unique:app_users',
                'password'  => 'required',
                'facebook'  => 'integer|max:1'
            );

    $validator = Validator::make(Input::only('nombres', 'apellidos', 'email', 'password', 'facebook'), $rules);

    if (!$validator->fails()) {

        try {
            $app_user = AppUser::create(array(
                            'username'  => Input::get('email'),
                            'first_name'=> Input::get('nombres'),
                            'last_name' => Input::get('apellidos'),
                            'email'     => Input::get('email'),
                            'password'  => Hash::make(Input::get('password')),
                            'is_facebook_permitted' => Input::get('facebook'),
                            )
                        );

            return Response::json(array(
                'id'        => $app_user->id,
                'nombres'   => $app_user->first_name,
                'apellidos' => $app_user->last_name,
                'email'     => $app_user->email,
                'facebook'  => $app_user->is_facebook_permitted,
                ), 201
            );

        } catch (Exception $e) {
            return Response::json(array($this->empty_user), 500);
        }

    } else {
        return Response::json(array($this->empty_user), 400);
    }
}
```

Figura 35. Método REST para crear una cuenta de usuario.

Este método REST permite crear una nueva cuenta de usuario y verificar que los datos del mismo se encuentren en un formato correcto.

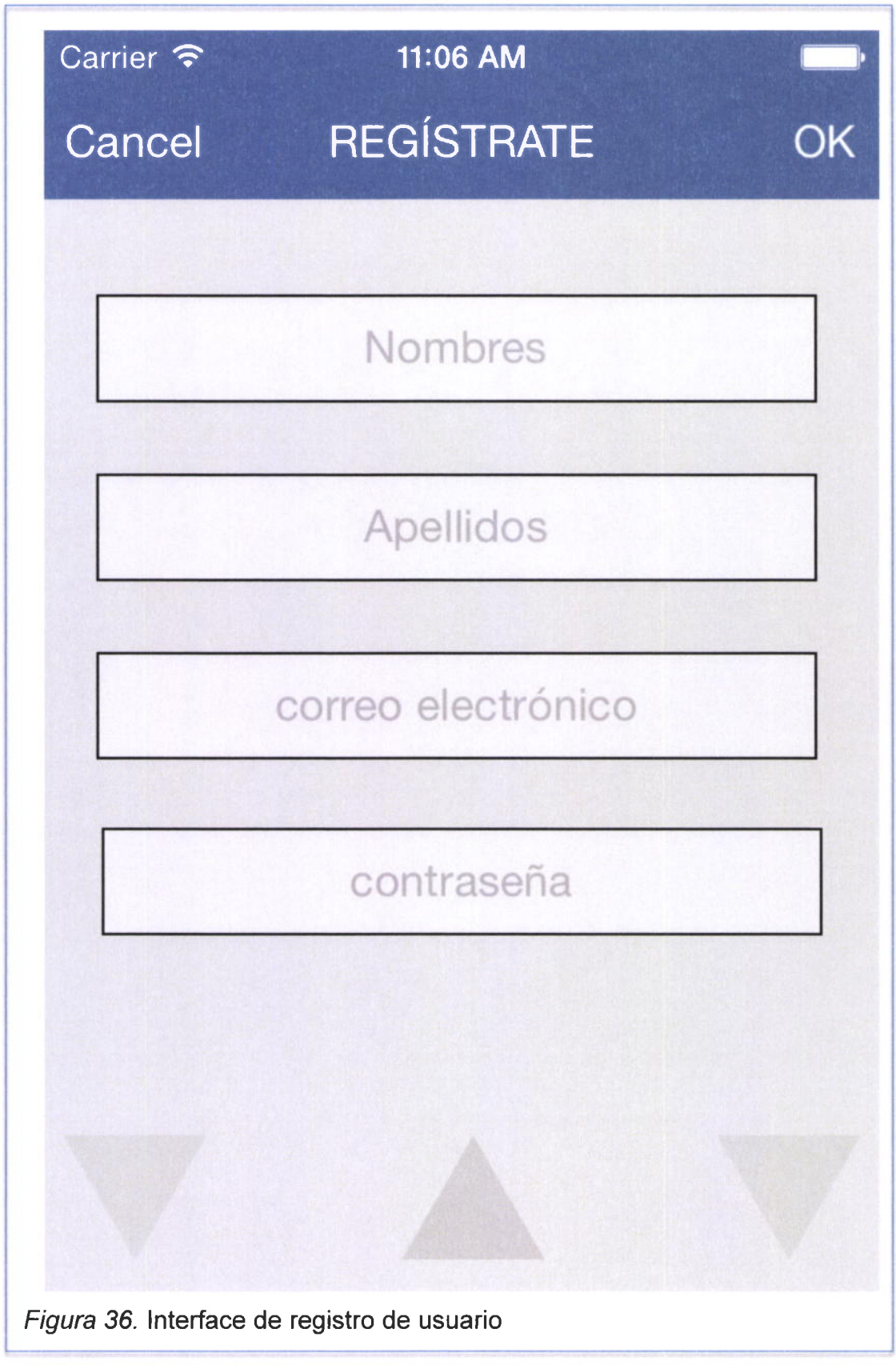


Figura 36. Interface de registro de usuario

3.5.3 Información biciQ

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.biciqData = [BCQBiciqInfo getInfo];
}

#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [self.biciqData count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"ListPrototypeCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];

    NSDictionary *cellData = [self.biciqData objectAtIndex:indexPath.row];
    cell.textLabel.text = cellData[@"title"];

    return cell;
}
```

Figura 37. Métodos de la aplicación para mostrar la información de la biciQ

Estos métodos son responsables de obtener los datos informativos sobre la biciQ y presentarlos en la interface de usuario.

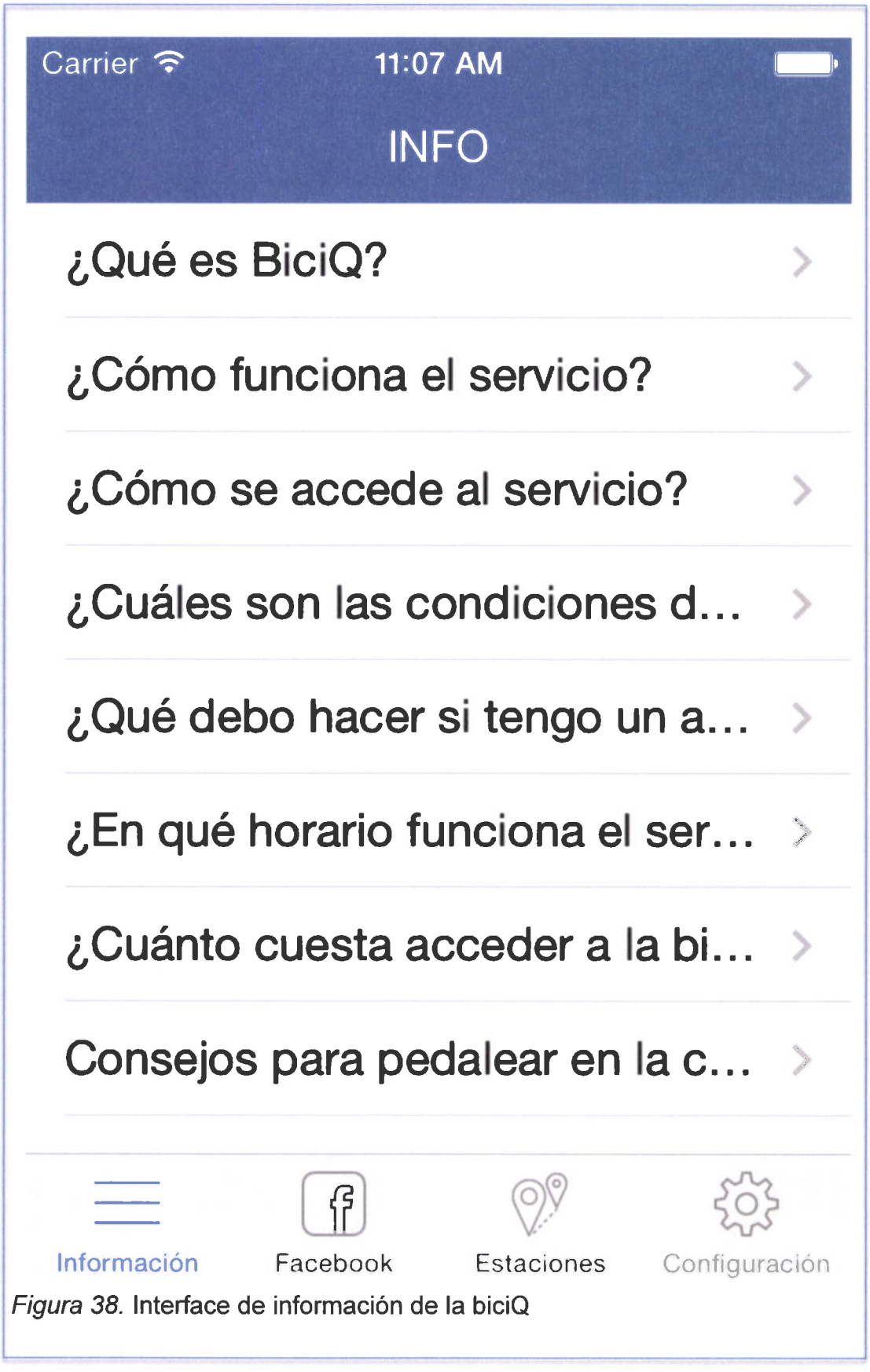


Figura 38. Interface de información de la biciQ

3.5.4 Entradas de Facebook de biciQ

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    UIActivityIndicatorView *activityIndicatorView = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleGray];
    self.navigationItem.leftBarButtonItem = [[UIBarButtonItem alloc] initWithCustomView:activityIndicatorView];

    self.facebookPostsTableView.dataSource = self;
    self.facebookPostsTableView.delegate = self;

    self.facebookPostItems = [[NSArray alloc] init];

    [self reload:nil];

}

- (IBAction)reload:(id)sender
{
    UIActivityIndicatorView *activityIndicatorView = (UIActivityIndicatorView *)self.navigationItem.leftBarButtonItem.customView;
    [activityIndicatorView startAnimating];

    self.navigationItem.rightBarButtonItem.enabled = NO;

    [BCQFacebookPost publicTimelinePostsWithLimit:[NSNumber numberWithInt:15] andBlock:^(NSArray *posts, NSError *error) {
        if (!error) {
            self.facebookPostItems = posts;
            [activityIndicatorView stopAnimating];
            [self.facebookPostsTableView reloadData];
        } else {
            BCQ_alertView(@"Error de Comunicación", @"No se puede conectar con facebook en este momento, intente más tarde.");
            [activityIndicatorView stopAnimating];
        }

        self.navigationItem.rightBarButtonItem.enabled = YES;
    }];
}
```

Figura 39. Métodos de la aplicación que muestran las entradas de Facebook en la interface

Estos métodos son responsables de obtener las últimas entradas de la página oficial en Facebook de la biciQ y presentarlos en la interface de usuario.

