Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

IMPLEMENTACIÓN DE API RESTFUL PARA USO EN APP DE OFERTAS (CHAPLIST), DESARROLLADA CON IONIC

Tamy Alfredo Vivas Carrillo
Keneth Efrén Ubeda Arriaza
Asesorados por el Ing. Herman Igor Véliz Linares

Guatemala, octubre de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DE API RESTFUL PARA USO EN APP DE OFERTAS (CHAPLIST), DESARROLLADA CON IONIC

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

TAMY ALFREDO VIVAS CARRILLO
KENETH EFRÉN UBEDA ARRIAZA
ASESORADO POR EL ING. HERMAN IGOR VÉLIZ LINARES

AL CONFERÍRSELES EL TÍTULO DE

INGENIEROS EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Raúl Eduardo Ticún Córdova
VOCAL V	Br. Henry Fernando Duarte García
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
EXAMINADOR	Ing. Oscar Alejandro Paz Campos
EXAMINADOR	Ing. Miguel Ángel Cancinos Rendón
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presentamos a su consideración nuestro trabajo de graduación titulado:

IMPLEMENTACIÓN DE API RESTFUL PARA USO EN APP DE OFERTAS (CHAPLIST), DESARROLLADA CON IONIC

Tema que nos fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 7 de noviembre de 2015.

Tamy Alfredo Vivas Carrillo **Keneth Efrén Ubeda Arriaza**

Guatemala 15 de abril de 2016

Ingeniero

Marlon Antonio Pérez Turk

Director

Escuela de Ciencias y Sistemas

Facultad de Ingeniería

Universidad de San Carlos de Guatemala

Ingeniero Pérez Turk:

Me complace saludarle, haciendo referencia al trabajo de graduación titulado "IMPLEMENTACION DE API RESTFUL PARA USO EN APP DE OFERTAS (CHAPLIST), DESARROLLADA CON IONIC", desarrollado por el estudiante universitario Tamy Alfredo Vivas Carrillo con número de carné 201212589 y el estudiante universitario Keneth Efrén Ubeda Arriaza con carné 201212728, que como asesor apruebo el contenido del mismo.

Para su conocimiento y efectos, sin otro particular, me suscribo.

Atentamente,

Ing. Herman Igor Veliz Linares

Asesor

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 11 de Mayo del 2016

Ingeniero
Marlon Antonio Pérez Türk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación de los estudiantes **TAMY ALFREDO VIVAS CARRILLO** con carné **201212589**, y **KENETH EFRÉN UBEDA ARRIAZA** con carné **201212728**, titulado: **"IMPLEMENTACIÓN DE API RESTFUL PARA USO EN APP DE OFERTAS (CHAPLIST), DESARROLLADA CON IONIC"**, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación ***"IMPLEMENTACIÓN DE API RESTFUL PARA USO EN APP DE OFERTAS (CHAPLIST), DESARROLLADA CON IONIC"****, realizado por los estudiantes TAMY ALFREDO VIVAS CARRILLO y KENETH EFRÉN UBEDA ARRIAZA aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pérez Türk
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 21 de octubre de 2016

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 514-2016

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **IMPLEMENTACIÓN DE API RESTFUL PARA USO EN APP DE OFERTAS (CHAPLIST), DESARROLLADA CON IONIC,** presentado por los estudiantes universitarios: **Tamy Alfredo Vivas Carrillo y Keneth Efrén Ubeda Arriaza,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

Guatemala, octubre de 2016

/gdech

ACTO QUE DEDICO A:

Dios	Por darme la fuerza cada día para seguir adelante en todos los aspectos de mi vida.
Mi familia	Por creer en mí y apoyarme durante todos estos años, ellos son los verdaderos protagonistas de este logro.
Mis amigos	Por estar siempre pendiente de mis actividades y por las palabras de apoyo que me dieron.
Mis compañeros	Porque ellos fueron el gran soporte durante mi preparación académica y profesional.

Tamy Alfredo Vivas Carrillo

ACTO QUE DEDICO A:

Dios	Por guiarme, y porque cada vez que le necesité Él estuvo ahí para darme la sabiduría e inteligencia para realizar cada proyecto de mi vida.
Mi familia	Por apoyarme en cada etapa y circunstancia de mi vida y darme la oportunidad y facilidad de prepararme para un mejor futuro.
Mis amigos	Por estar siempre pendiente de mis actividades y por las palabras de apoyo que me dieron.
Mis compañeros	Porque ellos fueron el gran soporte durante mi preparación académica y profesional.

Keneth Efrén Ubeda Arriaza

AGRADECIMIENTOS A:

Dios	Por su gran bendición y gracia hacia mi persona.
Mi familia	Por proveerme tanto tiempo, no solo de apoyo sino también de recursos, y por haberme instruido en el camino de bien para edificar mi vida.
Mis compañeros	Por toda la ayuda y apoyo en los momentos difíciles.
Universidad de San Carlos de Guatemala	Por brindarme la oportunidad de estar en la mejor casa de estudios.

Tamy Alfredo Vivas Carrillo

AGRADECIMIENTOS A:

Dios	Por su compañía y bendiciones en todo momento.
Mi familia	Por su amor incondicional, su sustento y enseñarme principios y valores que me conducen por el camino correcto.
Mis compañeros	Por toda la ayuda y apoyo ante los momentos difíciles.
Universidad de San Carlos de Guatemala	Por brindarme la oportunidad de estar en la mejor casa de estudios.
Facultad de Ingeniería	Por enseñarme que debo esforzarme para lograr mis metas.

Keneth Efrén Ubeda Arriaza

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

API	Interfaz de programación de aplicaciones, conjunto de servicios para ser utilizado por otro software.
Android	Sistema operativo móvil propiedad de Google.
ACID	Iniciales de atomicidad, consistencia, aislamiento y durabilidad. Que son características de una base de datos.
APK	Paquete de aplicación Android.
Benchmark	Estudio que consiste prácticamente en comparar el rendimiento de un sistema con otro como referencia.
Clúster	Conjunto de computadoras o servidores que funcionan por medio de red como un único sistema.
Cifrado	Procedimiento que, mediante un algoritmo y una clave, convierte un mensaje .legible a otro ilegible con el fin de proteger la información.
Cookie	Información que un sitio web coloca en el disco duro de una persona sobre una sesión de navegación.
CSRF	Falsificación de petición en sitios cruzados.

Callback	Es una llamada de una función que envía como parámetro otra función en JavaScript.
CLI	Interfaz de línea de comandos.
C9	Cloud 9.
DOM	Árbol de componentes HTML.
Estatus	Estado en que se encuentra una transacción.
Framework	Marco de trabajo que ofrece funcionalidades para el diseño o construcción de software.
Http	Protocolo de transferencia de hipertexto.
Https	Protocolo seguro de transferencia de hipertexto.
Hash key	Clave que se introduce para crear las credenciales de una API.
IaaS	Infraestructura como servicio.
IOS	Sistema operativo móvil de Apple.
IU	Interfaz de usuario.
JSON	Notación de objetos JavaScript.

Java	Lenguaje de programación orientado a objetos, propiedad de Oracle.
Middleware	Software que es intermediario para la comunicación entre aplicaciones.
MVW	Modelo vista y cualquier otra cosa.
MVC	Modelo vista controlador.
ORM	Mapeo relacional de objetos.
OSI	Modelo de interconexión de sistemas abiertos, útil para la referencia de protocolos de red y arquitecturas en capas.
Open Source	Filosofía de liberar software con el fin de ser utilizado, modificado o mejorado por otras personas.
Package name	Identificador de un proyecto de desarrollo de una App móvil y que solo el desarrollador conoce.
Plugins	Complemento de una aplicación que añade funcionalidades.
Promesa	Concepto utilizado en AngularJS que sirve para trabajar peticiones asíncronas como si fuesen síncronas.

PaaS	Plataforma como servicio.
Performance	Determina el buen desenvolvimiento de un sistema ante cierto estrés.
Redis	Motor de base de datos que persiste la información en memoria utilizando tablas *hash*.
Secret key	Clave que se otorga, luego de registrarse, para el consumo de una API.
SaaS	Software como servicio.
SO	Sistema operativo.
SSL/TLS	Capa segura de sokets/capa segura de transporte.
Token	En seguridad componente electrónico que se proporciona a un usuario para la gestión de acceso a un conjunto de servicios.
TCP/IP	Transmission control protocol/Internet protocol.

RESUMEN

El presente trabajo consiste en una guía para el uso e implementación de distintas tecnologías para el desarrollo de APIs RESTful y aplicaciones híbridas, utilizando *framework*s para su construcción.

En el primer capítulo se explica en qué consiste cada tecnología utilizada, desde máquinas virtuales en la nube, hasta conceptos básicos sobre APIs RESTful; bajo el enfoque de su impacto en Guatemala.

El segundo capítulo identifica el problema mediante el estudio de antecedentes y la realización de un b*enchmarking,* con otras aplicaciones móviles que solventan de alguna forma el problema identificado. En este capítulo se plantea, como solución, una aplicación móvil para mostrar ofertas sobre productos de distintos supermercados.

En el tercer capítulo se define un prototipo para una aplicación móvil y se explican ciertos aspectos de la misma; su diseño intuitivo y la forma de uso. Se explica además, la arquitectura del sistema.

En el cuarto capítulo se describen todos los aspectos técnicos a considerar: desde *framework*s utilizados, hasta lenguajes de programación y herramientas para manipulación de datos. Contiene además, un tutorial que es la guía real para el desarrollo e implementación de todas las tecnologías utilizadas.

OBJETIVOS

General

Proporcionar una guía básica que permita a los estudiantes de las distintas ramas de la informática, implementar tecnologías web actuales para la creación de APIs RESTful y aplicaciones híbridas para solventar un problema específico.

Específicos

1. Crear una aplicación móvil orientada a ofertas de supermercados, combinando distintas tecnologías.

2. Generar interés a un público general sobre el uso de herramientas móviles para la gestión de actividades mecánicas que son realizadas por las personas cada día.

3. Dar a conocer las tecnologías que pueden utilizarse en el ámbito de desarrollo web e híbrido.

INTRODUCCIÓN

La acelerada expansión de la tecnología en la población ha causado un incremento en el uso de dispositivos móviles, como teléfonos inteligentes y tabletas, además, las personas requieren cada vez más aplicaciones móviles para automatizar sus tareas cotidianas.

Actualmente, las personas tienen varias opciones de supermercados y los visitan con el fin de comprar productos de consumo diario. Estos comercios exponen a sus clientes las ofertas en un tiempo determinado, por medio de catálogos de ofertas publicados en diferentes medios de comunicación, como: prensa, televisión, redes sociales, entre otros.

La población, como cliente de los supermercados, busca obtener los mejores productos al mejor precio; en algunos casos las personas visitan más de un supermercado para tomar la mejor decisión de compra. Para ello se implementó una App de ofertas de supermercado en la que se podrá acceder a los catálogos de los distintos supermercados.

1. ESTUDIO DE LA TECNOLOGÍA Y SU IMPACTO EN GUATEMALA

En Guatemala, en el ámbito de desarrollo de software, muchas empresas ya establecidas como bancos de alimentación, siguen utilizando tecnologías de desarrollo enfocadas a productividad (como Java, Net de Microsoft.) debido a que los costos de la migración de sus sistemas que actualmente están en producción se elevan.

A continuación se describen distintas tecnologías que pueden aportar una mejora en el performance de los sistemas que utilizan, y que además pueden reducir considerablemente costos.

1.1. Máquinas virtuales en la nube

Es una tecnología que está siendo muy utilizada por las personas o entidades que desarrollan software, debido a su bajo coste.

Para ello es necesario definir el significado de nube; este concepto es muy conocido por las personas actualmente, y consiste en una serie de servicios que son expuestos a las personas para su uso, mediante la vía de internet.

Los servicios que la nube ofrece son variados, pero se pueden definir en tres grandes grupos que son:

- SaaS: aplicación montada en la nube que puede ser utilizada por varios usuarios desde un navegador web. Se pueden listar como ejemplos a las aplicaciones que provee Google, como Google Docs, Gmail, entre otros.

- PaaS: provee un servidor web con un sistema operativo que, a su vez, provee un entorno de programación. Ejemplos de este servicio pueden ser Cloud 9, Heroku, entre otros.

- IaaS: provee una serie de componentes como conectividad en red, clúster de almacenamiento, balanceadores de carga, entre otros. Ejemplos: Digital Ocean, HP Cloud Compute, entre otros.

Las máquinas virtuales en la nube son un PaaS que provee un entorno de desarrollo, así como una base de datos relacional.

1.1.1. Ventajas

La ventaja de utilizar una máquina virtual en la nube, es que se reduce el costo de comprar hardware como un servidor, equipo de red, entre otros. Y también se reduce el tiempo de configuración del equipo, ya que estos servicios proveen un entorno ya pre configurado listo para trabajar.

1.1.2. Desventajas

Hay ciertos términos y condiciones que establece el proveedor del servicio como la cantidad de peticiones por segundo al servidor, condiciones de pago y otras a las que se debe apegar a la hora de utilizar un servicio como este.

1.2. Protocolo HTTPS y RESTful

A continuación se explicarán el protocolo https y el término RESTful.

1.2.1. HTTPS

Es un protocolo seguro de transferencia de hipertexto, por su traducción del inglés, ubicado en la capa siete del modelo OSI. Se diferencia del HTTP en que este es una versión segura para la transferencia de información, ya que utiliza un cifrado SSL/TLS. La característica principal del HTTPS es que crea una vía cifrada de comunicación entre el cliente y el servidor, protegiendo los datos ante la interferencia de un tercero.

1.2.2. REST/RESTful

Significa transferencia de estado representacional (*Representational State Transfer*), conocido como RESTful y es un estilo de arquitectura de software para recursos distribuidos que utiliza HTTP como vía de comunicación.

Entre sus principios se pueden mencionar:

- Utiliza los verbos HTTP

Dentro de los verbos que utiliza REST como *get*, *post*, *put*, *delete* y se tiene la siguiente convención para su uso:

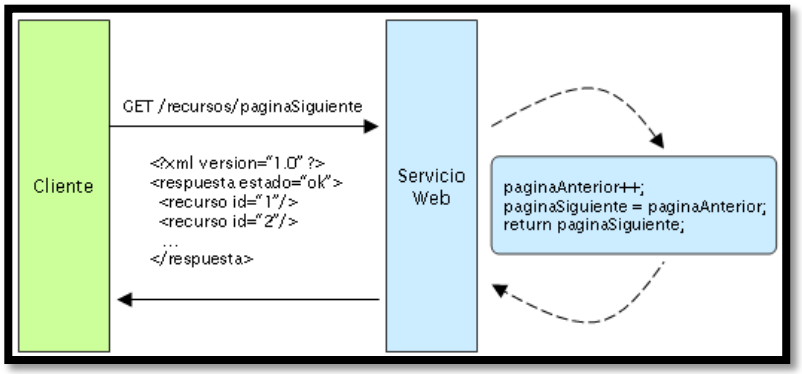
  - *Get*: comúnmente utilizado cuando se quiere obtener algún recurso.

- *Post*: utilizado para la creación de algún recurso.
- *Put*: para actualizar un recurso.
- *Delete*: utilizado para eliminar un recurso.

- No mantiene estados

Muchas veces se requiere tener un sistema que pueda escalar o incluso existen sistemas complejos que tienen la necesidad de utilizar balanceadores de carga entre dos o más servidores, o también implementan servidores intermedios para manejar todas las peticiones.

Figura 1. **Petición con estado**

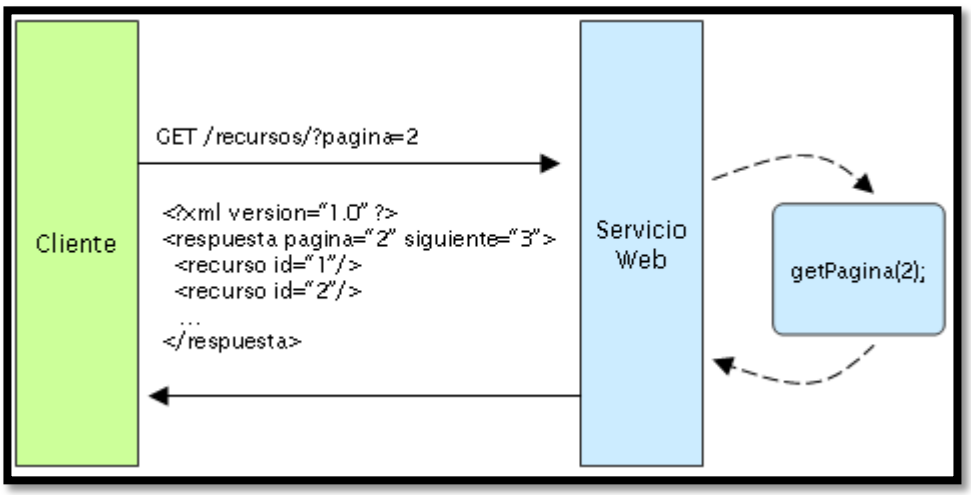


Fuente: *Introducción a los servicios web RESTful*. http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html. Consulta: marzo de 2016.

Es por este motivo que guardar el estado de cada petición resulta un problema más complicado de lo que se puede manejar; ante esto REST no

guarda estados, lo que quiere decir que cada petición fluye de forma independiente.

Figura 2. **Petición sin estado**



Fuente: *Introducción a los servicios web RESTful*. http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html. Consulta: marzo de 2016.

- **Expone las URLs en forma de directorios**

Es una particularidad en donde hace que REST sea muy sencillo de utilizar y de comprender, ya que los usuarios pueden acceder a los recursos mediante una URL con forma de directorio, esto hace que se vuelva intuitiva la manera de acceder a dichos recursos, por ejemplo:

  - http://www.miservidor.com/api/v1/cliente/{id}.

Se pueden ir utilizando los distintos verbos HTTP para ir descubriendo las distintas funcionalidades que se le pueden hacer a una misma URL.

- Puede transferir distintos lenguajes de marcado

Dentro los más utilizados en internet se pueden mencionar: XML y JSON. REST puede trabajar con ambos tipos de marcado ante las distintas peticiones.

Un buen servicio REST tiene que ser lo más simple posible, es por eso que en el atributo HTTP Accept del encabezado puede indicarse el tipo de contenido que se requiere o MIME (Media-Type).

Tabla I. **Tipos MIME más utilizados en servicios REST**

MIME-Type	Content-Type
JSON	Application/json
XML	Application/xml
XHTML	Application/xhtml+xml

Fuente: *Introducción a los servicios web RESTful*. http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html. Consulta: marzo de 2016.

1.3. Aplicaciones móviles

Las empresas, independientemente si son de desarrollo o no, tienen que tomar una decisión a la hora de desarrollar una aplicación móvil que involucre factores como: costo, tiempo, implementación, comunicación, entre otros. Por eso se tiene que tomar una decisión sobre desarrollar nativo, web o híbrido.

A continuación se describe en qué consiste cada grupo de aplicaciones.

1.3.1. Aplicaciones nativas

Son aquellas que están desarrolladas para que se ejecuten directamente en un SO específico y que, para lograr que funcionen en dos SO, es necesario desarrollar la misma aplicación en dos ambientes diferentes. Por ejemplo, una aplicación desarrollada en Java-Android, funcionará sobre el SO Android, pero para que funcione en un sistema IOS es necesario construirla en Objective-C, Swift, entre otros.

Una aplicación nativa puede consumir todas las APIs que posee el SO sobre el cual está ejecutándose dicha aplicación. Dependiendo del SO utilizado, así serán los lenguajes y las herramientas que se poseen para desarrollar una aplicación nativa.

Tabla II. **Diferencias para el desarrollo de aplicaciones nativas**

	Apple IOS	**Android**	**Blackberry Os**	**Windows Phone**
Lenguajes	Objective-C	Java	Java	C#, VB.Net
Herramientas	Xcode	Android SDK	BB Java Eclipse Plug-in	Visual Studio, Windows Phone
Formato	.App	.apk	.cod	.xap
Tiendas	App Store	Google Play	Blackberry App World	Windows Phone Market

Fuente: IBM. *El desarrollo de aplicaciones móviles nativas, web o híbridas*. p. 3.

1.3.2. Aplicaciones móviles web

Estas son ejecutadas desde los navegadores web que poseen los diferentes dispositivos móviles como Chrome o Safari (Android e IOS, respectivamente) y se construyen con HTML, CSS, JavaScript, entre otros.

Tabla III. **Diferencia entre aplicaciones web contra sitios web**

Característica	**Apl. web solo móviles**	**Sitios web solo móviles**
Herramientas y conocimientos	Escritas totalmente en HTML, CSS y JavaScript	Escritas totalmente en HTML, CSS y JavaScript
Ejecución	Acceso directo instalado, lanzado mediante apl. Nativa	Navegando por un sitio mediante URL
Experiencia del usuario	Touch, IU interactiva	IU mediante navegación entre páginas que muestran datos estáticos
Desempeño	IU reside localmente; aplicación con capacidad de respuesta y acceso offline	Todo el código se ejecuta desde un servidor; el rendimiento depende de la red

Fuente: IBM. *El desarrollo de aplicaciones móviles nativas, web o híbridas*. p. 5.

1.3.3. Aplicaciones híbridas

Este grupo combina el desarrollo nativo con distintas tecnologías de desarrollo web. Quiere decir, que se puede construir una aplicación móvil, en su mayoría con tecnologías web, teniendo comunicación con las APIs nativas y siendo multiplataforma.

La porción nativa de la aplicación emplea APIs del SO para crear un motor de búsqueda HTML, incorporado, que funcione como un puente entre el navegador y las APIs del dispositivo.

Existen varios *frameworks* open source para desarrollo híbrido como PhoneGap, Ionic, jQuery Mobile, entre otros.

1.3.4. Comparación entre aplicaciones móviles

Las aplicaciones nativas siempre serán mejores por su performance y por la forma en que se ejecutan directamente en el SO, pero tiene un alto costo al requerir un cambio, además que no son multiplataforma.

Las aplicaciones web, por su parte, son muy fáciles de desarrollar, son multiplataforma, pero su funcionalidad queda limitada al no utilizar las APIs nativas del SO.

Por último, las aplicaciones híbridas serían la combinación de lo mejor de ambos como el desarrollo web con comunicación a las APIs nativas del SO; y son multiplataforma.

Figura 3. **Aspecto visual entre las distintas aplicaciones móviles**

Fuente: IBM. *El desarrollo de aplicaciones móviles nativas, web o híbridas*. p. 7.

Tabla IV. **Comparación entre las distintas aplicaciones móviles**

Característica	**Aplicación nativa**	**Aplicación híbrida**	**Aplicación web**
Lenguaje de desarrollo	Solo nativo	Nativo y web o solo nativo	Solo web
Portabilidad y optimización de código	Bajo	Alto	Alto
Características de acceso específicas del dispositivo	Alto	Mediano	Bajo
Uso de conocimiento existente	Bajo	Alto	Alto
Gráficos avanzados	Alto	Mediano	Mediano
Flexibilidad de actualizaciones	Bajo (siempre tiendas)	Mediano (con frecuencia tiendas)	Alto
Experiencia de instalación	Alta (a partir de tienda)	Alta (a partir de la tienda)	Mediana (mediante navegador móvil)

Fuente: IBM. *El desarrollo de aplicaciones móviles nativas, web o híbridas*. p. 8.

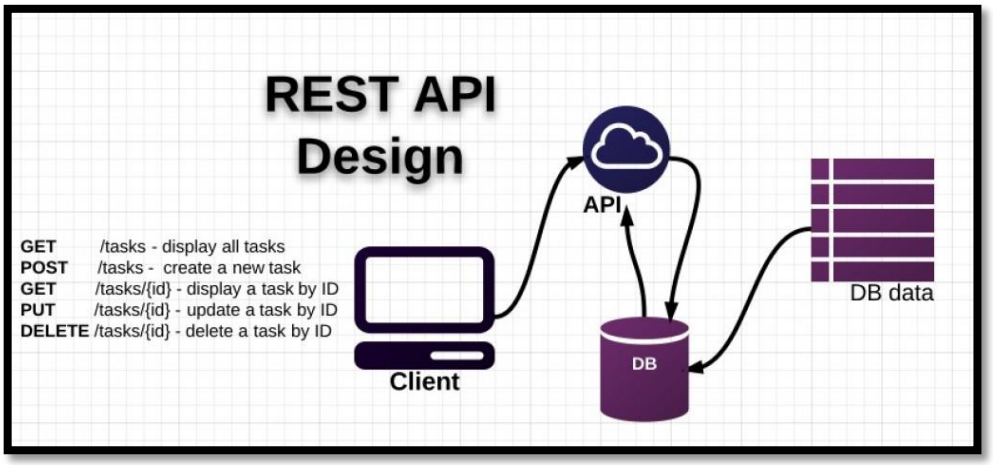
1.4. API REST y autenticación por *token*

A continuación se explicara la autenticación por *token* aplicada a las api rest.

1.4.1. API REST

Es un conjunto de funciones que son accedidas mediante el protocolo HTTP. El fin de una API REST es proveer servicios web que pueden ser consumidos por otros desarrolladores en distintos tipos de aplicaciones como sitios web, aplicaciones móviles, entre otros.

Figura 4. **Diseño de una API REST**



Fuente: SURGUY Maks. *Building RESTful API in Laravel*. http://maxoffsky.com/code-blog/building-restful-api-in-laravel-start-here/. Consulta: marzo de 2016.

Claros ejemplos de APIS REST son las que proveen Google, Twitter, Facebook, entre otros. Que dependiendo de la API así serán sus restricciones.

1.4.2. Autenticación con *token*

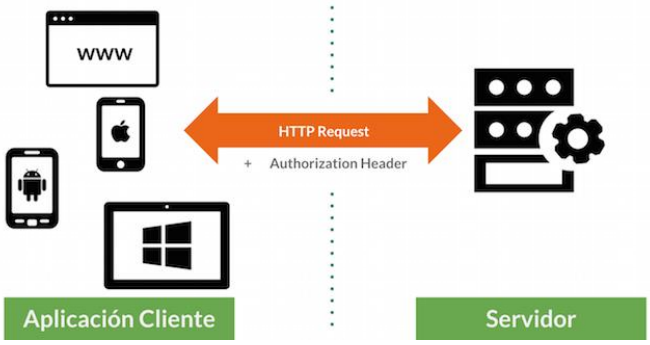
Una de las partes que no está muy bien definida en una API REST es la seguridad. No existe un estándar o regla a la hora de definir los servicios. Es por eso que una buena opción es el manejo de autenticación con *token*.

Esto contribuye en gran medida, ya que al autenticarse a la API REST, esta valida y devuelve un *token* generado, que será el que debe acompañar toda petición HTTP que realicen los usuarios.

Con esto la seguridad aumenta al no utilizar *cookies* para almacenar información de usuario, y de esa forma impedir ataques que manipulen dicha

información (CSRF Cross-Site Request Forgery). Para agregar más flexibilidad a la API REST, los *token*s que se generan deben ser almacenados del lado del cliente, así de esta forma se evita utilizar espacio en disco o memoria del servidor, y este último solo se encargará de descifrar el *token* y validar su autorización.

Figura 5. **Peticiones utilizando *token* en cabecera HTTP**



Fuente: AZAUSTRE, Carlos. *¿Qué es la autenticación basada en token?.* http://carlosazaustre.es/blog/que-es-la-autenticacion-con-*token*/. Consulta: marzo de 2016.

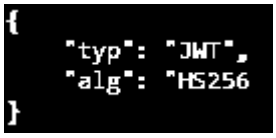
1.4.2.1. Estructura de un *token* con JSON WEB *token*s (JWT)

Un *token* con formato JWT está construido por 3 cadenas de texto (strings) separadas por punto, por ejemplo:

“eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1NGE4Y2U2MThlOTFiMGIxMzY2NWUyZjkiLCJpYXQiOiIxNDI0MTgwNDg0IiwiZXhwIjoiMTQyNTM5MDE0MiJ9.yk4nouUteW54F1HbWtgg1wJxeDjqDA_8AhUPyjE5K0U”

- *header*: es la primera cadena que compone el *token*, que a su vez está dividido en otras dos partes que son el tipo y la codificación utilizada como HMAC SHA256.

Figura 6. **Contenido del *header***



Fuente: elaboración propia, empleando programa *Brackets*.

El contenido ya codificado queda de la siguiente forma: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

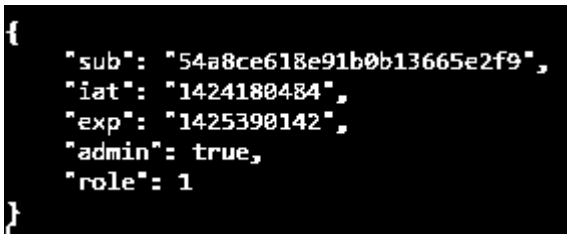
- *payload*: contiene atributos que definen el *token* y que son llamados JWT Claims, en donde los más comunes son:

    - Sub: es el sujeto del *token*, puede ser un identificador de usuario.

    - Iat: define la fecha de creación del *token* en formato de tiempo UNIX.

    - Exp: define la fecha de expiración del *token* en formato de tiempo UNIX.

Figura 7. **Contenido del *payload***

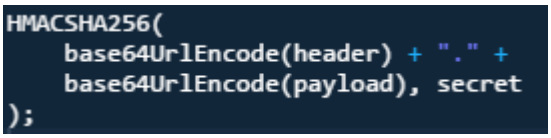


Fuente: elaboración propia, empleando programa *Brackets*.

En la figura 7 se muestra que pueden agregarse más campos al *payload*, y que codificado queda de la siguiente manera: eyJzdWIiOiI1NGE4Y2U2MThlOTFiMGIxMzY2NWUyZjkiLCJpYXQiOiIxNDI0MTgwNDg0IiwiZXhwIjoiMTQyNTM5MDE0MiJ9.

- *signature*: esta última parte está formada por el *header* y el *payload*, con la característica que van cifrados en Base64 con una clave secreta que tiene que estar almacenada en la API REST.

Figura 8. **Contenido del *signature***



Fuente: elaboración propia, empleando programa *Brackets*.

Codificada la parte del *signature* queda de la siguiente forma: yk4nouUteW54F1HbWtgg1wJxeDjqDA_8AhUPyjE5K0U, que es la última parte

que conforma al *token* con estructura JWT. Para comprobar este tipo de *token*, existe un sitio web que es JWT.io.