```
- (void)getPostsFrom:(NSString *)entity limitedBy:(NSNumber *)limit onCompletion:(JSONResponseBlock)completionBlock
{
    if (! self.untilPostID) {
        self.untilPostID = @"";
    }
    NSString *graphPath = [NSString stringWithFormat:@"%@/posts?fields=id,message,picture,actions&limit=%@&until=%@", entity, limit, self.untilPostID];

    if (FBSession.activeSession.isOpen) {
        // login is integrated with the send button -- so if open, we send
        [FBRequestConnection startWithGraphPath:graphPath
                                     parameters:nil
                                     HTTPMethod:@"GET"
                              completionHandler:^(FBRequestConnection *connection, id result, NSError *error) {
                                  if (!error) {
                                      // Sucess! Include your code to handle the results here
                                      self.untilPostID = [[result objectForKey:@"paging"] objectForKey:@"next"];
                                      completionBlock(@{@"feeds": [result objectForKey:@"data"]});
                                  } else {
                                      // An error occurred, we need to handle the error
                                      // See: https://developers.facebook.com/docs/ios/errors
                                      completionBlock(@{@"error": error});
                                  }
                              }];
    } else {
        [FBSession openActiveSessionWithReadPermissions: @[@"basic_info", @"public_profile", @"user_friends"]
                                           allowLoginUI:YES
                                      completionHandler:^(FBSession *session, FBSessionState status, NSError *error) {
                                          if (error) {

                                              completionBlock(@{@"error": error});

                                          } else if (FB_ISSESSIONOPENWITHSTATE(status)) {
                                              // send our requests if we successfully logged in
                                              [FBRequestConnection startWithGraphPath:graphPath
                                                                           parameters:nil
                                                                           HTTPMethod:@"GET"
                                                                    completionHandler:^(FBRequestConnection *connection, id result, NSError *error) {
                                                                        if (!error) {
                                                                            // Sucess! Include your code to handle the results here
                                                                            self.untilPostID = [[result objectForKey:@"paging"] objectForKey:@"next"];
                                                                            completionBlock(@{@"feeds": [result objectForKey:@"data"]});
                                                                        } else {
                                                                            // An error occurred, we need to handle the error
                                                                            // See: https://developers.facebook.com/docs/ios/errors
                                                                            completionBlock(@{@"error": error});
                                                                        }
                                                                    }
                                                                    ];
                                          }
                                      }
        ];
    }

}
```

Figura 40. Método de la aplicación que se conecta con Facebook para obtener las ultimas entradas

Este método es responsable de obtener las últimas entradas de Facebook a través de una llamada a un recurso REST utilizando el SDK de esta red social para iOS.

Dicho método realiza los siguientes procedimientos:

1. Define la URL del recurso REST, estructurada de la siguiente manera:
   - **<nombre_página>/posts:** Ruta base para obtener las entradas de una página de Facebook.
   - **?fields=id,message,picture,actions:** Se aplica un filtro para obtener solo ciertos campos en la respuesta.
   - **&limit=<número>&until=<número>:** Filtros aplicados para habilitar paginación en la respuesta.

2. Verifica si el usuario a realizado la autorización para acceder a Facebook desde la aplicación móvil. Si no esta autorizado, se presenta una interfaz de autorización para que el usuario permita la conexión entre Facebook y la aplicación móvil.

3. Finalmente, se realiza la llamada al recurso REST del *API* de Facebook a través de una petición *HTTP* de tipo *GET.* Los resultados son devueltos por este método en forma de una estructura de tipo diccionario.

Figura 41. Interface entradas de Facebook de la biciQ

3.5.5 Mapa de Estaciones y Ruta

```
-(NSArray *)biciqStations
{
    return [BCQStation getAllStations];
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self drawBikeStations];

    if ([CLLocationManager locationServicesEnabled]) {
        [self startStandardUpdates];
    } else {
        NSLog(@"LocationManager auth status %u", [CLLocationManager authorizationStatus]);
        BCQ_alertView(@"Error", @"Si desea saber su ubicación actual, cambie su configuración de privacidad");
    }

}

#pragma mark - Methods

- (void)drawBikeStations
{
    [self.mapView removeAnnotations:self.mapView.annotations];
    [self.mapView addAnnotations:self.biciqStations];
    [self.mapView showAnnotations:self.biciqStations animated:YES];
}

- (void)updateUI
{
    self.speedLabel.text = [NSString stringWithFormat: @"%.2f", self.currentSpeed];
}

- (void)setCurrentSpeed:(double)currentSpeed
{
    _currentSpeed = (currentSpeed * 3600.00) / (1000.00);
}

- (void)startStandardUpdates
{

    if (! _locationManager) {
        _locationManager = [[CLLocationManager alloc] init];
    }

    self.locationManager.delegate = self;
    self.locationManager.desiredAccuracy = kCLLocationAccuracyBest;

    self.locationManager.distanceFilter = 10;

    [self.locationManager startUpdatingLocation];
}

- (IBAction)showCurrentLocation {
    [self.mapView setCenterCoordinate:self.mapView.userLocation.location.coordinate animated:YES];
}



- (MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id <MKAnnotation>)annotation {

    static NSString *identifier = @"biciqStation";

    if ([annotation isKindOfClass:[MKUserLocation class]]) {
        return nil;
    }

    MKAnnotationView *annotationView = [mapView dequeueReusableAnnotationViewWithIdentifier:identifier];
    if (! annotationView) {
        annotationView = [[MKPinAnnotationView alloc] initWithAnnotation:annotation reuseIdentifier:identifier];
        annotationView.canShowCallout = YES;
        annotationView.image=[UIImage imageNamed:@"biciqStationPin"];
    }

    annotationView.annotation = annotation;

    return annotationView;
}

#pragma mark - CLLocationManagerDelegate methos

- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray *)locations {

    CLLocation* location = [locations lastObject];

    self.currentSpeed = location.speed;
    self.userCurrentLatitude = location.coordinate.latitude;
    self.userCurrentLongitude = location.coordinate.longitude;