2. IDENTIFICACIÓN DEL PROBLEMA Y SOLUCIÓN PLANTEADA

En Guatemala es común que las empresas como: supermercados, farmacias, tiendas de electrodomésticos y tiendas de tecnología, entre otras, publiquen sus ofertas, sea semanales, mensuales o bien en un rango de fechas determinado.

Por lo regular, las empresas publican este tipo de ofertas por un medio impreso, como la prensa, revistas, entre otros. Algunas empresas tienen un apartado de ofertas en sus páginas web y las que más invierten en publicidad tienen anuncios de televisión.

En la actualidad, las aplicaciones móviles tienen un gran impacto en la población guatemalteca, ya que operan 21.7 millones de líneas de telefonía móvil, según la Superintendencia de Telecomunicaciones (SIT). Del total de líneas se estima que entre el 8 y 10 % del mercado tienen un sistema operativo inteligente, es decir de cada 100 teléfonos entre 8 y 10 son Smartphone. Por lo que la solución planteada consiste en una App, para teléfonos inteligentes con sistema operativo Android que sea capaz de mostrar las ofertas mencionadas anteriormente de una forma agradable, cómoda e intuitiva.

2.1. Antecedentes

El término oferta se utiliza para enfatizar el precio de un producto determinado, indicando que este es menor al normal o bien un conjunto de productos que al comprarlos su precio unitario disminuye. Este término es muy utilizado en el mercado a nivel mundial, ya que la competencia entre las diferentes industrias es muy fuerte. Cada institución se ve obligada a promocionar sus productos empleando distintas estrategias, entre las cuales está la oferta.

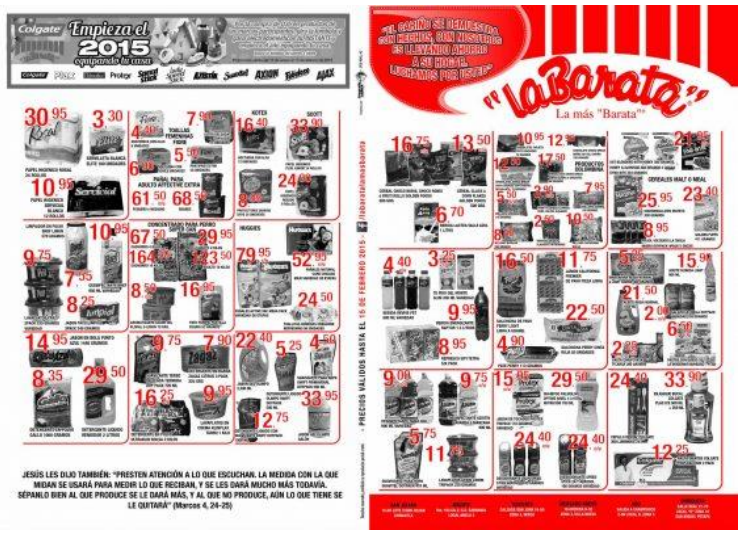
Es importante que las empresas que se dedican a vender productos al consumidor final hagan ofertas a sus clientes o consumidores ya que de esta manera captan su atención. Por ejemplo, en Guatemala se tiene establecimientos como La Torre, Walmart, Econosúper, Maxibodegas, La Despensa, La Barata, Paiz, supermercados que se dedican a la venta de productos de uso cotidiano. Este tipo de industria tiene como estrategia fundamental para su negocio publicar ofertas en un periodo establecido, semanal, mensual, etc. Por lo regular, este tipo de establecimientos publican catálogos de ofertas en la prensa o medios impresos que se encuentran en los locales para que las personas puedan tomar uno y así puedan revisar cuáles son los productos en oferta.

A continuación se ilustrarán la forma tradicional de catálogos de ofertas en Guatemala utilizados por algunos de los más prestigios supermercados del país.

Figura 9. **Ofertas La Torre**

Fuente: *Encuentra tu producto preferido #Toledo en oferta en supermercados La Torre.* https://www.pinterest.com/pin/465630048946056104/. Consulta: marzo de 2016.

Figura 10. **Ofertas La Barata**



Fuente: *La Barata Molino de Las Flores.* http://directorio.guatemala.com/listado/la-barata-molino-de-las-flores.html. Consulta: marzo de 2016.

Figura 11. **Ofertas Walmart**

Fuente: elaborado por Prensa Libre.

Como se puede notar, las publicaciones de ofertas tienen una estructura similar y los medios utilizados son los mismos. Por lo que, la información que se muestra en una oferta tradicional puede ser mostrada en una App móvil para optimizar el proceso de búsqueda y detalle.

2.2. Descripción del problema

Los clientes de las ofertas de los establecimientos, por lo regular, se enteran de las ofertas solamente si compran algún medio escrito o bien cuando ya se encuentran en el local del establecimiento. Esto impide que los clientes tomen la mejor decisión al adquirir sus productos, ya que no pueden comparar precios entre los productos de diferentes supermercados.

Los catálogos de ofertas suelen estar muy cargados de productos y no siempre es fácil localizar el producto de mayor interés. Los clientes deberían tener la facilidad de buscar sus productos más frecuentes y mantener un listado de los mismos para identificar fácilmente los de mayor interés.

Otra dificultad que se puede presentar a los clientes, es que la información les llega tarde y cuando se dan cuenta, ya hicieron su compra y no sabían que había productos en oferta. Además, aun cuando se habla de la misma empresa, dependiendo de la localidad, sus precios de ofertas varían, entonces, para los clientes no es fácil saber si el lugar que ellos frecuentan aplican las ofertas que están publicadas, o bien cuando llegan al local, no saben si en otra los productos están en oferta.

El aspecto clave o más relevante es que las ofertas del mismo producto no se encuentran en el mismo establecimiento, sino que se puede encontrar el mismo producto a un precio de oferta distinto en los diferentes establecimientos. Este punto es clave, ya que para el cliente es importante encontrar su producto al mejor precio.

2.3. Solución planteada

El medio por el cual el catálogo de ofertas es publicado, limita al cliente en cuanto a tomar la mejor decisión de compra.

Por lo que se propone una App móvil que permita visualizar las ofertas de los distintos establecimientos, categorizadas por establecimiento, en la que se muestre un catálogo de productos indicando su precio normal y el de oferta. Además, el cliente podrá localizar el establecimiento en donde la oferta se encuentra disponible y, para mejorar la difusión de los mejores precios, el

usuario podrá compartir sus ofertas con su círculo social. Por lo que la aplicación tiene 3 pilares:

- Encontrar la mejor oferta
- Ubicar el establecimiento de la mejor oferta
- Compartir la mejor oferta

2.3.1. Encontrar la mejor oferta

Consiste en tener un catálogo de ofertas categorizadas por establecimiento. Cuando el usuario desee visualizar las ofertas disponibles tendrá una pantalla en la que se mostrarán los establecimientos que tienen ofertas disponibles; al elegir un establecimiento se redirigirá al catálogo de productos en sí y el usuario podrá navegar entre los productos en oferta. El usuario también podrá ver el detalle de los productos y su descripción para una mejor visualización de la oferta.

Al visualizar las ofertas y su detalle, el usuario podrá marcar productos como favoritos con el fin de identificar los productos que él más frecuenta y así tener un acceso actualizado y rápido a dichos productos

Con el fin de facilitar la búsqueda, también se implementará un buscador en el que se escribe alguna palabra y se mostrarán todas las coincidencias que se encuentran en el catálogo del establecimiento seleccionado.

Cuando un producto es marcado como favorito hay un contador que va aumentado con el fin de llevar un conteo de cuántas personas ese producto es favorito. Se utiliza este contador, ya que en la pantalla de inicio se mostrarán las

ofertas con más favoritos que los usuarios han marcado y así se podrá tener acceso a los productos preferidos por los usuarios.

2.3.2. Ubicar el establecimiento de la mejor oferta

Es importante conocer las ofertas disponibles que los distintos establecimientos ofrecen; sin embargo, aunque al enterarse de la oferta y conocer los precios, es fundamental saber en dónde se encuentra esa oferta y dónde es el lugar más cercano para que se pueda adquirir el producto.

Dado que ubicar el producto en oferta es importante, la solución propuesta tiene integración con los mapas de Google y tiene identificados los distintos establecimientos en los que se puede adquirir las ofertas, además hay un detalle como dirección, número telefónico, entre otros.

Un aspecto importante es saber dónde el usuario está ubicado, por lo que si el usuario tienen encendido su GPS y da el permiso de ser ubicado la, solución podrá mostrar en el mapa su ubicación y, por consiguiente, los establecimientos a su alrededor.

2.3.3. Compartir la mejor oferta

El aspecto social es fundamental entre las personas, como se ha podido notar, las herramientas con más apoyo y crecimiento en la actualidad son las redes sociales entre las que se puede mencionar: Facebook, Twitter, Instagram, Snapchat, entre otras. Sin duda es importante que la solución propuesta para ser una solución completa esté integrada con el aspecto social.

En medida que la misma sociedad comparta las ofertas, el alcance de las mismas será mayor que, si simplemente se publican en medios electrónicos, escritos o de televisión. Dado que el aspecto social es importante, la solución propuesta tendrá una integración con Facebook para compartir las ofertas que más le han gustado al usuario.

La integración con Facebook consiste en que el círculo de amigos del usuario que esté usando la solución podrá compartir, hacer publicaciones entre ellos con el producto de oferta que se haya seleccionado. El usuario también podrá hacer publicaciones en su muro con las ofertas de su interés y así difundir las ofertas por medio de Facebook.

La idea principal de integrar la solución con Facebook es que los usuarios puedan difundir las ofertas y así tener un mayor alcance y las personas puedan adquirir sus productos a un mejor precio.

2.4. *Benchmarking*

Es importante para medir el rendimiento de la App tener un parámetro de comparación y para ello se utiliza el *benchmarking*, que consiste en una serie de comparaciones con Apps similares a la que se propone y con ello tener un mejor criterio de cambios y mejoras de la App.

A continuación se comparará la App con dos Apps cuyo propósito es similar.

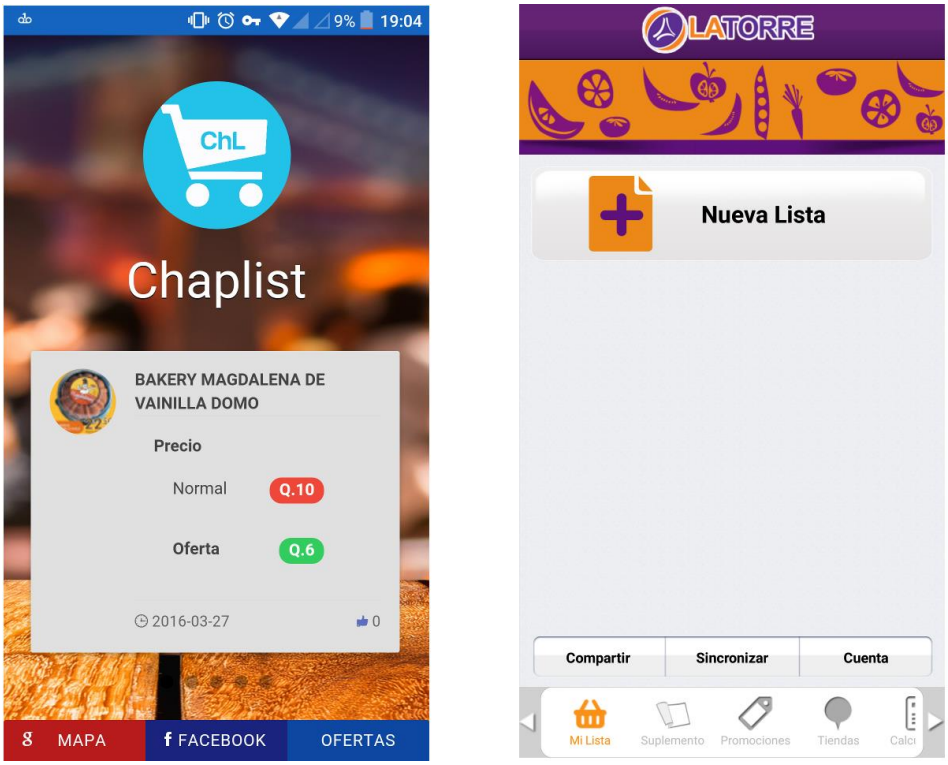
2.4.1. La Torre App

A continuación se comparará La Torre App contra ChapList.

2.4.1.1. Pantalla inicial

En la pantalla inicial ChapList muestra una sección de las ofertas más consultadas por los usuarios, también tiene acceso a los mapas, Facebook y las ofertas. Por su parte, La Torre App tiene como pantalla principal las listas de productos que los usuarios realizan. También tiene accesos para compartir las listas, sincronizar con una cuenta de Facebook y sus demás funcionalidades.

Figura 12. **Pantalla inicio ChapList La Torre**

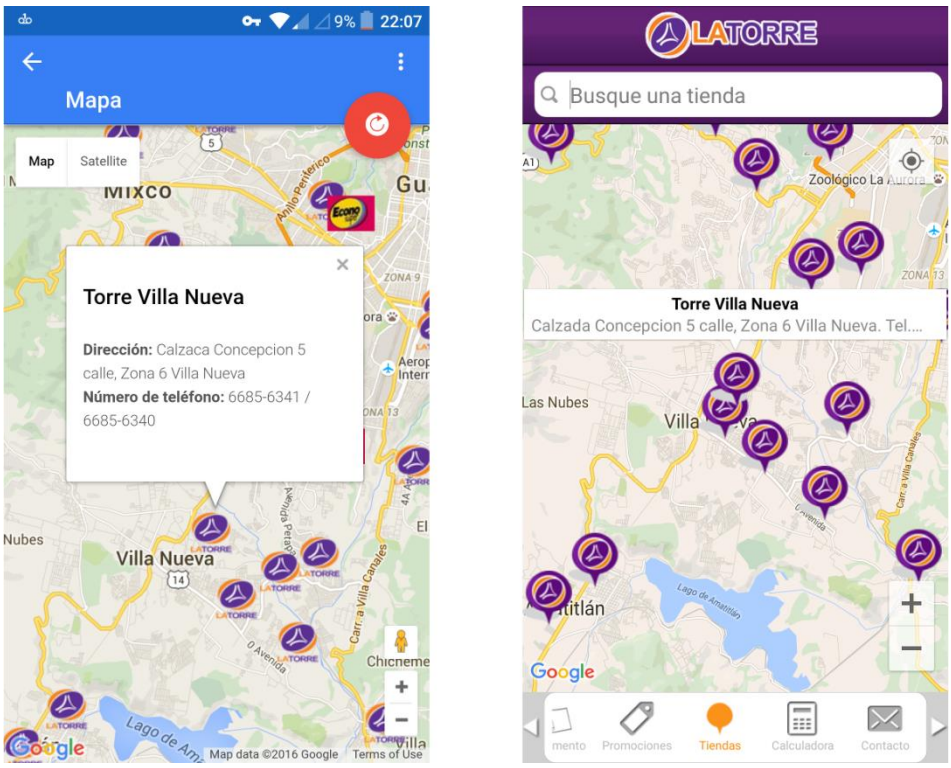


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.1.2. Mapas

En la pantalla de mapas ChapList posee claridad al momento de mostrar la información de los supermercados, además posee la función de localización mediante GPS para ubicar la localización actual. La Torre App posee al igual que ChapList mapas para ubicar sus propias tiendas la ventaja que posee La Torre App en este caso, respecto a ChapList, es que tiene un buscador, lo cual hace el proceso de localización más personalizado y sencillo.

Figura 13. **Mapas ChapList La Torre**

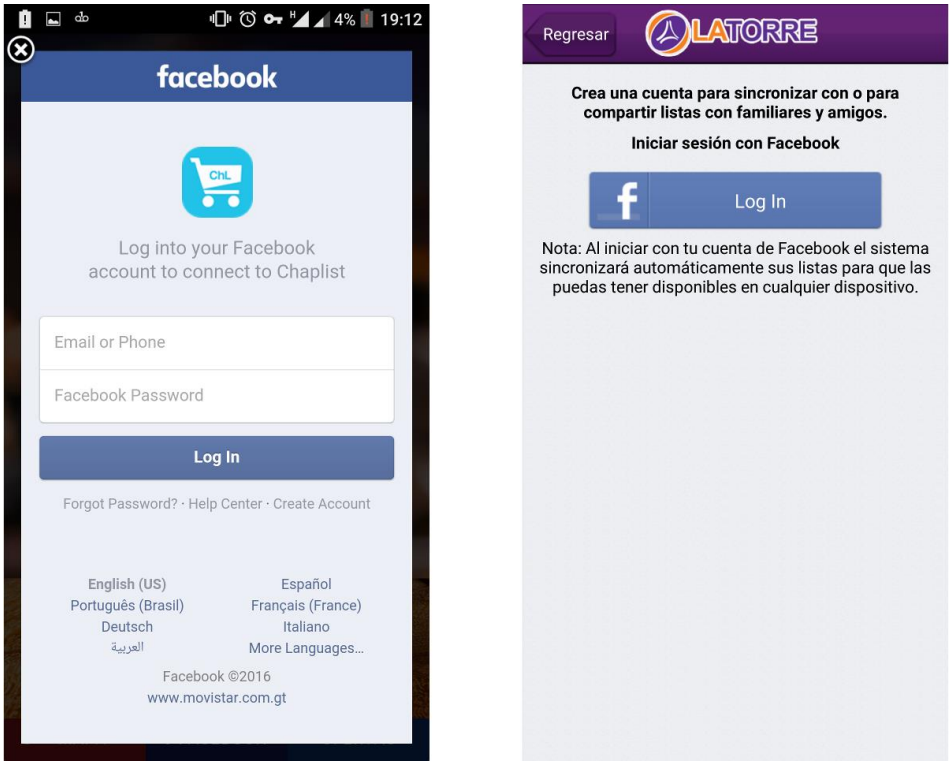


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.1.3. Facebook

En este caso La Torre App posee integración con Facebook, sin embargo, tiene distinto propósito que el de ChapList, ya que La Torre App utiliza Facebook para guardar las listas de productos en diferentes dispositivos y estar sincronizados, en cambio ChapList utiliza la integración para compartir las ofertas con el círculo social del usuario.

Figura 14. **Facebook ChapList La Torre**

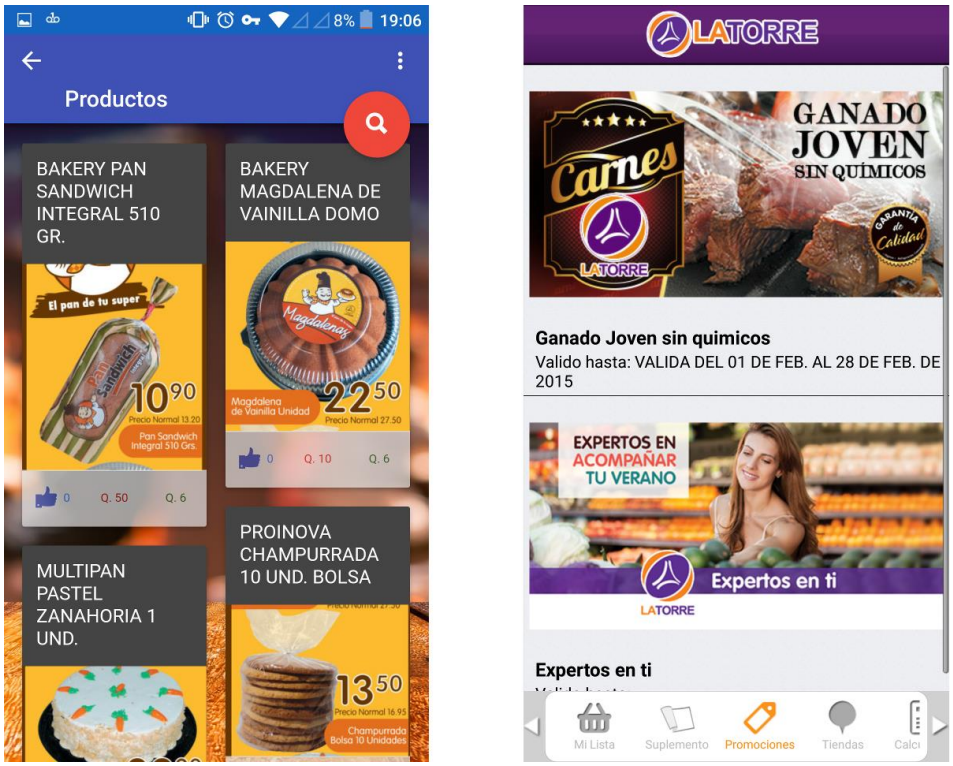


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.1.4. Ofertas

El objetivo de ChapList es mostrar las ofertas, mientras que La Torre App es hacer listas de productos. Sin embargo, La Torre App posee una sección de ofertas donde los usuarios pueden consultar las ofertas de ellos. La sección de ofertas de la torre es más genérica que la de ChapList, por el mismo hecho de tener distintos objetivos.

Figura 15. **Ofertas ChapList La Torre**

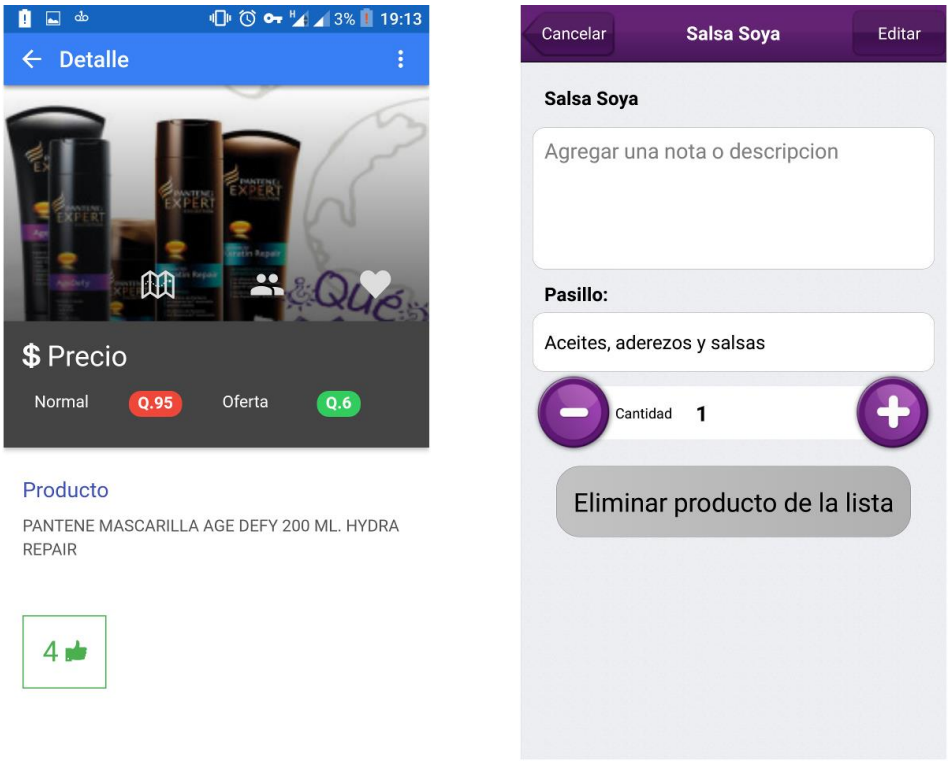


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.1.5. Detalle productos

Al dirigirse al detalle de los productos se puede observar que ChapList tiene un mejor aspecto visual y de acceso, ya que desde esta pantalla se puede acceder a los mapas, compartir y agregar el producto a favoritos. Por su parte, La Torre App tiene algunos campos para completar la información del producto.

Figura 16. **Detalle ChapList La Torre**

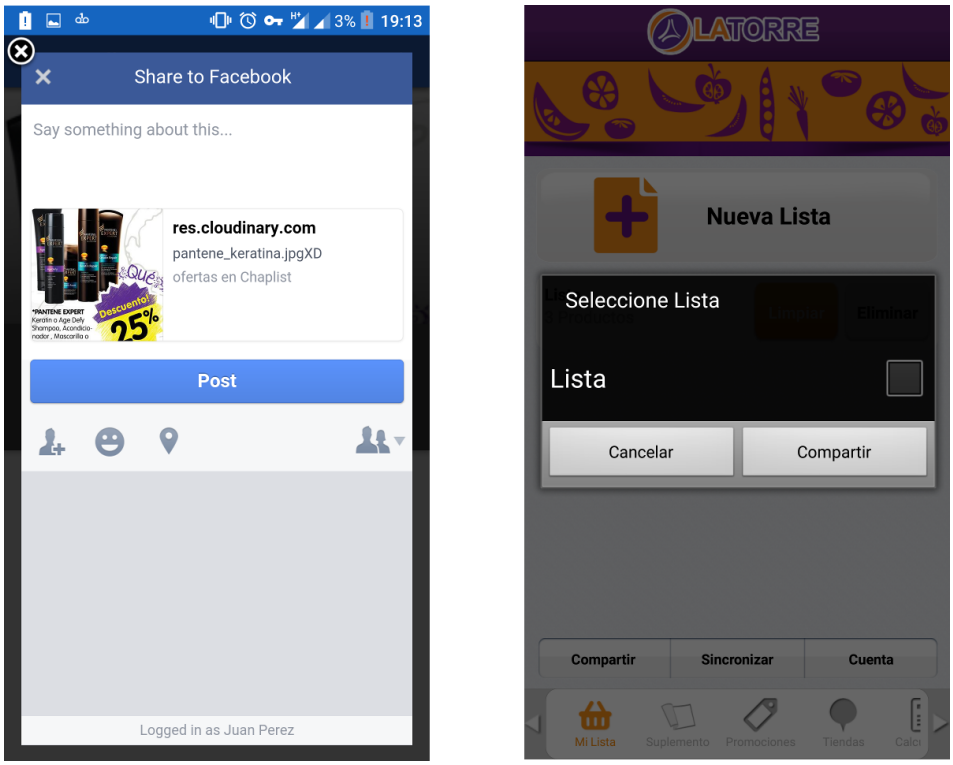


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.1.6. Compartir

La funcionalidad de compartir de las dos Apps es muy distinta, ya que La Torre App comparte listas de productos vía correo electrónico, mientras que ChapList puede compartir ofertas por medio de Facebook.

Figura 17. **Compartir ChapList La Torre**



Fuente: elaboración propia, empleando programa Ionic *framework*.

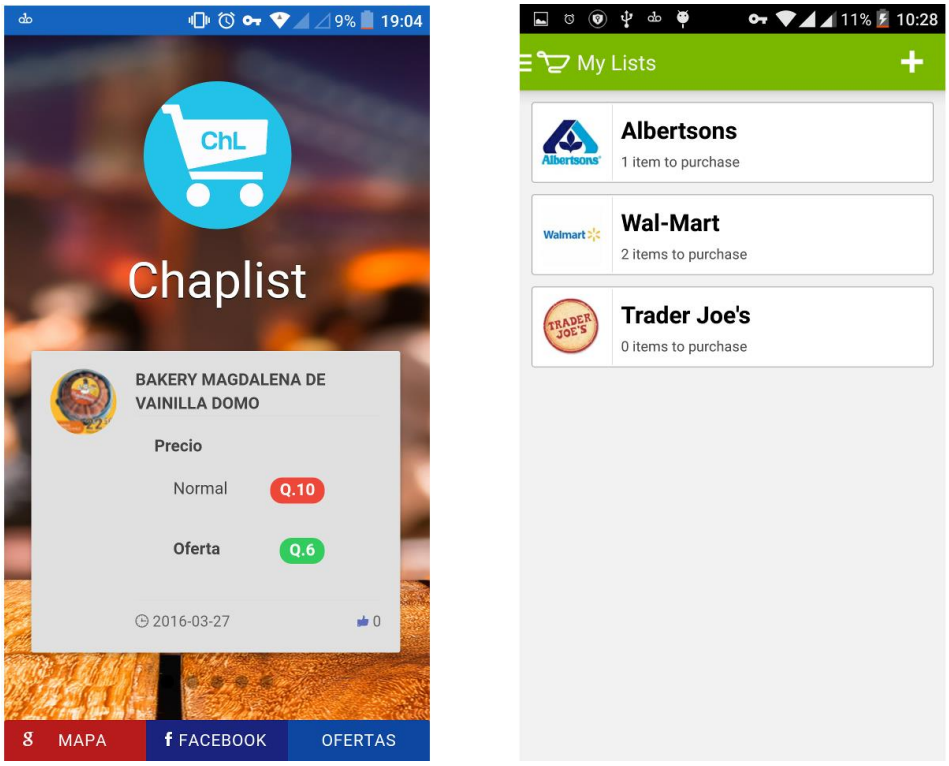
2.4.2. MyShopi

Es un app para realizar listas de supermercado. A continuación se comparará con ChapList.

2.4.2.1. Pantalla inicial

Myshopi en su pantalla inicial muestra las listas del supermercado realizadas por el usuario y las clasifica según el establecimiento que el usuario eligió para realizar su compra. Para acceder a las otras funcionalidades en Myshopi es necesario introducirse dentro de las listas, Por su parte, ChapList sigue teniendo ventajas en cuanto al acceso e información mostrada en la pantalla inicial.