    NSLog(@"latitude %+.6f, longitude %+.6f\n, speed:%.2f\n",
              location.coordinate.latitude,
              location.coordinate.longitude,
          location.speed);
    [self updateUI];
}
```

Figura 42. Métodos para presentar el mapa, las estaciones de la biciQ y los datos de la ruta

Estos métodos son responsables de iniciar el módulo del mapa, obtener y presentar la información sobre cada estación de la biciQ y usar la ubicación y velocidad actual del usuario para mostrar en la interface y generar la ruta.

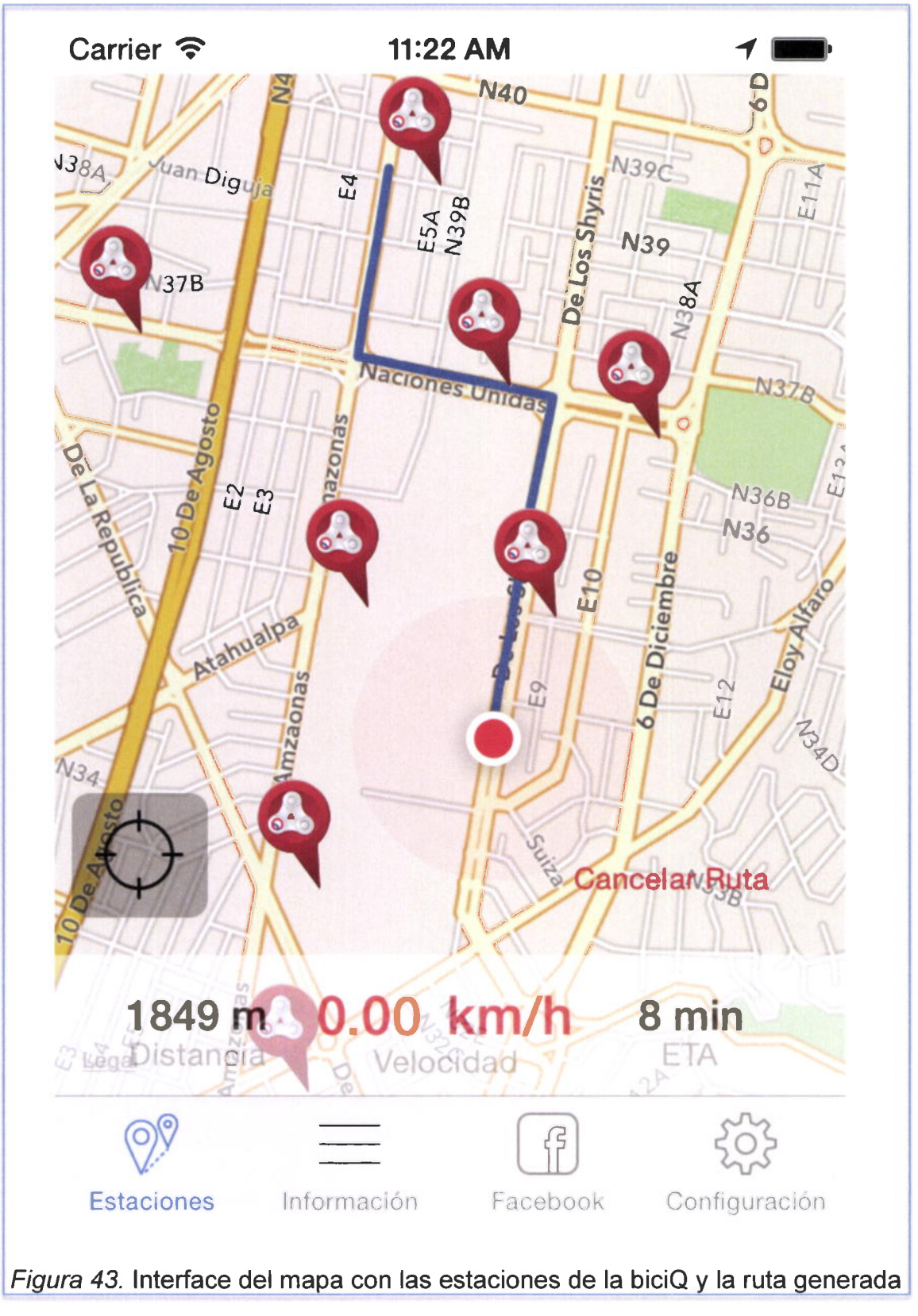


Figura 43. Interface del mapa con las estaciones de la biciQ y la ruta generada

3.6. Adoptar un desarrollo basado en pruebas

3.6.1. Pruebas Unitarias

Para la creación de pruebas unitarias se eligió la libraría nativa de Xcode llamada XCTest.

Esta librería permite una integración total con el código desarrollado en Xcode facilitando la ejecución de las pruebas automatizadas. Otra gran ventaja es que las pruebas se las realiza desde el mismo entorno de desarrollo de código. En si permitiendo la creación de avanzadas suites de pruebas de una manera muy ágil.

Las pruebas realizadas se dividieron según las interfaces o el flujo de datos por interfaz gráfica.

Cada caso de prueba contiene uno o más métodos que se traducen a lo que son pruebas unitarias y cada caso de prueba se puede traducir a una prueba de aceptación en código.

La estructura general de un caso de prueba usando la librería XCTest es la siguiente:

- Todas los casos de pruebas deben heredar de la clase ***XCTestCase***.
- ***- (void)setUp*** es un método heredado de la clase base mencionada anteriormente. El bloque de código que pertenezca a este método será invocado antes de cada prueba unitaria.
- ***- (void)tearDown*** es un método heredado de la clase padre mencionada anteriormente. El bloque de código que pertenezca a este método será invocado después de cada prueba unitaria.
- El nombre de cada método de una prueba unitaria debe empezar con la palabra ***test***, ejemplo: testIngresoDeUsuario.

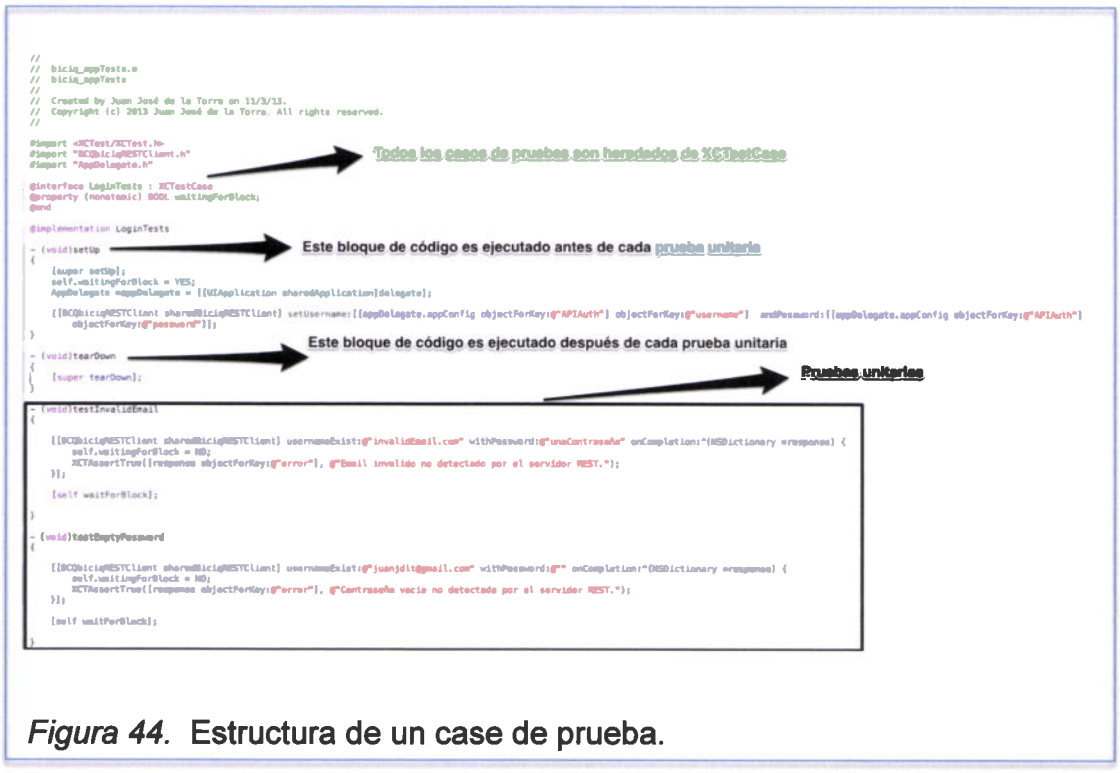


Figura 44. Estructura de un case de prueba.

A continuación se presentan los casos de pruebas realizados, con sus respectivas pruebas unitarias:

```
@interface LoginTests : XCTestCase
@property (nonatomic) BOOL waitingForBlock;
@end

@implementation LoginTests

- (void)setUp
{
    [super setUp];
    self.waitingForBlock = YES;
    AppDelegate *appDelegate = [[UIApplication sharedApplication]delegate];

    [[BCQbiciqRESTClient sharedBiciqRESTClient] setUsername:[[appDelegate.appConfig objectForKey:@"APIAuth"] objectForKey:@"username"]
        andPassword:[[appDelegate.appConfig objectForKey:@"APIAuth"] objectForKey:@"password"]];
}

- (void)tearDown
{
    [super tearDown];
}

- (void)testInvalidEmail
{

    [[BCQbiciqRESTClient sharedBiciqRESTClient] usernameExist:@"invalidEmail.com" withPassword:@"unaContraseña" onCompletion:^(NSDictionary
        *response) {
        self.waitingForBlock = NO;
        XCTAssertTrue([response objectForKey:@"error"], @"Email invalido no detectado por el servidor REST.");
    }];

    [self waitForBlock];

}

- (void)testEmptyPassword
{

    [[BCQbiciqRESTClient sharedBiciqRESTClient] usernameExist:@"juanjdlt@gmail.com" withPassword:@"" onCompletion:^(NSDictionary *response)
        {
        self.waitingForBlock = NO;
        XCTAssertTrue([response objectForKey:@"error"], @"Contraseña vacia no detectada por el servidor REST.");
    }];

    [self waitForBlock];

}

- (void)testNotRegisteredUser
{

    [[BCQbiciqRESTClient sharedBiciqRESTClient] usernameExist:@"not_registered_user@email.com" withPassword:@"123456789" onCompletion:^
        (NSDictionary *response) {
        self.waitingForBlock = NO;
        XCTAssertTrue([response objectForKey:@"error"], @"No se detectó un usuario desconocido.");
    }];

    [self waitForBlock];

}

- (void)testLogin
{

    [[BCQbiciqRESTClient sharedBiciqRESTClient] usernameExist:@"juanjdlt@gmail.com" withPassword:@"12345678" onCompletion:^(NSDictionary *
        response) {
        self.waitingForBlock = NO;
        XCTAssertFalse([response objectForKey:@"error"], @"No se pudo autentificar un usuario.");
    }];

    [self waitForBlock];

}
```

Figura 45. Caso de prueba para el ingreso del usuario.