Figura 18. **Pantalla inicio ChapList Myshopi**

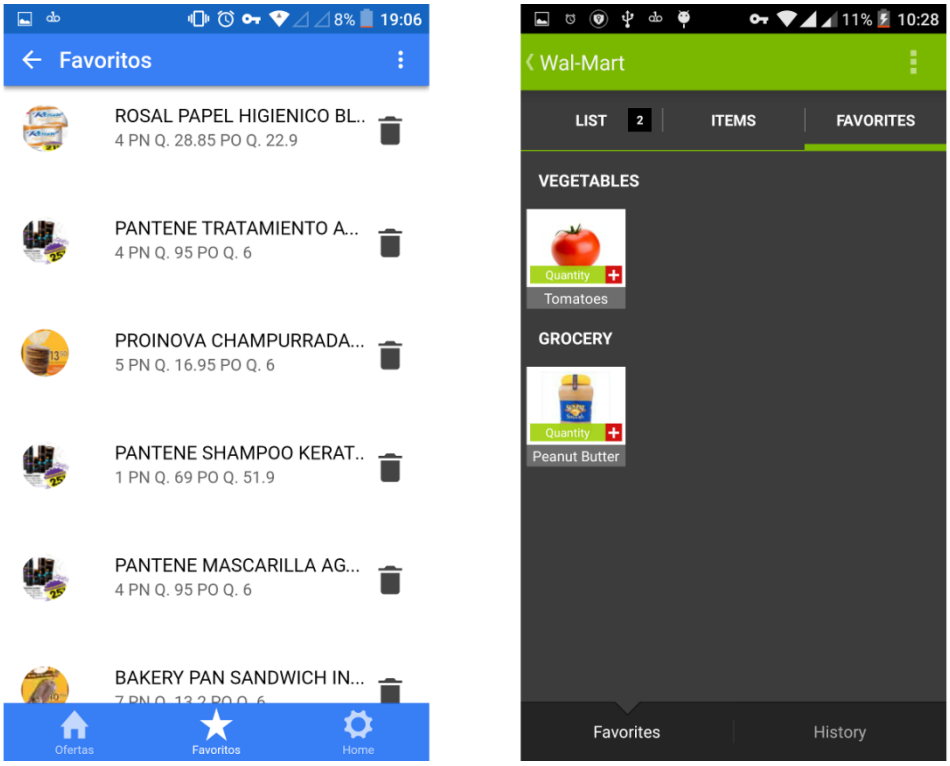


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.2.2. Favoritos

La sección de favoritos de MyShopi es donde se muestran los productos favoritos de los usuarios, dentro de la misma sección de favoritos tiene un historial. En cuanto a ChapList y su sección de favoritos se puede observar, en la figura 18, que muestra los precios como un detalle en los productos y estos están en forma de lista. Cuando se compara esta pantalla entre las dos Apps, se observa que el acceso a los favoritos es mucho más rápido en ChapList, sin embargo, el aspecto visual de Myshopi es agradable e intuitivo.

Figura 19. **Favoritos ChapList Myshopi**

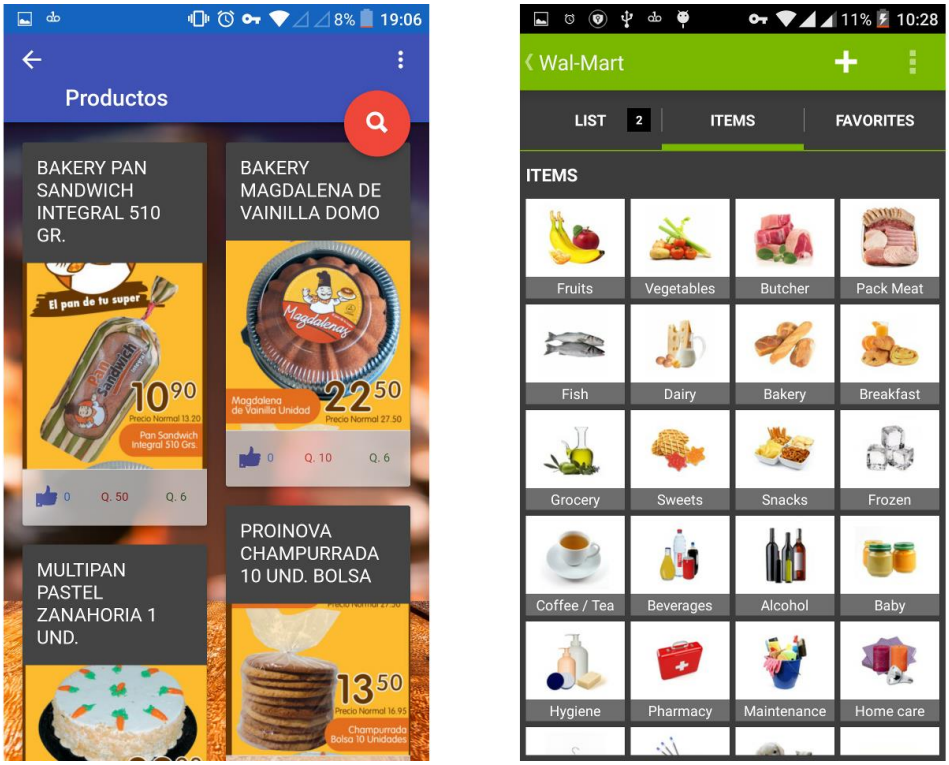


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.2.3. Productos

ChapList muestra los productos de una forma muy visual, abarcando información importante como precio normal, de oferta y cuántas personas marcaron la oferta como favorita. El objetivo es que el usuario pueda, mientras navega, percibir toda la información importante de los productos. Al comparar la sección de productos de Myshopi con ChapList se observa que Myshopi se enfoca en abarcar la mayor cantidad de productos en la pantalla, mientras ChapList, en la información de los productos. Las dos formas son muy agradables visualmente, sin embargo, difieren según el propósito.

Figura 20. **Productos ChapList Myshopi**

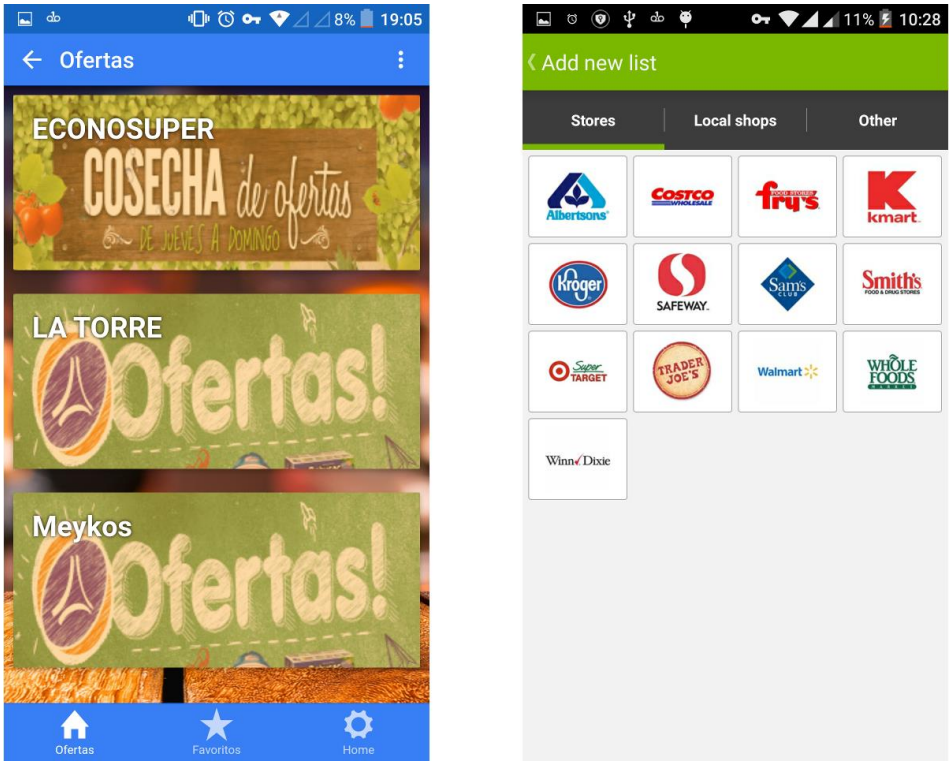


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.2.4. Supermercados

ChapList tienen una sección de establecimientos en la que cada uno ocupa bastante espacio en la pantalla con el fin de mostrar información relevante del establecimiento. Por su parte, Myshopi ocupa menos espacio por establecimiento, lo cual hace muy fácil de elegir el establecimiento preferido por el cliente.

Figura 21. **Supermercados ChapList Myshopi**

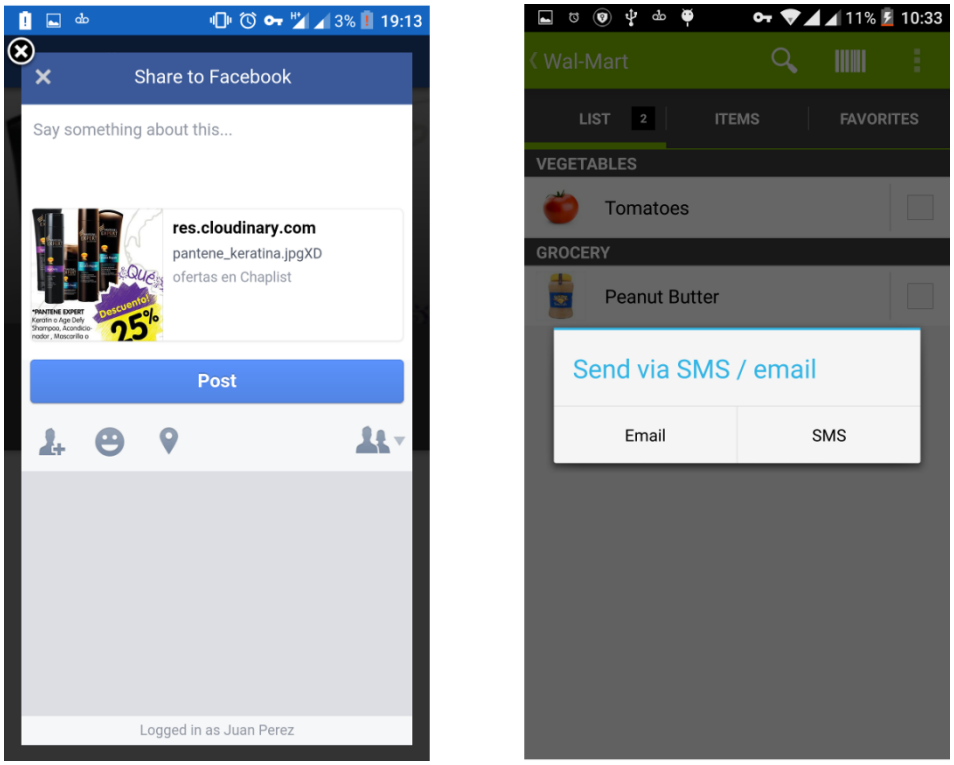


Fuente: elaboración propia, empleando programa Ionic *framework*.

2.4.2.5. Compartir

La funcionalidad de compartir en ChapList y Myshopi son diferentes, ya que Myshopi comparte lista de productos vía mensaje de texto o bien correo electrónico, mientras que ChapList comparte ofertas por medio de la integración con Facebook. La integración con Facebook le hace tener ventaja a ChapList, respecto a Myshopi en el aspecto social.

Figura 22. **Compartir ChapList Myshopi**



Fuente: elaboración propia, empleando programa Ionic *framework*.

3. DISEÑO DE LA APLICACIÓN BAJO LA NECESIDAD IDENTIFICADA

El diseño de la App es fundamental para su correcto funcionamiento y aceptación de parte de los usuarios. Se eligió cuidadosamente el diseño como su arquitectura debido a que el sistema debe quedar abierto a futuros cambios y mejoras que con el tiempo sean requeridas.

A continuación se explicará con detalle el diseño, utilidad y arquitectura de la App.

3.1. Prototipo

A continuación se describirá el prototipo de la aplicación.

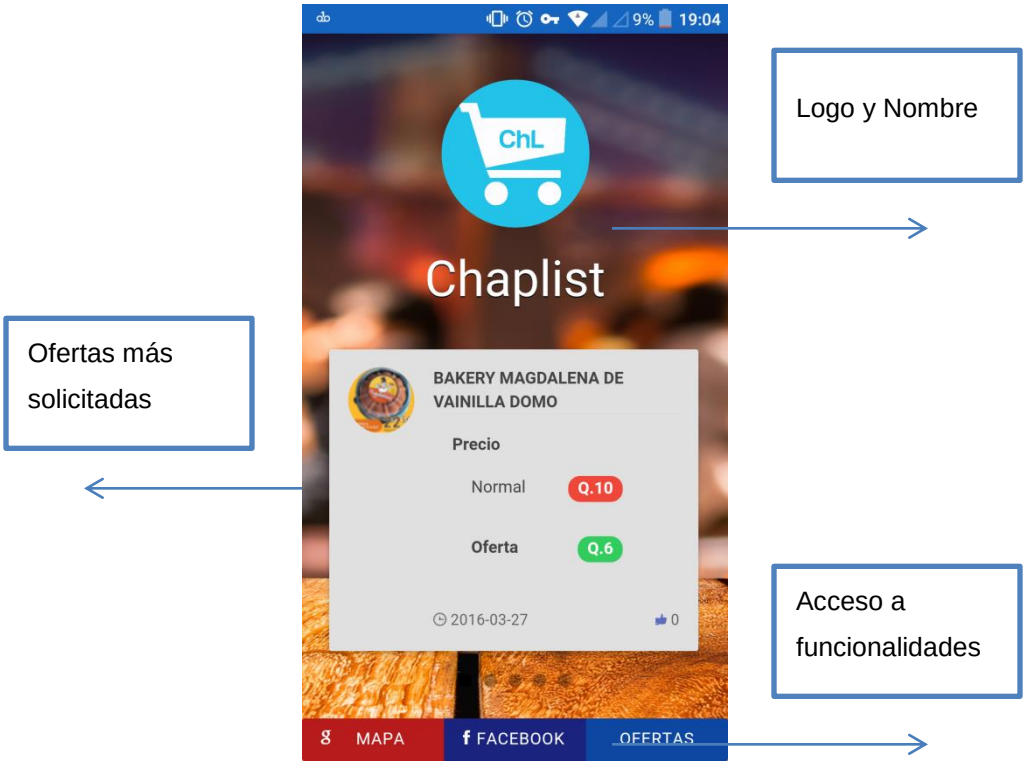
3.1.1. Pantalla inicio

La pantalla de inicio se divide en tres secciones:

- Primera sección**:** se encuentra el logotipo de la App y el nombre, para que en todo momento, el usuario se sienta identificado con la imagen de la misma.
- Segunda sección: consiste en las 5 ofertas más marcadas en los favoritos de los usuarios. Para navegar entre ofertas basta con deslizar la tarjeta de oferta de izquierda a derecha.

- Tercera sección: aquí hay tres botones para acceder a las funcionalidades principales de la App; ofertas, mapas y compartir vía Facebook.

Figura 23. **Pantalla inicio ChapList**



Fuente: elaboración propia, empleando programa Ionic *framework*.