```
@interface SignUpTests : XCTestCase
@property (nonatomic) BOOL waitingForBlock;
@end

@implementation SignUpTests

- (void)setUp
{
    [super setUp];
    self.waitingForBlock = YES;
    AppDelegate *appDelegate = [[UIApplication sharedApplication]delegate];

    [[BCQbiciqRESTClient sharedBiciqRESTClient] setUsername:[[appDelegate.appConfig objectForKey:@"APIAuth"] objectForKey:@"username"]
        andPassword:[[appDelegate.appConfig objectForKey:@"APIAuth"] objectForKey:@"password"]];
}

- (void)testInvalidEmail
{
    NSDictionary *data =
    @{
      @"first_name": @"Primer Nombre",
      @"last_name": @"Apelliddo",
      @"email": @"emailInvalida.comasd3w4",
      @"password": @"contraseña",
      @"is_facebook_permitted": @"1"
      };

    [[BCQbiciqRESTClient sharedBiciqRESTClient] registerNewUser:data onCompletion:^(NSDictionary *response) {
        self.waitingForBlock = NO;

        XCTAssertTrue([response objectForKey:@"error"], @"No se detectó un email incorrecto.");
    }];
    [self waitForBlock];
}

- (void)testExistingEmail
{
    NSDictionary *data =
    @{
      @"first_name": @"Primer Nombre",
      @"last_name": @"Apelliddo",
      @"email": @"rdlt60@gmail.com",
      @"password": @"12345678",
      @"is_facebook_permitted": @"1"
      };

    [[BCQbiciqRESTClient sharedBiciqRESTClient] registerNewUser:data onCompletion:^(NSDictionary *response) {
        self.waitingForBlock = NO;
        XCTAssertTrue([response objectForKey:@"error"], @"No se detectó que un usuario ya existe con ese email.");
    }];
    [self waitForBlock];
}

- (void)testIncompleteData
{
    NSDictionary *data =
    @{
      @"first_name": @"Primer Nombre",
      @"last_name": @"",
      @"email": @"unemailbueno@hotmail.com",
      @"password": @"",
- (void)testSignUp
{
    NSDictionary *data =
    @{
      @"first_name": @"NombreTest",
      @"last_name": @"ApellidoTest",
      @"email": [[NSString alloc] initWithFormat:@"test_%u@biciq.com", arc4random()],
      @"password": @"12345678",
      @"is_facebook_permitted": @"1"
      };

    [[BCQbiciqRESTClient sharedBiciqRESTClient] registerNewUser:data onCompletion:^(NSDictionary *response) {
        self.waitingForBlock = NO;
        XCTAssertFalse([response objectForKey:@"error"], @"Usuario no creado.");
    }];
    [self waitForBlock];
    //TODO:: Delete test user
}


- (void)waitForBlock
{
    while(self.waitingForBlock) {
        [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode
                                 beforeDate:[NSDate dateWithTimeIntervalSinceNow:0.1]];
    }
}

@end
```

Figura 46. Caso de prueba para el registro de un usuario.

```
#import <XCTest/XCTest.h>
#import "BCQBiciqInfo.h"

@interface BiciqInfoTests : XCTestCase

@end

@implementation BiciqInfoTests

- (void)testBiciqInfo
{
    XCTAssertTrue((([[BCQBiciqInfo getInfo] count] > 0), @"No existen datos informativos sobre la biciQ");
}

@end
```

Figura 47. Caso de prueba para la información sobre la biciQ.

```
#import <XCTest/XCTest.h>
#import "BCQFacebookAPIClient.h"


@interface FacebookFeedTests : XCTestCase
@property (nonatomic) BOOL waitingForBlock;
@end

@implementation FacebookFeedTests

- (void)setUp
{
    [super setUp];
    self.waitingForBlock = YES;
}

- (void)testPullBiciqFacebookPosts
{
    [[BCQFacebookAPIClient sharedFacebookAPIClient] getPostsFrom:@"biciq" limitedBy:[[NSNumber alloc] initWithInt:10] onCompletion:^
        (NSDictionary *response) {
        self.waitingForBlock = NO;
        XCTAssertFalse(([response objectForKey:@"error"]), @"No se pudo obtener las noticias de facebook. Error: %@", response);
    }];

    [self waitForBlock];
}

- (void)waitForBlock
{
    while(self.waitingForBlock) {
        [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode
                                 beforeDate:[NSDate dateWithTimeIntervalSinceNow:0.1]];
    }
}

@end
```

Figura 48. Caso de prueba para las entradas de Facebook.

```
#import <XCTest/XCTest.h>
#import "BCQStation.h"

@interface MapsAndStationsTests : XCTestCase

@end

@implementation MapsAndStationsTests

- (void)testAllStations
{
    XCTAssertTrue((([[BCQStation getAllStations] count] == 25), @"No están completas las estaciones de la biciQ.");
}

@end
```

Figura 49. Caso de prueba para las estaciones que se muestran en el mapa.

El resultado fue exitoso al ejecutar todos los casos de pruebas con sus respectivas pruebas unitarias.

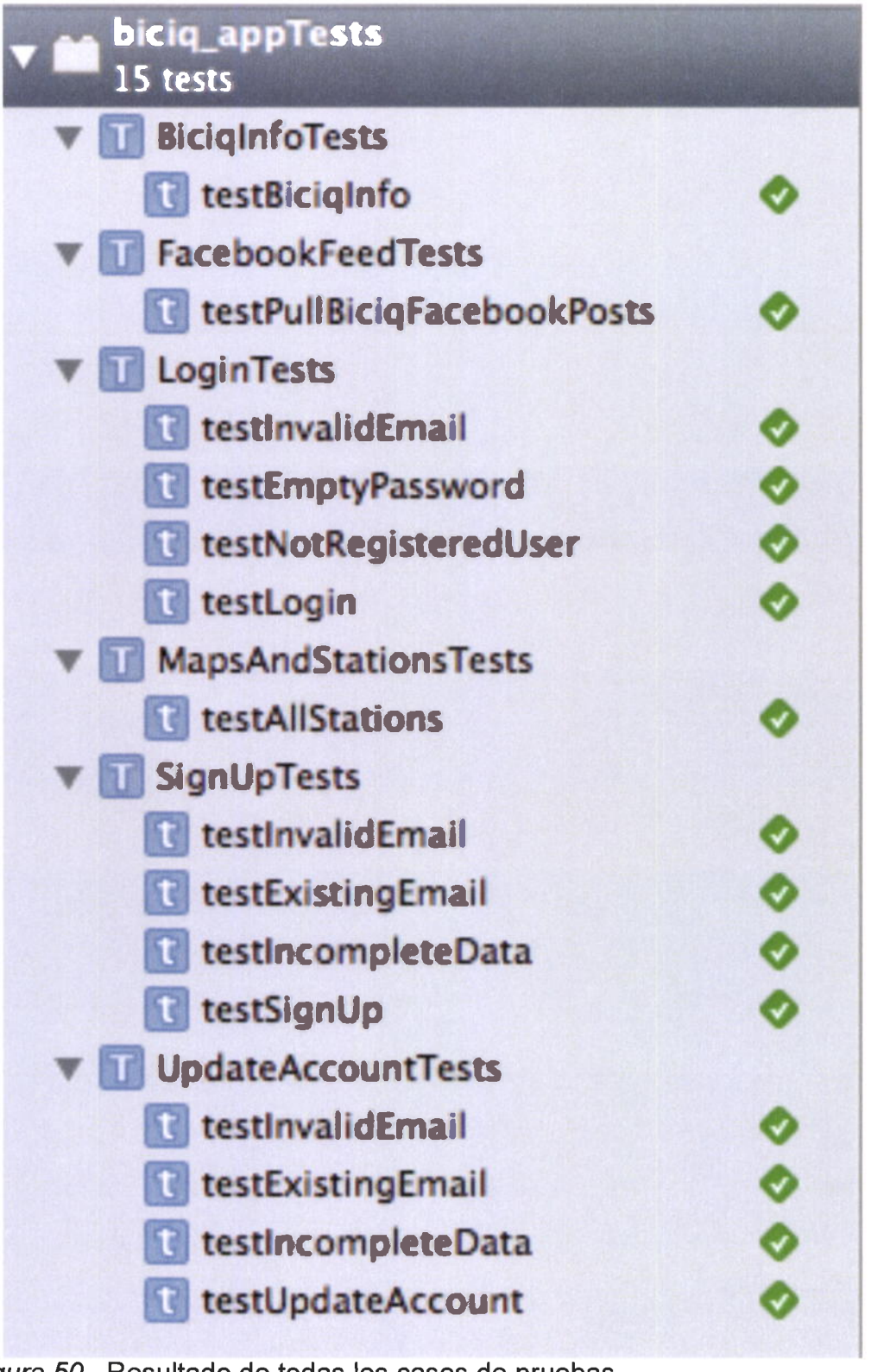


Figura 50. Resultado de todas los casos de pruebas.

3.6.2. Pruebas de Aceptación

Las pruebas de aceptación fueron realizadas a varios usuarios desde un dispositivo iPhone 4 con iOS7 y conexión a internet mediante plan de datos.

Estas fueron las pruebas realizadas según las historias de usuario definidas al momento de iniciar el proyecto:

Tabla 24. Prueba de Aceptación ID1

APLICACIÓN MOVIL COMPATIBLE CON IPHONE		
ID:	1	
ID Historia de usuario:	1	
Ambientación:	Condiciones normales, realizadas en el exterior en un día no lluvioso.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Abrir la aplicación	Se muestre la imagen de bienvenida y se presente correctamente la aplicación	OK
Pre-condiciones:	-	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	-	
EJECUTORES		
Nombre	**Apellido**	**Firma**
María Paola	De la Torre León	

Tabla 25. Prueba de Aceptación ID2

ACCESO A LA APLICACIÓN MOVIL		
ID:	2	
ID Historia de usuario:	2	
Ambientación:	Condiciones normales, realizadas en el exterior en un día no lluvioso.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Ingresar a la aplicación con las credenciales de un usuario previamente registrado	Validación de las credenciales e ingreso a la aplicación.	OK
Pre-condiciones:	El usuario que ejecuta la prueba ya dispone de un nombre de usuario y contraseña.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	El ejecutor de la prueba ingresó correctamente a la aplicación. También pudo observar los mensajes de alerta cuando ingreso incorrectamente las credenciales.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
María Paola	De la Torre León	

Tabla 26. Prueba de Aceptación ID3

CREACIÓN DE UNA CUENTA DE USUARIO		
ID:	3	
ID Historia de usuario:	3	
Ambientación:	Condiciones normales, realizadas en el exterior en un día no lluvioso.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Digitar los datos para crear una nueva cuenta de usuario	Validar los datos ingresados e ingresar a la aplicación.	OK
Pre-condiciones:	El usuario deberá ingresar a la interface de crear una cuenta.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	El ejecutor de la prueba ingreso correctamente a la aplicación tras haber ingresado sus datos. También se pudo observar las alertas al momento que se ingreso datos inválidos.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
Ramiro	De la Torre	

Tabla 27. Prueba de Aceptación ID4

VISUALIZAR LA INFORMACIÓN DEL SERVICIO BICIQ		
ID:	4	
ID Historia de usuario:	4	
Ambientación:	Condiciones normales, realizadas en el exterior en un día no lluvioso.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Seleccionar el menú de información.	Visualizar el titulo de cada ítem de información en la interface de tipo lista.	OK
Presionar sobre un ítem de la lista.	Visualizar una interface con una imagen y texto sobre el ítem recién presionado.	OK
Pre-condiciones:	El usuario debe ingresar a la aplicación con sus credenciales.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	El ejecutor visualizo varios ítems informativos.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
Ramiro	De la Torre	

Tabla 28. Prueba de Aceptación ID5

VISUALIZAR LAS ÚLTIMAS ENTRADAS DE FACEBOOK		
ID:	5	
ID Historia de usuario:	5	
Ambientación:	Condiciones normales, realizadas en el exterior en un día no lluvioso.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Seleccionar el menú de Facebook.	Visualizar el titulo con una imagen por cada entrada o ítem de la lista.	OK
Presionar sobre un ítem de la lista.	Visualizar una interface con una imagen y texto sobre la entrada de Facebook recién presionada.	OK
Pre-condiciones:	El usuario debe ingresar a la aplicación con sus credenciales y debe autorizar a Facebook para conectarse con la aplicación.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	El ejecutor visualizo varios ítems informativos y en la interface de la lista presionó el botón para obtener más entradas de Facebook, con un resultado exitoso.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
Ramiro	De la Torre	

Tabla 29. Prueba de Aceptación ID6

MOSTRAR LAS ESTACIONES DE LA BICIQ EN UN MAPA		
ID:	6	
ID Historia de usuario:	6	
Ambientación:	Condiciones normales, realizadas en el exterior en un día no lluvioso.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Seleccionar el menú del mapa.	Visualizar las estaciones en el mapa.	OK
Presionar sobre el icono de una estación.	Visualizar en una ventana emergente el nombre y la dirección de la estación.	OK
Pre-condiciones:	El usuario debe ingresar a la aplicación con sus credenciales.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	El ejecutor visualizo varios iconos en el mapa y presionó sobre algunos para mirar el nombre y dirección de las estaciones.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
Ramiro	De la Torre	

Tabla 30. Prueba de Aceptación ID7

CREAR UN RUTA EN EL MAPA		
ID:	7	
ID Historia de usuario:	7	
Ambientación:	Condiciones normales, realizadas en el exterior en un día no lluvioso y estando en movimiento en una bicicleta.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Elegir la estación de inicio de la ruta y la de llegada	Visualizar la ruta creada en el mapa y la actualización de los datos informativos sobre el progreso de la ruta.	OK
Pre-condiciones:	El usuario debe ingresar a la aplicación con sus credenciales y seleccionar el menú del mapa.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	El ejecutor realizó la prueba con éxito al crear dos rutas, una entre estaciones de la biciQ y otro entre la ubicación actual del ejecutor y una estación. También se percató de los datos informativos que se actualizan mientras recorre la ruta.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
Ramiro	De la Torre	

Tabla 31. Prueba de Aceptación ID8

COMPARTIR EN FACEBOK LA RUTA		
ID:	8	
ID Historia de usuario:	8	
Ambientación:	Condiciones normales, realizadas en el exterior en un día no lluvioso y estando en movimiento en una bicicleta.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Presionar el botón de compartir ruta. Opcionalmente editar el mensaje a ser compartido en Facebook.	Visualizar una nueva entrada en la cuenta de Facebook del ejecutor.	OK