3.1.2. Menú lateral

Este puede ser accedido mediante deslizar la pantalla desde el borde derecho hacia el borde izquierdo. En el menú se tiene acceso a las siguientes funcionalidades:

- Ofertas
- Favoritos
- Facebook
- Mapas

Figura 24. **Menú lateral ChapList**

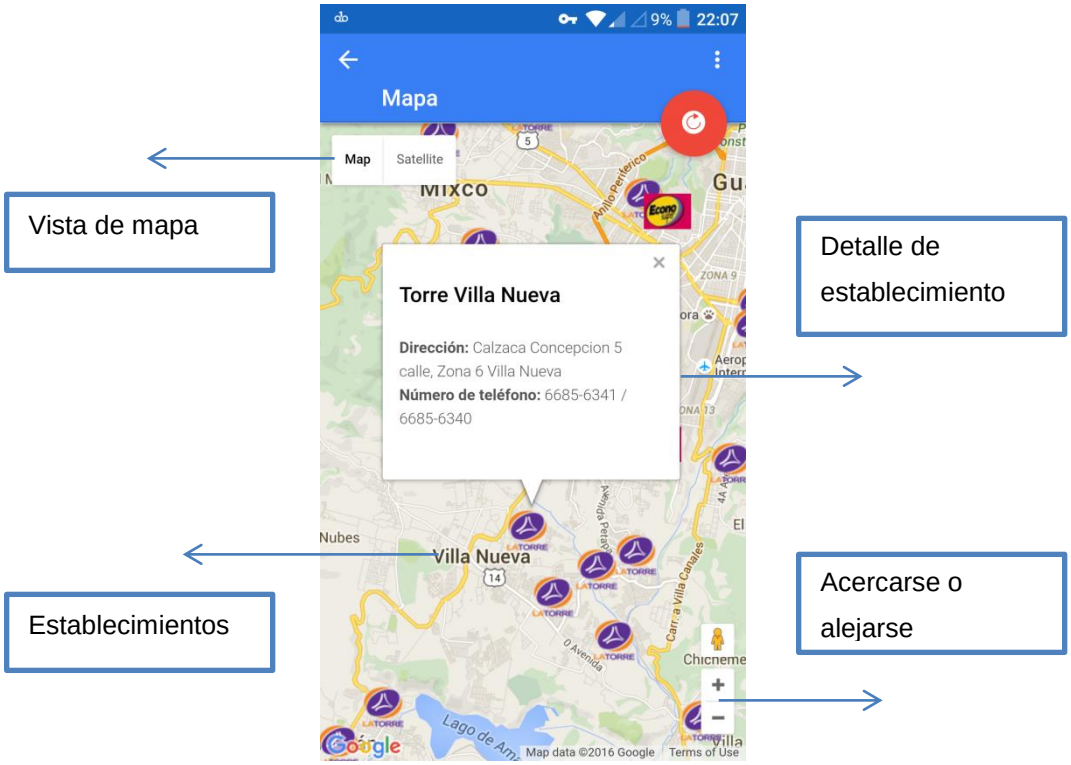
Fuente: elaboración propia, empleando programa Ionic *framework*.

3.1.3. Pantalla mapas

Esta pantalla muestra la localización geográfica de los establecimientos de ofertas. Al pulsar sobre el ícono de un establecimiento se muestra la información importante del mismo; dirección, teléfono y nombre. También está disponible la localización del usuario por medio del GPS y las funcionalidades

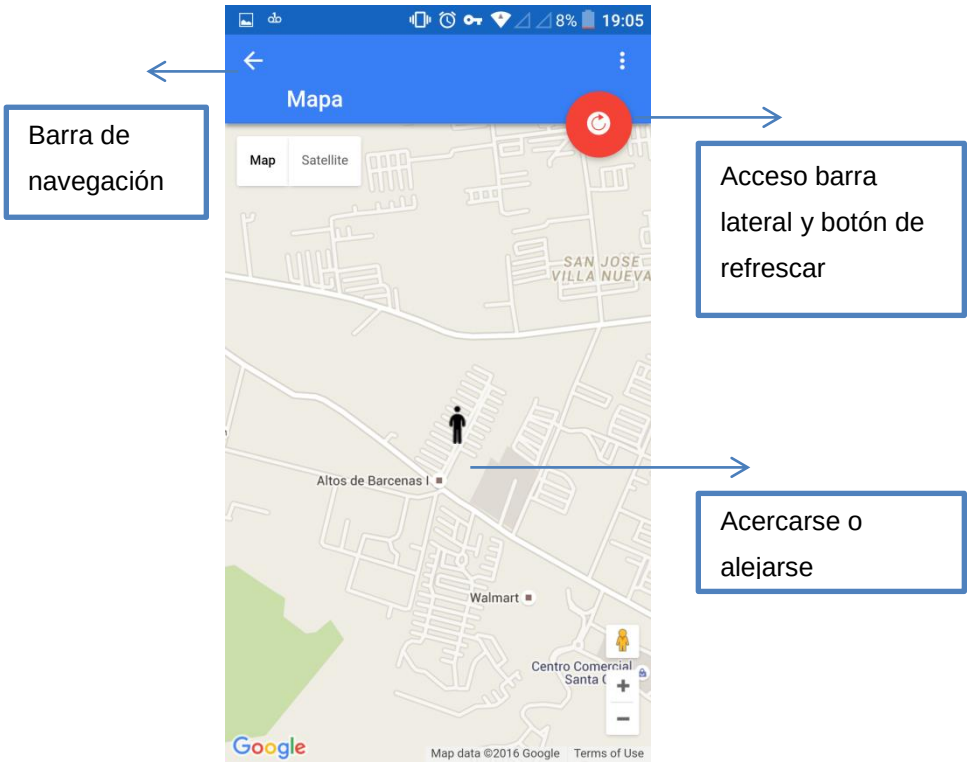
de Google para navegar en el mapa: vista normal, vista satélite, acercarse, alejarse, entre otras. En esta pantalla, también se puede acceder al menú lateral y, adicional a esto, hay una barra de navegación para regresar a la pantalla anterior.

Figura 25. **Pantalla mapas ChapList**



Fuente: elaboración propia, empleando programa Ionic *framework*.

Figura 26. **Pantalla mapas GPS ChapList**



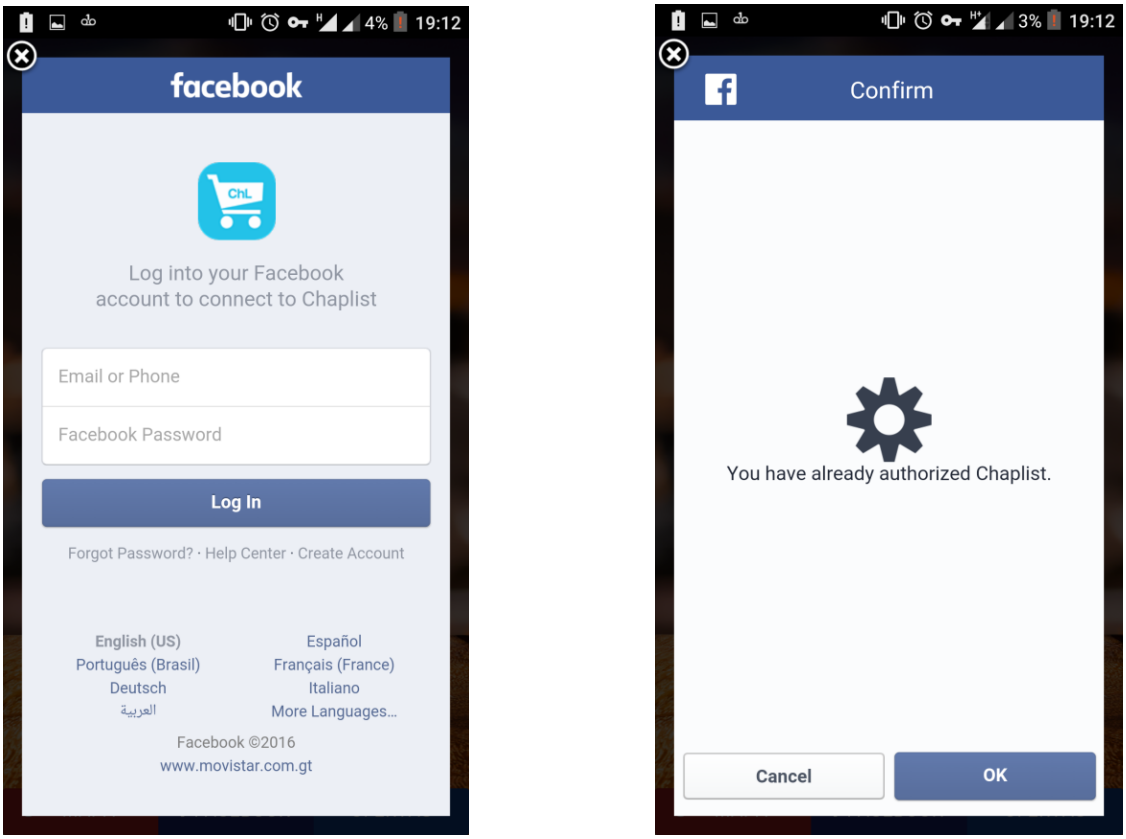
Fuente: elaboración propia, empleando programa Ionic *framework*.

3.1.4. Pantalla Facebook

Esta pantalla muestra el perfil del usuario registrado y la lista de amigos que utilizan la App, pero antes el usuario debe ingresar con su usuario y contraseña y autorizar a ChapList el acceso a su información. Desde la pantalla de perfil el usuario podrá postear sin necesidad de salirse de la App.

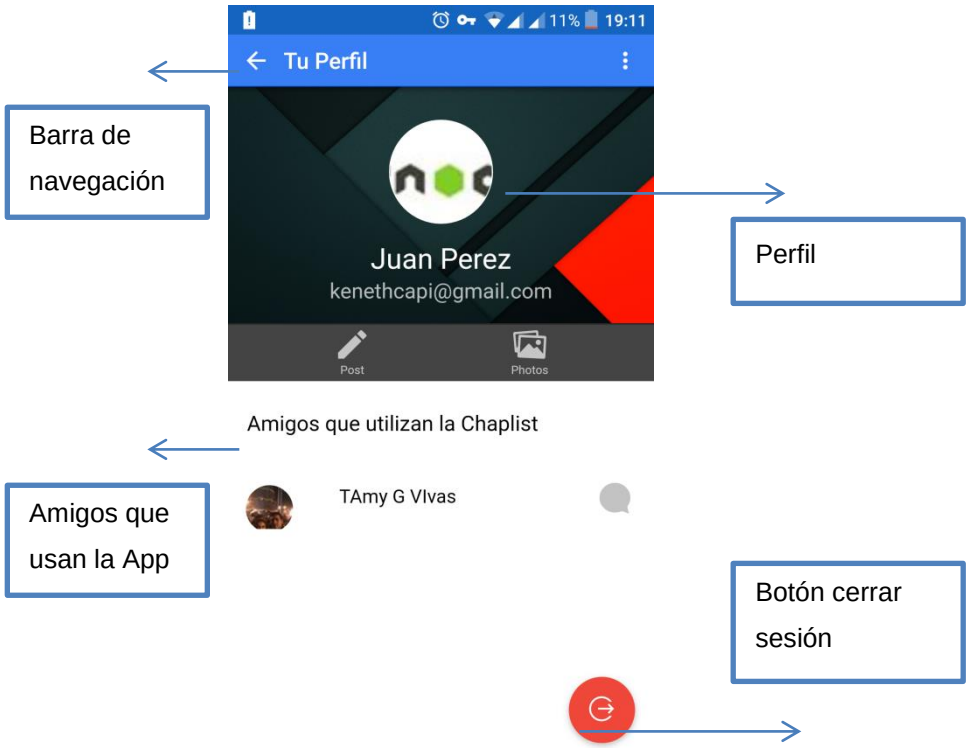
La pantalla posee su barra de navegación, acceso al menú lateral y un botón para cerrar la sesión.

Figura 27. **Ingreso y autorización Facebook**



Fuente: elaboración propia, empleando programa Ionic *framework*.

Figura 28. **Perfil Facebook ChapList**



Fuente: elaboración propia, empleando programa Ionic *framework*.

3.1.5. Pantalla ofertas

Se muestran lo establecimientos para que el usuario elija uno y se dirija a las ofertas de ese establecimiento. Además se implementa una barra en la parte inferior de la pantalla para acceder a la pantalla inicial, favoritos o quedarse en ofertas.

Figura 29. **Pantalla ofertas ChapList**

Fuente: elaboración propia, empleando programa Ionic *framework*.

3.1.6. Pantalla productos

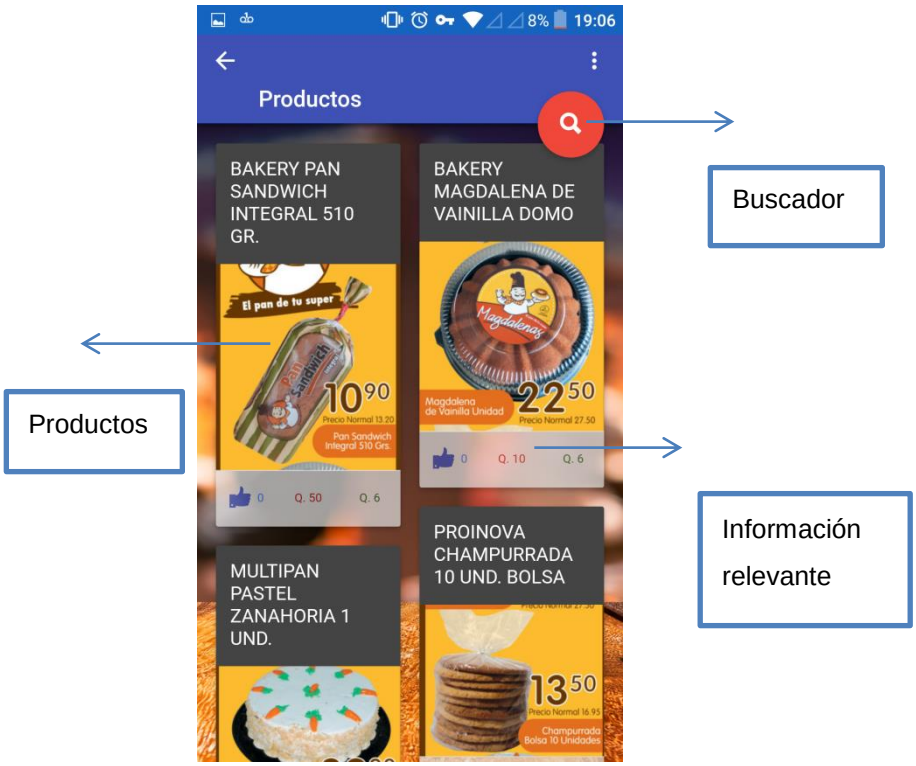
Una de las pantallas más importantes es la de productos, ya que en ella los usuarios pueden navegar y buscar sus productos de interés.

Los productos se muestran en dos columnas y la navegación es de arriba hacia abajo, esto permite paginar y así no cargar tanto al momento de abrir la pantalla. Así que los productos van mostrándose y cargándose conforme el usuario se desliza hacia debajo de la pantalla. Los productos se muestran en tarjetas que poseen la siguiente información:

- Precio normal
- Precio de oferta
- Número de usuario que marcaron el producto como favorito

Para complementar esta pantalla se implementó un buscador y así hacer más sencilla la búsqueda de los productos de interés.

Figura 30. **Pantalla productos ChapList**

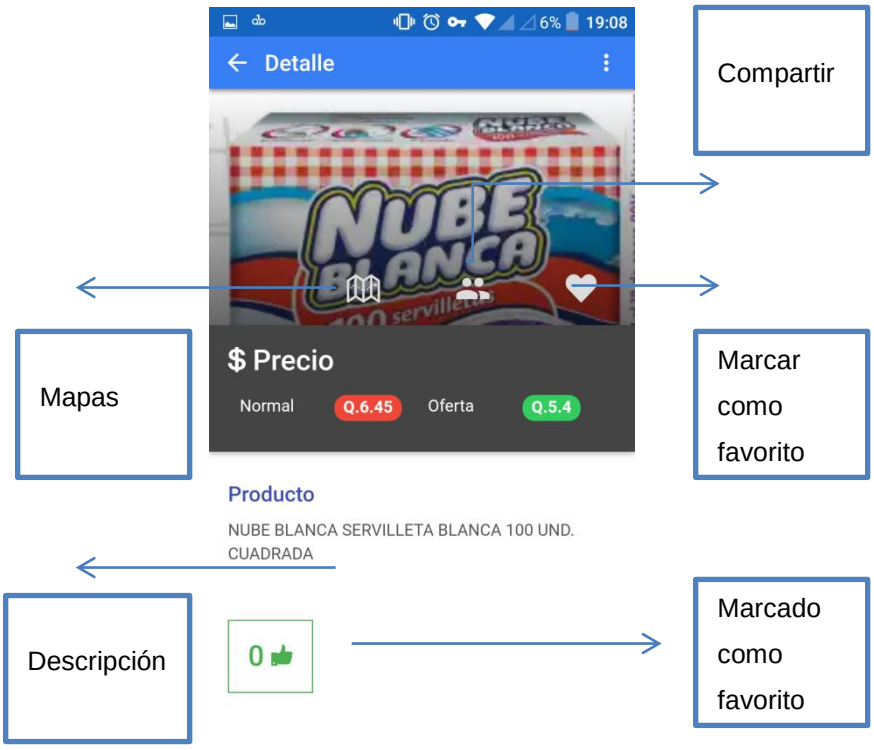


Fuente: elaboración propia, empleando programa Ionic *framework*.

3.1.7. Pantalla detalle

En la pantalla de detalle de productos se muestra la descripción del producto, su imagen y el número de usuarios que han marcado el producto como favorito. También posee acceso a mapas, compartir y marcar como favorito.

Figura 31. **Pantalla detalle ChapList**

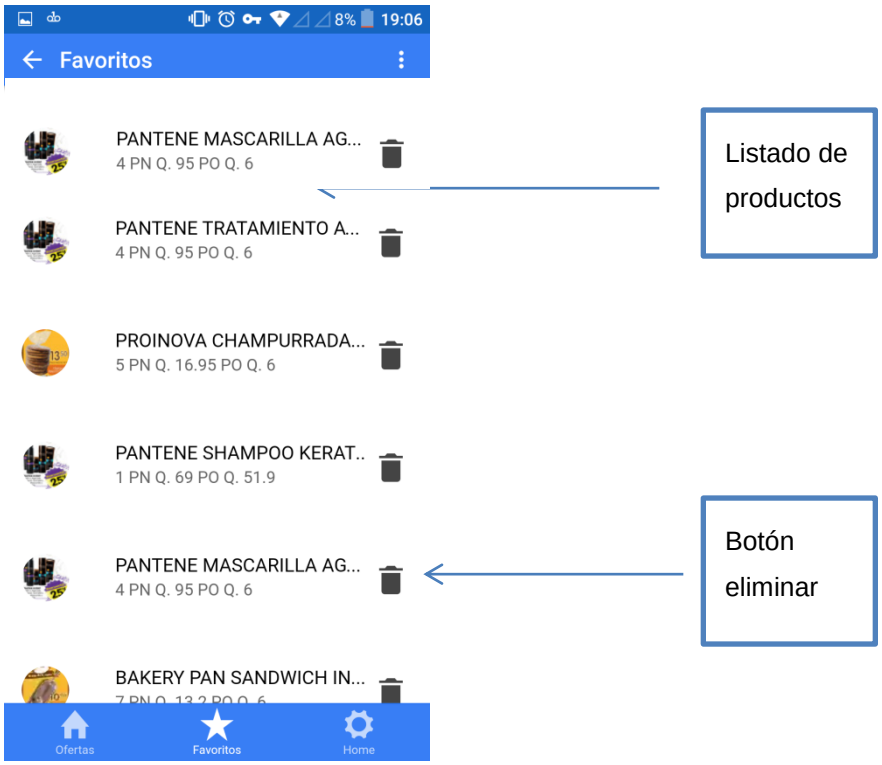


Fuente: elaboración propia, empleando programa Ionic *framework*.

3.1.8. Pantalla favoritos

Esta pantalla muestra los productos en forma de listado, la información relevante del producto y posee un botón para eliminar el producto del listado de favoritos.

Figura 32. **Pantalla favoritos ChapList**

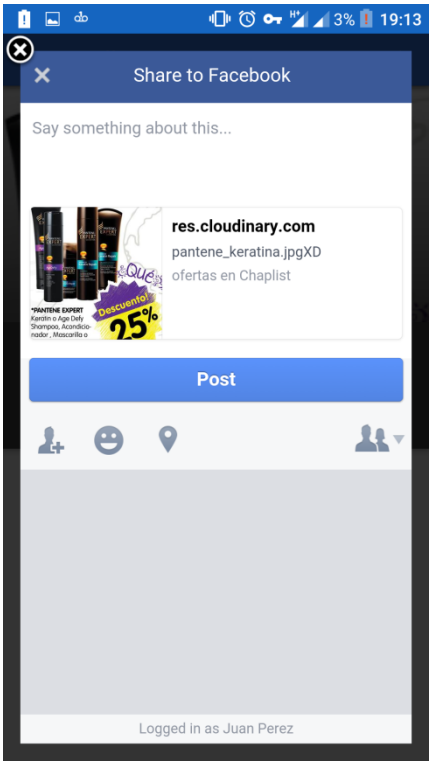


Fuente: elaboración propia, empleando programa Ionic *framework*.

3.1.9. Pantalla compartir

La pantalla de compartir va en conjunto con la integración con Facebook, ya que por medio de un post, es como la oferta puede ser compartida. Al pulsar el botón de compartir automáticamente la App toma la imagen del producto y la coloca en un post cuyo comentario el usuario lo debe redactar. El usuario puede elegir compartirlo con algún amigo que esté usando la App o bien publicarlo en su muro.

Figura 33. **Pantalla compartir ChapList**

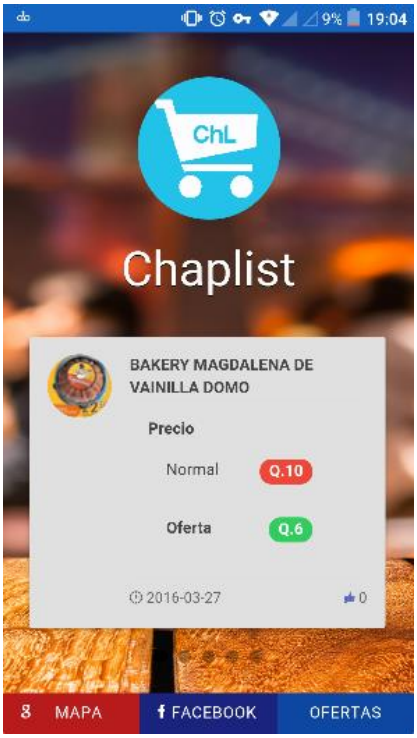


Fuente: elaboración propia, empleando programa Ionic *framework*.

3.2. Diseño intuitivo y utilidad

La aplicación móvil que se desarrolló cuenta con varias funcionalidades y atajos que redirigen a distintos contenidos que son mostrados a los usuarios.

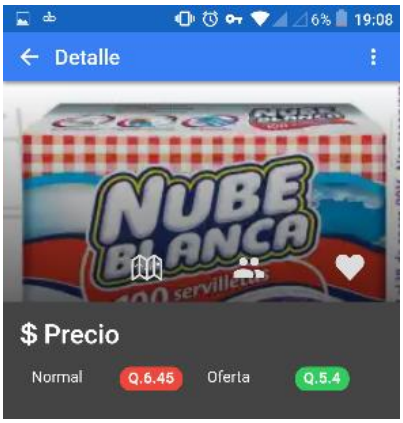
Figura 34. **Pantalla inicial**



Fuente: elaboración propia, empleando programa Ionic *framework*.

En la pantalla inicial se muestran las tres funciones principales que es el uso de Facebook de Google Maps, y las ofertas de los distintos supermercados. Al presionar sobre el listado de productos favoritos que se despliega en la pantalla principal se redirige directamente al contenido del producto como se describe en la figura 35.

Figura 35. **Detalle de producto**

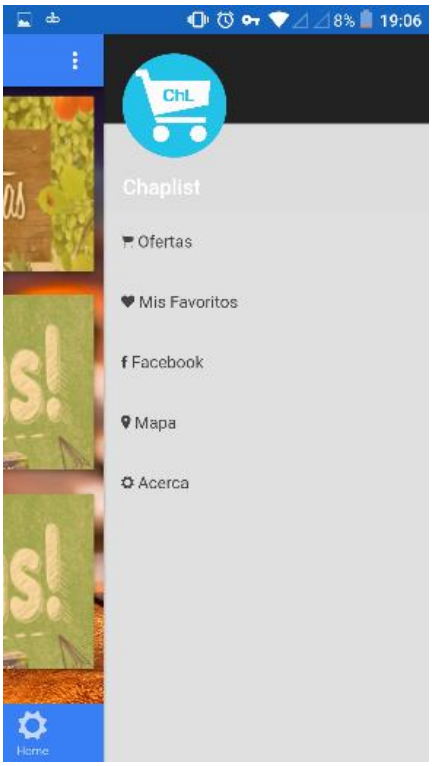


Fuente: elaboración propia, empleando programa Ionic *framework*

Cada ícono que se coloca sobre la imagen del producto permite hacer tres acciones diferentes, el mapa redirige automáticamente a la ubicación de las distintas tiendas, el de las dos personas permite compartir dicho producto vía Facebook con el listado de amigos que utilicen la aplicación de un usuario determinado, y por último, el corazón agrega dicho producto al listado de favoritos.

Cada pantalla cuenta con flechas de navegación, en la esquina superior izquierda para retornar a la pantalla anterior o bien cuenta con un submenú que aparece pulsando sobre los tres puntos ubicados en la esquina superior derecha o arrastrando la pantalla de derecha a izquierda (esta última opción no funciona en la pantalla de mapas).

Figura 36. **Submenú de opciones**



Fuente: elaboración propia, empleando programa Ionic *framework*.

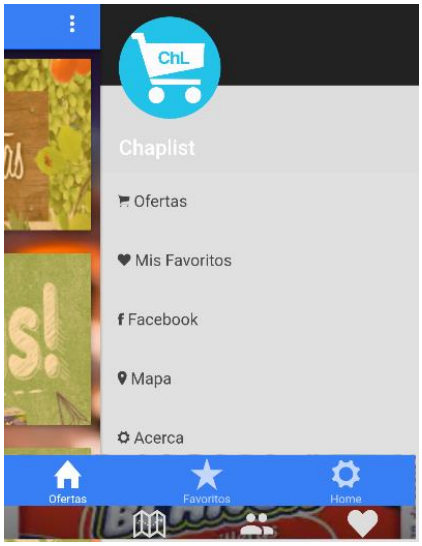
También se cuenta con la pantalla de ofertas que se compone de varias pestañas útiles al usuario para no moverse entre contenidos, sino solamente mostrar el que desea ver.

Figura 37. **Uso de pestañas para navegación de contenido**

Fuente: elaboración propia, empleando programa Ionic *framework*.

Cada botón o vínculo de las funcionalidades de la aplicación está representada, principalmente con un ícono que representa la funcionalidad que realiza o hacia qué tipo de contenido dirige, de esta forma se mejora la experiencia del usuario.

Figura 38. **Íconos representativos**



Fuente: elaboración propia, empleando programa Ionic *framework*.

Una parte importante es que en las pantallas de supermercados y ofertas se tiene la opción de *pull-down* que consiste en tirar de la pantalla hacia abajo que permite actualizar el contenido que se está mostrando.

También se muestran al usuario mensajes de advertencia o de guía cada vez que se realice una acción entre toda la aplicación.

3.3. Arquitectura del sistema

La arquitectura se realizó pensando en un sistema abierto a cambios y mejoras, debido a la problemática se espera que la App siga creciendo en cantidad de catálogos de productos ofertados que más establecimientos (no solamente supermercados) se muestren en la App.

Otro factor importante para elegir la arquitectura es el mantenimiento que se le debe dar a un sistema, y en este caso agregar más funcionalidades para que la App se mantenga actualizada y cubra la necesidad de los usuarios.

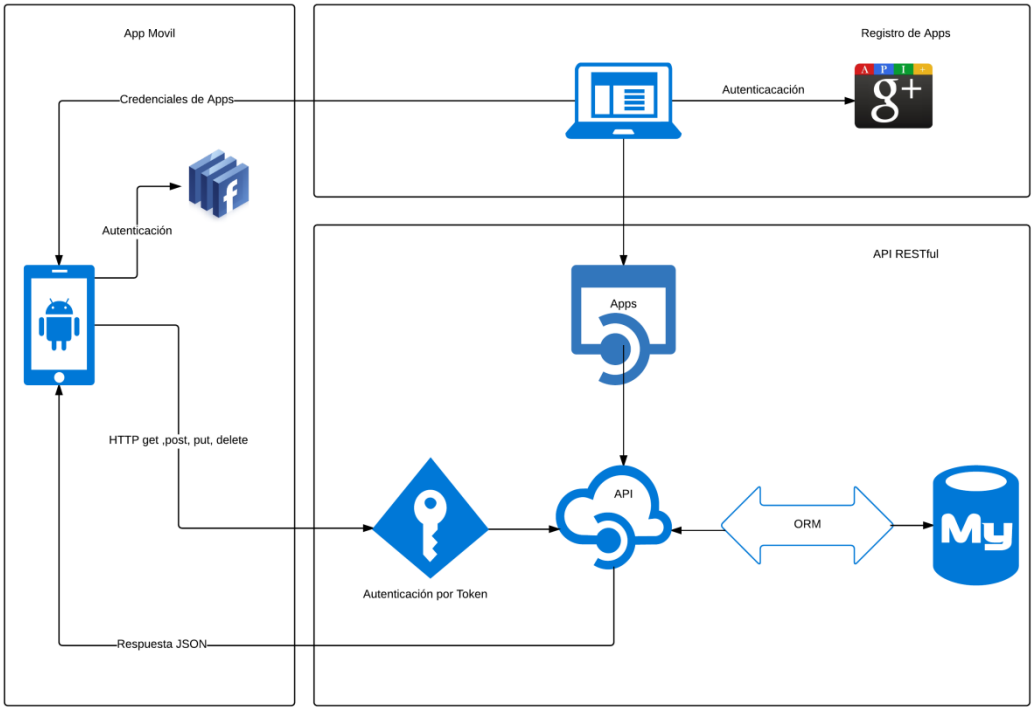
Hay que tomar muy en cuenta el número de usuarios que haga uso de la App, ya que la API debe suplir en todo momento la información de todas las peticiones realizadas, por lo que debe haber un buen manejo de concurrencia.

La arquitectura del sistema consiste en tres grandes subsistemas:

- Aplicación móvil
- API RESTful
- Plataforma de registro de Apps

En la figura 39 se muestra la arquitectura del sistema completo.

Figura 39. **Arquitectura**



Fuente: elaboración propia, empleando programa Ionic *framework*.

Como se ilustra en la figura 39, cada uno de los subsistemas posee relación entre ellos.