Pre-condiciones:	El usuario debe ingresar a la aplicación con sus credenciales, seleccionar el menú del mapa y crear una ruta.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	El ejecutor logró compartir los datos de su ruta o recorrido, editando el mensaje personalizado de la entrada de Facebook.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
Gloria	León	

Tabla 32. Prueba de Aceptación ID9

EDITAR EL PERFIL DEL USUARIO		
ID:	9	
ID Historia de usuario:	9	
Ambientación:	Condiciones normales.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Presionar sobre un campo para ser editado	Se desplegara una vista con solo el campo a ser editado	OK
Ingresar el nuevo valor	Poder digitar el nuevo valor en el campo de texto	OK
Presionar el botón para guardar la configuración	Guardar la configuración y regresar a la vista principal del menú de configuración.	OK
Pre-condiciones:	El usuario debe ingresar a la aplicación con sus credenciales y seleccionar el menú de configuración.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	El ejecutor logró editar su nombre, email y contraseña. También se mostro las alertas cuando trato de ingresar datos inválidos.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
Gloria	León	

Tabla 33. Prueba de Aceptación ID10

ELIMINAR CUENTA DE USUARIO		
ID:	10	
ID Historia de usuario:	9	
Ambientación:	Condiciones normales.	
SECUENCIA DE LA PRUEBA		
Procedimientos	**Resultados esperados**	**Ok**
Presionar sobre el campo de eliminar cuenta	Se desplegará una alerta solicitando al usuario confirmar la acción de eliminar la cuenta.	OK
Confirmar la eliminación de la cuenta.	Eliminar la cuenta y salir de la aplicación.	OK
Pre-condiciones:	El usuario debe ingresar a la aplicación con sus credenciales y seleccionar el menú de configuración.	
Post-condiciones:	-	
FALLAS		
Fallas Encontradas	**Descripción**	**Gravedad**
-	-	-
Comentarios:	Después de ser eliminada la cuenta del ejecutor con éxito, el ejecutor intentó ingresar nuevamente a la aplicación, con las credenciales de la cuenta eliminada, pero no tuvo éxito.	
EJECUTORES		
Nombre	**Apellido**	**Firma**
Gloria	León	

3.6.3 Pruebas de Caja Blanca

Esta prueba fue evaluada por un desarrollador móvil conocedor de las tecnologías usadas en la aplicación.

El evaluador tuvo acceso total al código de la aplicación y a un teléfono iPhone con la aplicación móvil instalada en su versión final.

Estas pruebas fueron divididas en módulos y sus resultados fueron los siguientes:

Tabla 34. Informe de prueba de caja blanca

Autenticación de un usuario		
Fecha Revisión	**Comentarios**	**OK**
10 de Febrero del 2014	La organización del código y su funcionalidad cumplen todo lo establecido por la historia de usuario.	OK
Creación de una cuenta de usuario		
Fecha Revisión	**Comentarios**	**OK**
10 de Febrero del 2014	La organización del código y su funcionalidad cumplen todo lo establecido por la historia de usuario.	OK
Visualizar información sobre la biciQ		
12 de Febrero del 2014	Se encuentra toda la información principal de la biciQ y la estructura interna de los datos es la adecuado, ya que es escalable.	OK
Visualizar las ultimas entradas del Facebook		
12 de Febrero del 2014	Cumple con lo requerido y a nivel de código existe una capa de abstracción en el API de Facebook y los métodos.	OK
Mostrar las estaciones de la biciQ		
17 de Febrero del 2014	Se muestran todas las estaciones de la biciQ y los lineamientos de código para manejo de mapas están establecidos de una forma clara.	OK

Crear una ruta en el mapa de las estaciones de la biciQ		
Fecha Revisión	**Comentarios**	**OK**
17 de Febrero del 2014	El código se encuentra bastante optimizado el cual ayuda a minimizar el uso de los recursos del dispositivo. También se ejecuto la funcionalidad de compartir la ruta en Facebook, la cual funciono de manera correcta.	OK
Editar y eliminar el perfil del usuario		
Fecha Revisión	**Comentarios**	**OK**
19 de Febrero del 2014	No se encontrar fallo alguno a nivel de código y funcionalidad.	OK

4. Conclusiones y Recomendaciones

4.1. Conclusiones

- Gracias al entorno de desarrollo Xcode, creado por Apple, se logró codificar la aplicación móvil para dispositivos iPhone 4 o superior. Se usó el lenguaje de programación Objective-C, siendo este el usado para desarrollos nativos para plataformas iOS. Xcode también facilitó a automatizar la pruebas unitarias, agilizar la búsqueda y solución de errores en el código, la integración de librerías externas y con la ayuda del simulador de dispositivos iOS, a realizar pruebas funcionales.

- La metodología de desarrollo extremo (XP), resulto de gran ayuda al establecer un proceso lógico y funcional de cómo manejar un proyecto de desarrollo de software. Empezando desde las historias de usuario al ser una herramienta simple y dinámica para obtener los requerimientos del proyecto, estas no fueron un problema para el cliente en entender ni completar dicho artefacto de la metodología XP. Para el desarrollador fue rápido captar las ideas plasmadas en dichas historias para definir el cronograma de iteraciones, las tarjetas técnicas de Clase-Responsabilidad-Colaboradores y los diagramas de entidades, con el fin de obtener un clara visión de las funcionalidades y arquitectura de la aplicación móvil y así empezar el desarrollo de una manera bastante practica y confiable. A nivel de codificación la metodología estableció un desarrollo basado en pruebas, considerando que el código y su estructura sea modular, simple y escalable, integrando este frecuentemente.
- A pesar de que algunas herramientas o procesos internos de la metodología de desarrollo extrema fueron diseñadas para un equipo de desarrollo, se logró ejecutar la gran mayoría de los procesos sin dificultad alguna.

- Usando la documentación que expone Apple en la siguiente dirección de internet: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html y tomando como ejemplo algunas aplicaciones móviles ya desarrolladas, fue posible crear una interfaz gráfica simple, atractiva y de fácil manejo para el usuario final.

- Observando código de otras aplicaciones desarrolladas en Objective-C y guías de desarrollo de empresas medianas y grandes, se estableció una guía de codificación sólida y clara, la cual fue aplicada durante todo el proceso de creación del código.

- Usando Xcode más una cuenta certificada de desarrolladores de aplicaciones para iOS, se logró probar la aplicación móvil en un dispositivo iPhone 4 con iOS 7.x.x y en un dispositivo iPhone 5S con iOS 7.x.x. Usando la misma herramienta, se creó casos de pruebas por cada acción que se puede realizar en la aplicación móvil, dentro de estas se implementó métodos que prueban características aisladas, estos métodos se los considera como pruebas unitarias. De esta manera se obtuvo constantemente un ambiente de prueba.

4.2. Recomendaciones

- Las peticiones que se realizan desde la aplicación móvil al servidor REST, deberían ser transportadas bajo un enlace seguro y cifrado, con el fin de proteger la integridad y veracidad de las credenciales del usuario.

- Para mantener el interés de los usuarios frente a la aplicación móvil, se sugiere hacer ligeras actualizaciones en el diseño y funcionalidades cada 4 meses.

- Se recomienda estar pendiente de las nuevas herramientas, métodos y estándares de diseño que estipula Apple cada cierto tiempo, con el fin de mantener un código actualizado y atractivo hacia el usuario final.

- Se debería implementar un nodo de respaldo para el servidor REST y para la base de datos y estos deben ser replicados de forma automática.

- Si, el servicio biciQ empieza a crecer de una manera rápida y constante, se debería implementar servicios REST para obtener las estaciones y la información general de la base de datos externa.

Referencias

Ambler, S. W. (2012). *Agile Modeling.* Recuperado el 5 de Junio de 2013 de http://www.agilemodeling.com/artifacts/userStory.htm

Apple. (n.d.). *iOS Developer Library.* Recuperado el 17 de Mayo de 2013 de Learning Objective-C: A Primer: https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/

BiciQ - Documentación. (2012). Recuperado el 5 de Mayo de 2013 de http://www.biciq.gob.ec/web/index.php/servicios/repositorio-de-archivos/category/documentacion.html

BiciQ. (2012). Recuperado el 5 de Mayo de 2013 de http://www.biciq.gob.ec/web/index.php/servicios/repositorio-de-archivos/category/guias.html

Bluk. (12 de Septiembre de 2009). *Apache Wink.* Recuperado el 29 de Septiembre de 2013 de Apache Wink : 1 Introduction to Apache Wink: http://wink.apache.org/documentation/1.0/html/1%20Introduction%20to%20Apache%20Wink.html

Chalmeida. (31 de Agosto de 2012). *1 078 personas pedalean por Quito en las bicis públicas | HOY | Noticias del Ecuador y el Mundo | Ecuador - Quito - Guayaquil.* Recuperado el 5 de Mayo de 2013 de http://www.hoy.com.ec/noticias-ecuador/1-078-personas-pedalean-por-quito-en-las-bicis-publicas-560412.html

Chromatic. (2009). *Extreme Programming Pocket Guide.* Sebastopol, Estados Unidos: O'reilly.

Ide, A. (2013). *PHP just grows & grows.* Recuperado el 18 de Mayo de 2013 de Netcraft: http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html

Miller, P. (2010). *iPhone 4 announced, launching June 24 for $199 with new Facetime video chat.* Recuperado el Mayo de 25 de 2013 de Engadget: http://www.engadget.com/2010/06/07/iphone-4-announced/

ANEXOS

ANEXO 1: MANUAL DE USUARIO

Aplicación Móvil

1. Ingreso de usuario

Figura 51. Pantalla de inicio de sesión

Para ingresar a la aplicación el usuario necesita digitar su correo electrónico, su contraseña y presionar el botón "ingresar". En caso de no disponer una cuenta para ingresar a la aplicación, debe presionar el link "Regístrate", para proceder a crear una cuenta.

1.1 Mensajes de error

Figura 52. Mensaje de error sobre el correo electrónico

Este mensaje es mostrado cuando el usuario no ingresa un correo electrónico o el correo electrónico esta incorrecto.

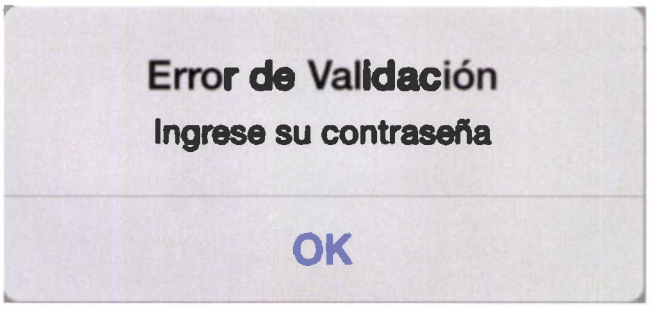


Figura 53. Mensaje de error sobre la contraseña de usuario

Este mensaje es mostrado cuando el usuario no ingresa una contraseña.

Figura 54. Mensaje de error sobre credenciales del usuario

Este mensaje es mostrado cuando la combinación de usuario y contraseña digitados, no se encuentra en la base de datos.

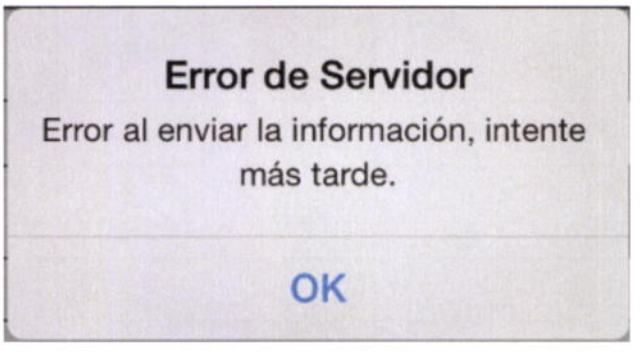


Figura 55. Mensaje de error de comunicación con el servidor

Este mensaje es mostrado cuando el dispositivo no esta conectado a una red con internet o el servidor tiene alguna falla.

2. Creación de una cuenta de usuario

Figura 56. Pantalla de creación de cuenta de usuario.

Para crear una cuenta, el usuario debe llenar todos los campos de texto, una vez completados dichos campos debe presionar el botón que se encuentra en la esquina superior derecha para guardas los datos y crear la cuenta.

Si los datos ingresados fueron procesados satisfactoriamente, el usuario ingresara a la aplicación de forma automática.

Si desea cancelar el proceso de creación de la cuenta de usuario y regresar a la pantalla de inicio de sesión, presionar el botón que se encuentra en la esquina superior izquierda

2.1 Mensajes de Error

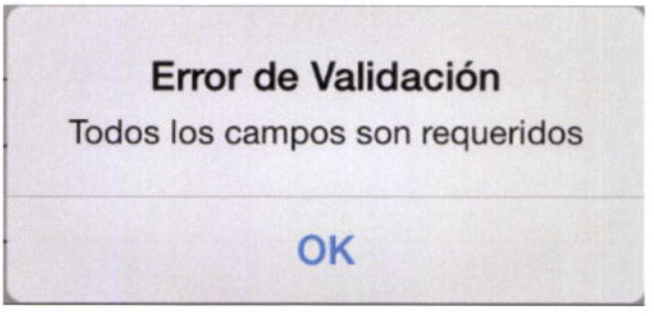


Figura 57. Mensaje de error acerca de los campos de texto

Este mensaje es mostrado cuando el usuario deja algún campo del formulario vacío.

Figura 58. Mensaje de error sobre el correo electrónico

Este mensaje es mostrado cuando el usuario no ingresa un correo electrónico o el correo electrónico esta incorrecto.

Figura 59. Mensaje de error sobre la validez de los datos

Este mensaje es mostrado cuando el usuario ingresa datos inválidos o el correo electrónico ya pertenece a un usuario ya registrado.

Figura 60. Mensaje de error de comunicación con el servidor.

Este mensaje es mostrado cuando el dispositivo no esta conectado a una red con internet o el servidor presenta alguna falla.

3. Ver información sobre la biciQ

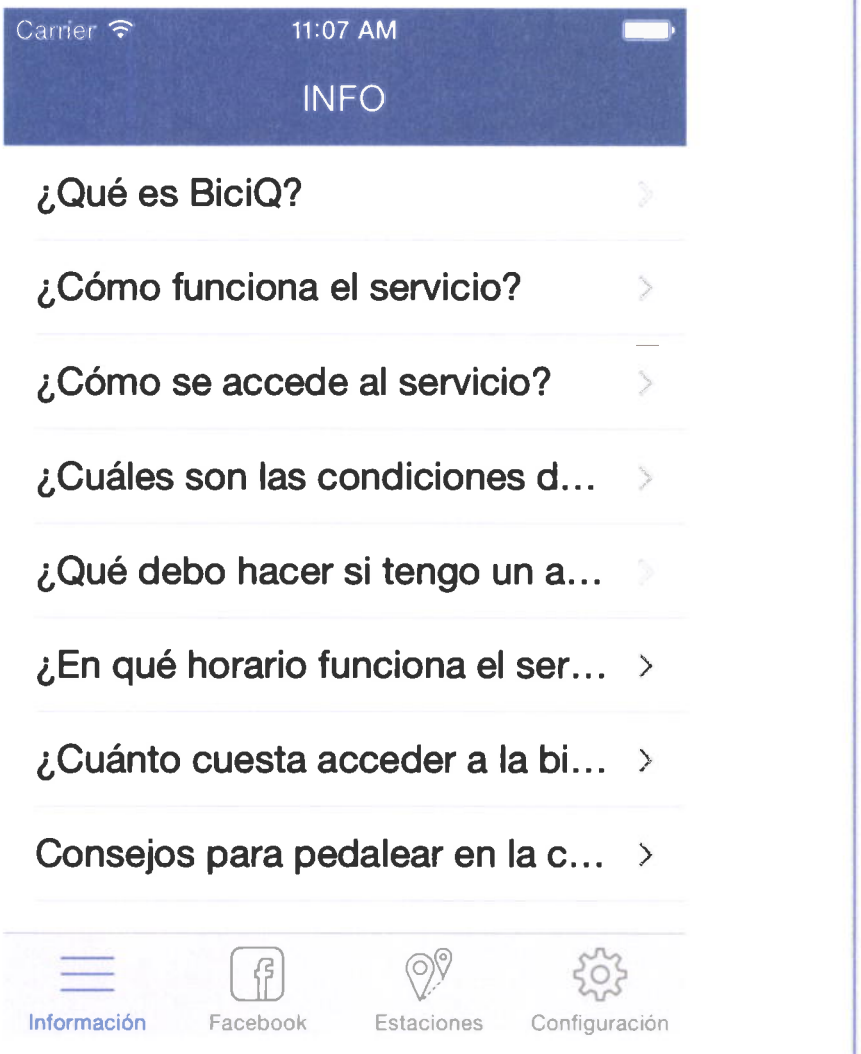


Figura 61. Pantalla con lista de datos informativos sobre el servicio.

Dentro de la aplicación, presionar el icono con el titulo "Información", ubicado en menú de la parte inferior de la pantalla.

La información es presentado en una tabla con desplazamiento vertical. Cada celda de la tabla tiene el titulo del dato informativo. Si el usuario desea ver más información debe presionar encima de la celda y se presentara una pantalla similar a la siguiente.

Figura 62. Pantalla información sobre el servicio biciQ.

En esta pantalla es presentado al usuario una imagen y el texto que describe el dato informativo que eligió en la pantalla anterior.

Cuando el texto es extenso, el usuario debe desplazar la pantalla para relevar la demás información.

Para regresar a la lista de todos datos informativos el usuario debe presionar el botón que se encuentra en la parte superior izquierda.

4. Ver las entradas de Facebook de la pagina oficial de biciQ

Figura 63. Pantalla con la lista de las últimas entradas en Facebook.

Dentro de la aplicación, presionar el icono con el titulo “Facebook”, ubicado en menú de la parte inferior de la pantalla.

La información es presentada en una tabla con desplazamiento vertical. Cada celda de la tabla tiene un extracto de la noticia y opcionalmente una imagen relevante.

Si el usuario desea cargar más entradas debe presionar el botón que se encuentra en la parte superior derecha.

Para ver la entrada completa de Facebook debe presionar encima de la celda y se presentara una pantalla similar a la siguiente.

Nota: Para ver las noticias de Facebook el usuario deberá ingresar sus credenciales de la cuenta de ese servicio, en un pantalla que será desplegada al momento de mostrar las noticias, esta pantalla de autorización será mostrada solo una vez.

Figura 64. Pantalla con la entrada completa de Facebook.

En esta pantalla es presentado al usuario una imagen y el texto completo de la entrada de Facebook que eligió en la pantalla anterior.

Cuando el texto es extenso, el usuario debe desplazar la pantalla para relevar la demás información.

Para regresar a la lista de todas las entradas de Facebook el usuario debe presionar el botón que se encuentra en la parte superior izquierda.

4.1 Mensajes de Error

Figura 65. Mensaje de error sobre la conexión con Facebook.

Este mensaje es mostrado cuando el teléfono no esta conectado a un a red con internet o cuando Facebook presenta alguna falla en su servicio.

5. Ver las estaciones de la biciQ

Figura 66. Pantalla que muestra la ubicación de las estaciones de la biciQ.

Dentro de la aplicación, presionar el icono con el titulo “Estaciones”, ubicado en menú de la parte inferior de la pantalla.

Cada estación de la biciQ se encuentra representada con el mismo icono, el logotipo de la biciQ contenido por un objeto de fondo rojo, la posición actual del usuario es representada por un icono redondo de color rojo y con borde blanco.

Al presionar el botón con fondo plomo, ubicado en la parte inferior izquierda del mapa, la vista del mapa se centrará y se acercara a la posición actual del usuario.

En la sección translucida ubicada en la parte inferior del mapa, tenemos la siguiente información:

- **Distancia:** Distancia de una estación a otra, en metros. Este dato estará activo cuando un ruta es creada.
- **Velocidad:** Velocidad actual del usuario, en kilómetros por hora. Este dato está siempre activo.
- **ETA:** Tiempo de llegada aproximado para completar la ruta, en minutos. Este dato estará activo cuando un ruta sea creada.

Para obtener más información sobre un estación en específico, el usuario deberá presionar sobre el icono de una parada y se desplegará una pantalla similar a la siguiente:

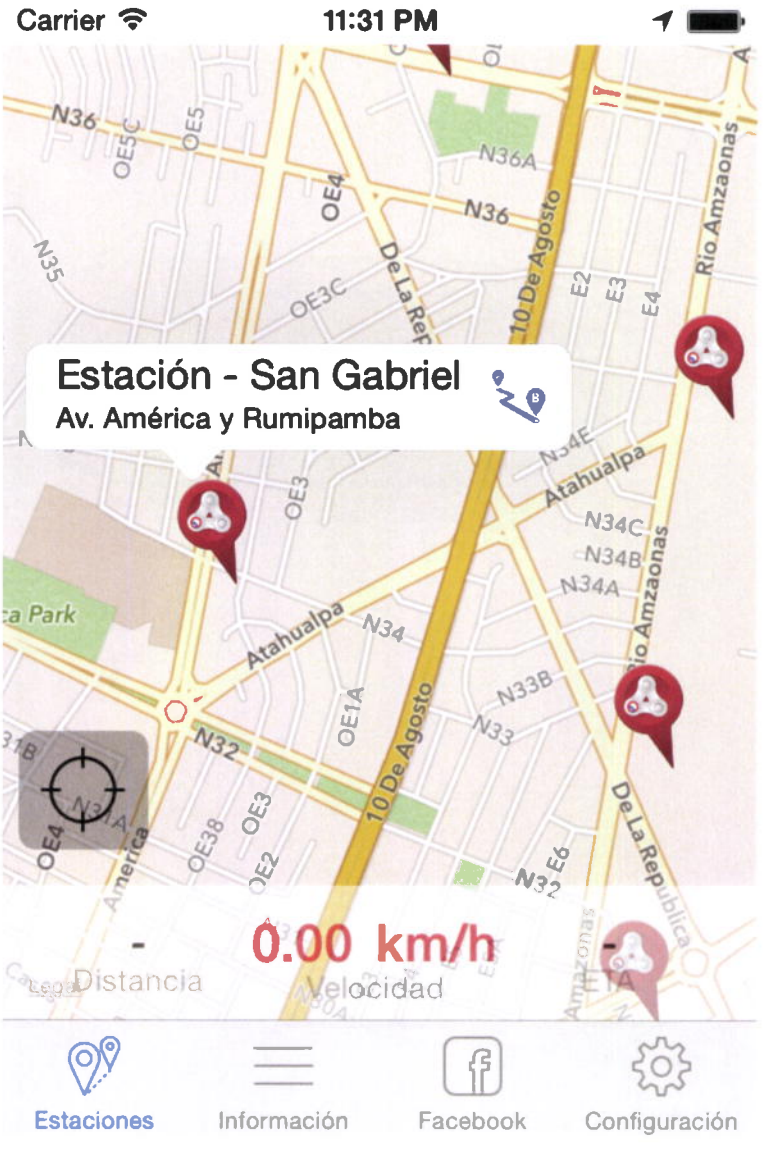


Figura 67. Pantalla que muestra el detalle de una estación de la biciQ.

Al presionar sobre el icono azul ubicado en la sección derecha de la ventana emergente de información de una estación, se desplegará las siguientes opciones:

Figura 68. Pantalla que muestra las opciones de generar una ruta.

En la pantalla emergente desplegada, tenemos la siguiente información:

- **Remover ruta actual:** Elimina la ruta existente y activa, que se encuentra en ese momento presente en el mapa.
- **Elegir como punto inicial:** Define la estación o ubicación actual como punto inicial de ruta a ser creada.
- **Elegir como punto final:** Define la estación o ubicación actual como punto final de ruta a ser creada.

- **Cancel:** Cierra la ventana emergente sin realizar cambio alguno sobre la ruta.

Una vez seleccionado el punto inicial y final se mostrará en el mapa la ruta y se actualizaran los datos informativos de la sección inferior translucida, conforme el usuario cambia de posición.

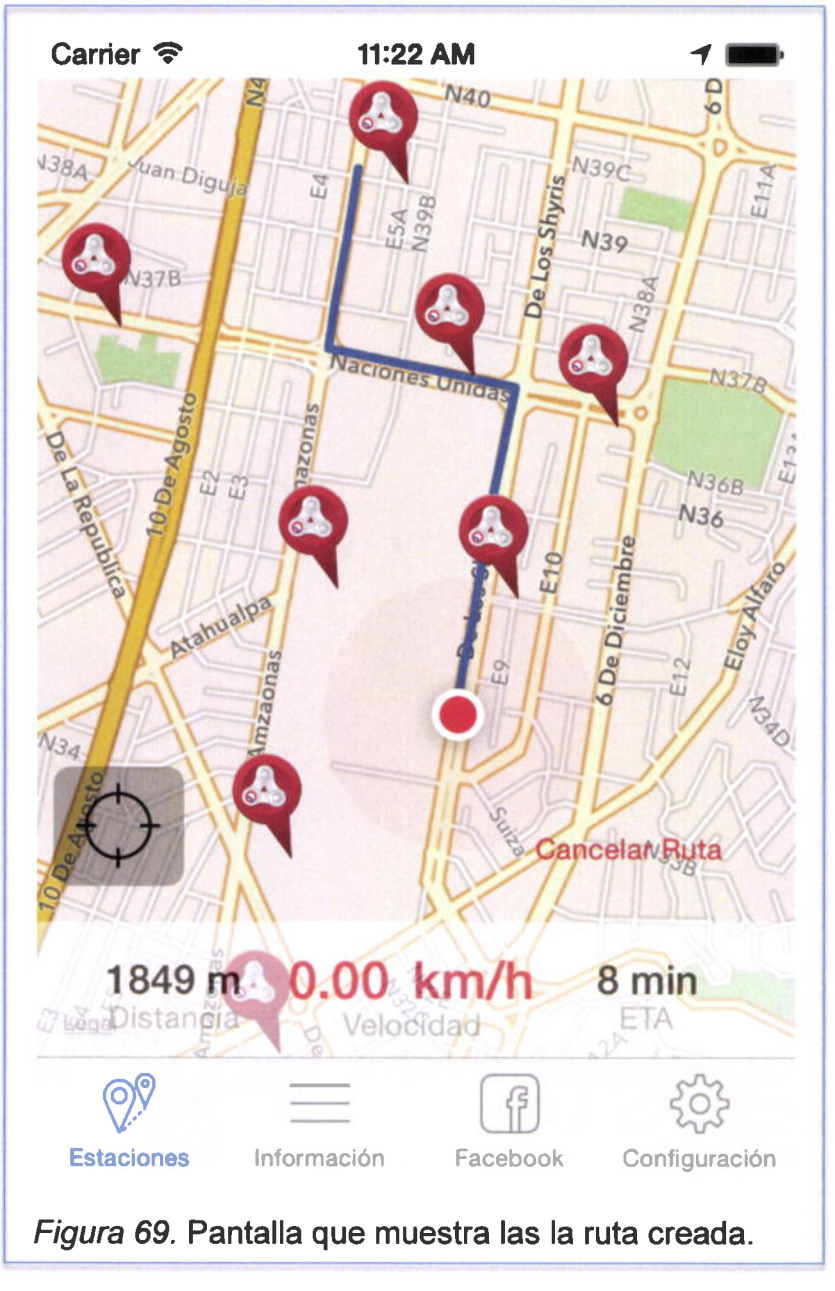


Figura 69. Pantalla que muestra las la ruta creada.

Una vez creada la ruta, el usuario podrá eliminar la ruta de dos maneras. La primera es mediante la pantalla emergente que es desplegada al momento de

presionar el icono de ruta en la ventana de información de una parada. La segunda opción es presionar el botón "Cancelar Ruta" que se encuentra en la esquina inferior derecha del mapa, este botón esta presente solo cuando una ruta se encuentra activa.

5.1 Mensajes de Permiso/Información

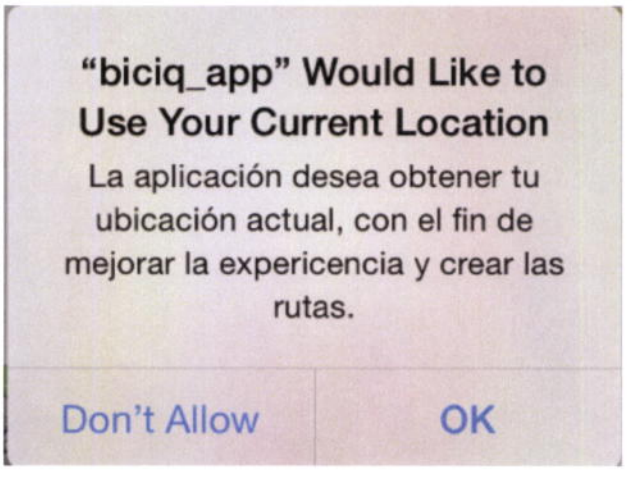


Figura 70. Mensaje sobre otorgar permiso de ubicación.

Este mensaje es mostrado la primera vez que iniciamos la aplicación, solicitando permiso al usuario para usar su ubicación actual con el fin de mostrarlas en el mapa y crear las rutas.

5.2 Mensajes de Error

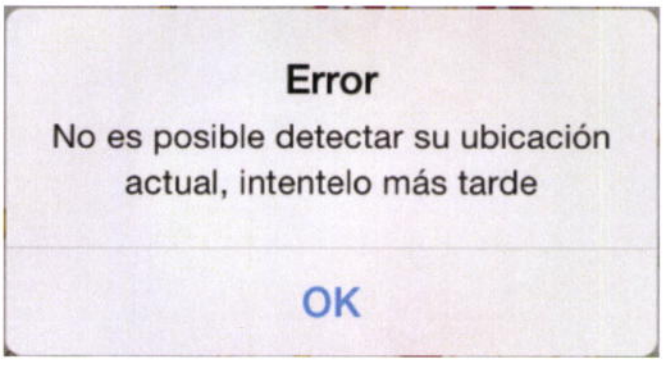


Figura 71. Mensaje de error sobre la ubicación actual del usuario.

Este mensaje es mostrado cuando no es posible detectar la ubicación actual del usuario debido a problemas de conexión o alguna falla en el *hardware*.

Figura 72. Mensaje de error II sobre la ubicación actual del usuario.

Este mensaje es mostrado porque el usuario denegó el permiso para que la aplicación acceda a su ubicación actual.

6. Configuraciones

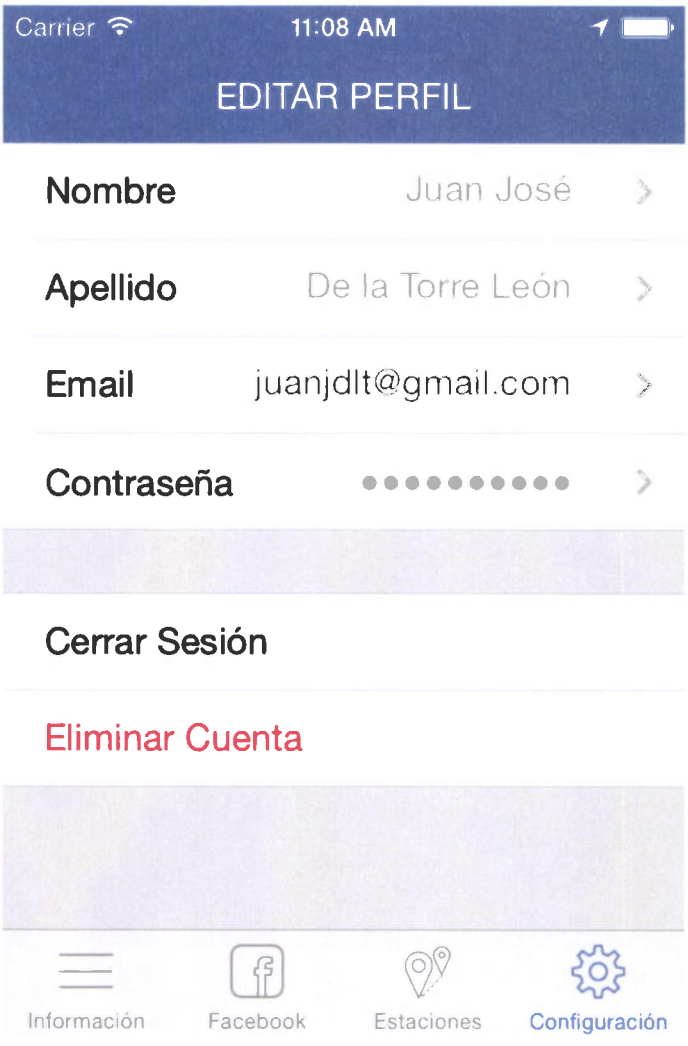


Figura 73. Pantalla sobre configuración de la cuenta de usuario

En esta pantalla el usuario podrá editar sus datos de la cuenta y cerrar la sesión actual.

Para editar sus datos personales, el usuario deberá editar los campos y después presionar el botón "Guardar" que se encuentra ubicado en la esquina superior derecha.

Para cerrar la sesión actual y volver a la pantalla de inicio de sesión, el usuario deberá presionar el botón "Cerrar Sesión" que se encuentra en la parte inferior del formulario.

6.1 Mensajes de Error

Figura 74. Mensaje de error acerca de los campos de texto

Este mensaje es mostrado cuando el usuario deja algún campo del formulario vacío.

Figura 75. Mensaje de error sobre el correo electrónico

Este mensaje es mostrado cuando el usuario no ingresa un correo electrónico o el correo electrónico esta incorrecto.

Figura 76. Mensaje de error sobre la validez de los datos

Este mensaje es mostrado cuando el usuario ingresa datos inválidos o el correo electrónico ya pertenece a un usuario registrado.

Figura 77. Mensaje de error de comunicación con el servidor.

Este mensaje es mostrado cuando el dispositivo no esta conectado a una red con internet o el servidor presenta alguna falla.