La relación entre la App móvil y la API RESTful es mediante el protocolo HTTP utilizando la convención *REST* mencionada en el capítulo 1. Adicional al protocolo de comunicación entre la App y la API se utiliza seguridad de peticiones mediante el uso de un *token* que se construye a partir del UUID del teléfono móvil con cierto tiempo determinado para su expiración. Si el *token* coincide y no ha caducado, la API procede a autenticar la App, ya que para que la API responda a una App esta debe estar registrada en el área de registro de Apps y así validar su veracidad. Si la API no valida alguno de los anteriores

puntos, entonces la petición es rechazada respondiendo con el número de error, según sea el caso, de lo contrario la App retorna la información requerida, siempre indicando el estatus. La información se retorna en un JSON que contiene toda la información requerida incluyendo el estatus.

La relación entre la App móvil y el registro de Apps es simple, ya que solo consiste en indicar las credenciales que se proporcionan al registrar la App en la en las peticiones que se realizan a la API.

La relación entre la API y el registro de Apps es similar al que tiene con la App móvil, pero más sencillo, ya que solo consiste en registrar la App por medio del *package name, hashkey* y nombre de la App. Una vez es registrada la App se proporciona un *secret key* que se utiliza para realizar las peticiones hacia la API. La comunicación entre el registro y la API se realiza mediante el protocolo HTTP y se utiliza autenticación por *token* para las peticiones.

3.3.1. API RESTful

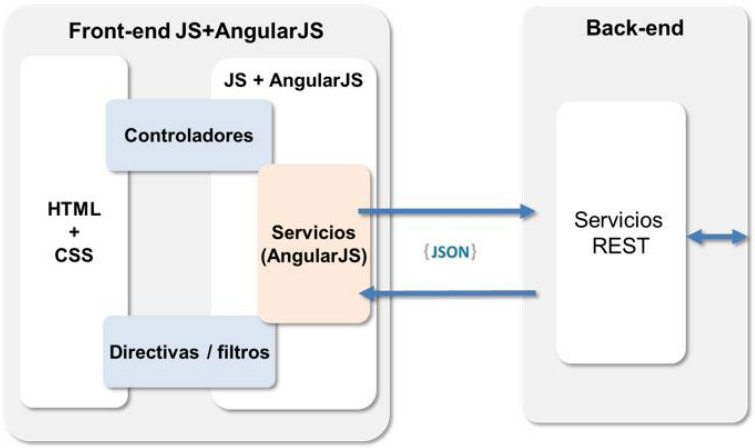
El eje central del sistema es la API RESTful, ya que toda la información es almacenada y provista por la misma. Tanto la plataforma de registro y App móvil realizan peticiones, por lo que es necesario una buena arquitectura y estructura y así atender cada una de las peticiones de manera óptima y segura. La API se realizó con el *framework* para APIs ActionHero, que facilita manejar versiones de la API, autenticación y bloqueo de protocolos.

El *framework* elegido es específicamente para desarrollar APIs, por lo que su estructura hace que el manejo de las mismas sea óptimo. También facilita el manejo de concurrencia al implementar un clúster de servidores y balancear la carga de peticiones.

3.3.2. Plataforma de registro

En la figura 40 se ilustra la arquitectura de la plataforma de registro.

Figura 40. **Arquitectura plataforma de registro**



Fuente*: Arquitectura angularJS*. http://blog.gfi.es/super-heroes-en-la-web-con-angularJS/. Consulta: abril de 2016.

4. DOCUMENTACIÓN BASE PARA EL DESARROLLO DE LA APLICACIÓN

4.1. Herramientas

A continuación describen las distintas herramientas tanto de software y hardware utilizadas en el desarrollo de la aplicación.

4.1.1. ActionHero

Es un *framework* para desarrollo de APIs es muy fácil de utilizar, permite crear proyectos escalables, además provee el uso de *sockets* TCP y *sockets* web.

Es capaz de procesar peticiones y tareas (*tasks*) en paralelo, lo que permite ejecutar ciertas acciones en segundo plano, como el envío de emails. Puede funcionar individualmente o en clúster de servidores, dependiendo de la carga de clientes que tenga la API. No construye APIs que sean esencialmente RESTful debido a que utiliza también *sockets*, pero se puede trabajar bajo este esquema si se requiere.

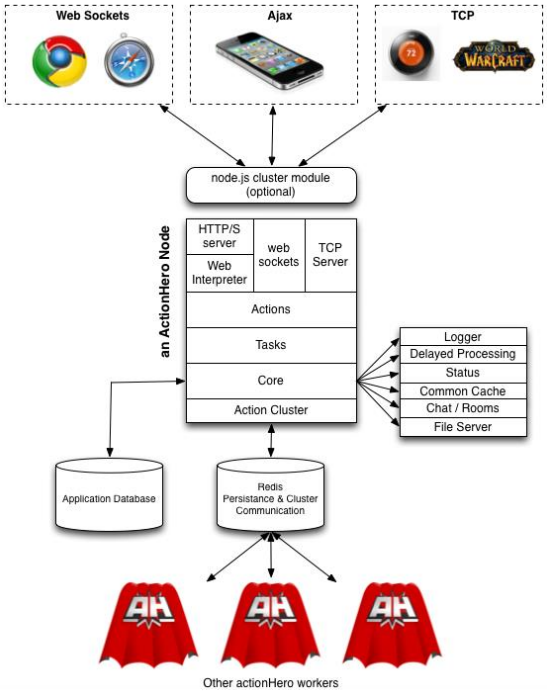
4.1.1.1. Requerimientos

- Node JS versión 4.0.0 o mayor
- NPM (Node Package Management)
- Redis (útil para configurar clústeres)
- Manejo de cache, entre otros. (Es opcional)

4.1.1.2. Composición de ActionHero

La forma en que se componen todas las partes que conforman el *framework* se detallan en la figura 41.

Figura 41. **Secciones de ActionHero**

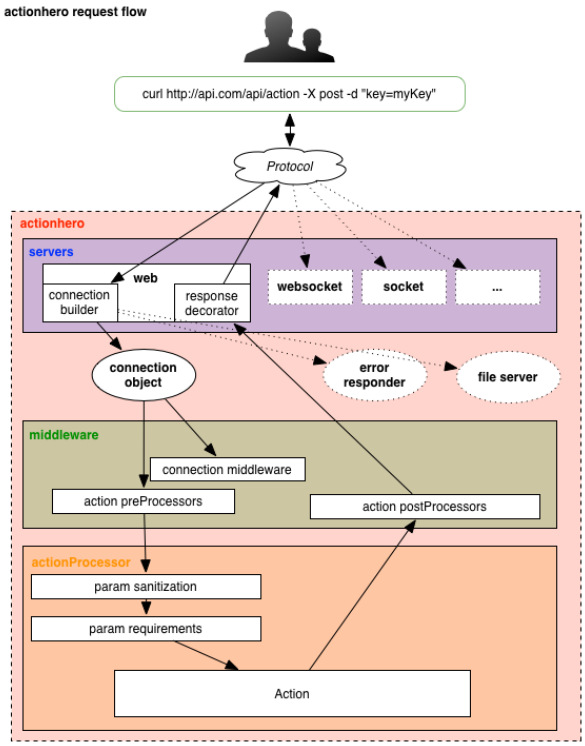


Fuente: *actionhero sections*. http://www.actionherojs.com/docs/#actionhero-sections. Consulta: marzo de 2016.

La comunicación que se tiene con los usuarios puede ser de tres tipos: como HTTP/S, como *sockets* web o como *sockets* TCP. Independientemente de la vía de comunicación todas las peticiones pasan por distintas capas. Action (acción) ejecuta ciertas instrucciones, más atrás se encuentran los Tasks que se ejecutan en segundo plano.

Si se requiere el uso de base de datos o la gestión de archivos, loggs, entre otros. ActionHero utiliza la capa Core. Finalmente, una de las principales funcionalidades del AngularJS es la implementación de clústers con varios servidores y para ello se utiliza la capa Action-clúster que utiliza una base Redis para mantener comunicación con otras APIs. Cabe destacar que ActionHero, al no definírsele una base de datos, utiliza Redis para el manejo de persistencia, pero si no se tiene instalado Redis, ActionHero utiliza Fake-Redis.

Figura 42. **Flujo de peticiones**



Fuente: *actionhero request flow*. http://www.actionherojs.com/docs/#request-flow. Consulta: marzo de 2016.

En las secciones del *framework* se puede observar que la capa de los Action es la que gestiona lo que los usuarios piden con las distintas acciones que realiza ActionHero, pero para llegar a esos Action cada petición pasa por varias etapas previas.

Cada petición (HTTP/S, socket web, socket TCP) pasa por un objeto de conexión que envía la petición a dos Middlewares que manejan internamente ActionHero (se pueden definir Middlewares personalizados e implementarlos en los Action), definidos como Pre-processors y Post-processors. Pre-processors para cuando se realiza una petición que ejecuta un Action y Post-processors para la respuesta a dicha petición.

Posteriormente, la petición ya pasa al *action-procesor* en donde el param-requirements verifica que los parámetros que se solicitan para ejecutar determinado Action estén definidos dentro de la petición (por ejemplo, una acción que solicite un Id de usuario para retornar el nombre del mismo), y finalmente se ejecuta el Action y empieza su rutina de respuesta hasta el usuario.

Para profundizar más en la estructura y funcionamiento de ActionHero se recomienda ver la documentación oficial.

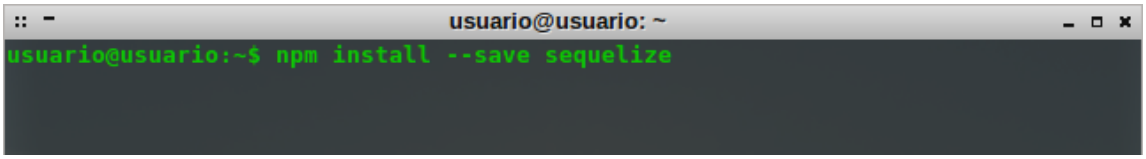
4.1.2. Sequelize

Es un ORM basado en promesas para NodeJS que soporta bases de datos como MySQL, PostgreSQL, MariaDB, SQLite y MSSQL. Es un ORM transaccional en el que se pueden utilizar relaciones y replicaciones entre otros.

El objetivo de utilizar Sequelize es manejar la capa de datos como objetos que se basan en modelos definidos por el desarrollador. La manipulación de los modelos se apega más al paradigma de programación que al de Bases de datos, por lo que facilita al desarrollador las consultas. Sin embargo, podría llegar a tener limitaciones cuando se desea realizar una consulta muy compleja. Sequelize, para mitigar esta problemática, también ofrece una sintaxis para realizar consultas en código SQL puro.

Sequelize es un paquete para NodeJS y está disponible en el gestor de paquetes NPM.

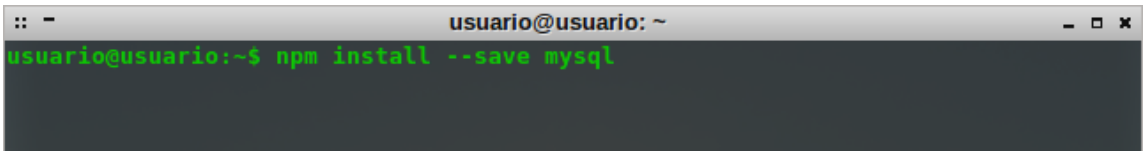
Figura 43. **Instalación Sequelize**



Fuente: elaboración propia, empleando programa de consola de Linux.

Una vez listo el ORM se debe instalar el gestor de base de datos, por ejemplo en este caso MySQL.

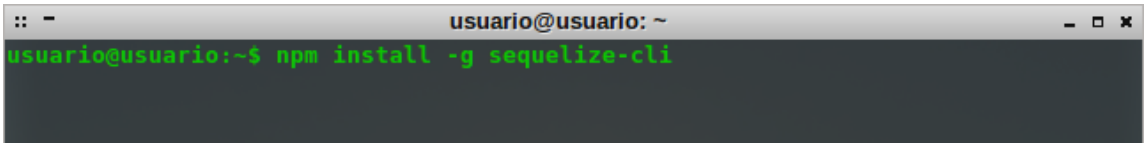
Figura 44. **Instalación MySQL**



Fuente: elaboración propia, empleando programa de consola de Linux.

Una vez instalado el ORM y el gestor de bases de datos queda un paso más para utilizar Sequelize y es instalar el CLI que se sirve para autogenerar código a partir de instrucciones predefinidas.

Figura 45. **Instalación CLI Sequelize**



Fuente: elaboración propia, empleando programa de consola de Linux.

Para utilizar el CLI antes de ejecutar un comando se debe colocar la palabra reservada, Sequelize.

En la tabla V se muestran los comandos disponibles en el CLI;

Tabla V. **CLI Sequelize**

Acción	**Comando**
Ejecutar migraciones pendientes	db:migrate
Actualizar tabla de migraciones	db:migrate:old_schema
Revertir la última migración	db:migrate:undo
Revertir todas las migraciones	db:migrate:undo:all
Correr los seeders	db:seed
Eliminar la información de la base de datos	db:seed:undo
Eliminar la información de la base de datos	db:seed:undo:all
Ayuda	Help
Inicializar el proyecto	Init
configuraciones	Init:config
Inicializar migraciones	init:migrations
Inicializar modelos	Init:models
Inicializar seeders	Init:seeders
Crear una nueva migración	migration:create
Crear un nuevo modelo	model:create
Crear un nuevo seed	seed:create
Ver la versión	version

Fuente: elaboración propia.

4.1.3. GIT

Es un control de versiones distribuido, de código abierto, utilizado para la gestión de cualquier tipo de proyecto y repositorios, es fácil de utilizar y tiene una CLI muy funcional

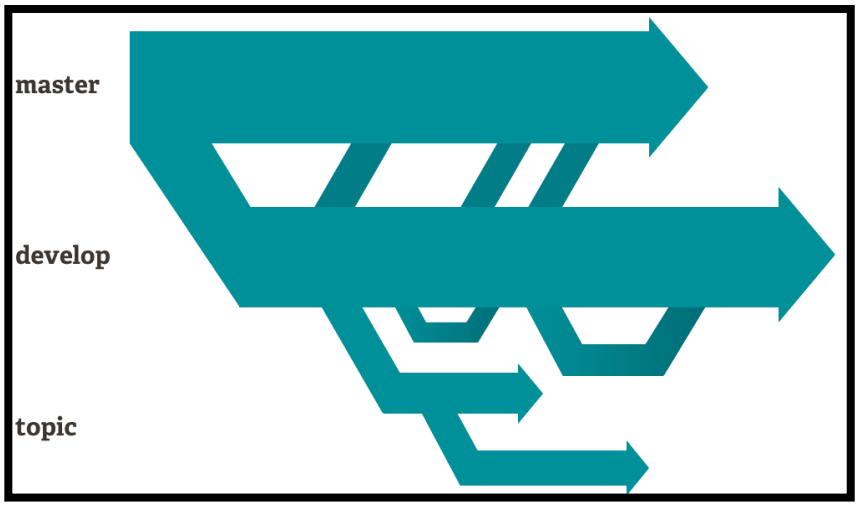
Es una herramienta muy útil de SCM (Source Code Management), ya que tiene funcionalidades como manejo de ramas para varias líneas de trabajo.

4.1.3.1. ***Branching* and *Merging***

Cuando se crea un repositorio utilizando GIT, este por defecto crea una rama principal llamada Máster, que es la que debería contener la última versión funcional del proyecto que se está desarrollando, pero si en un mismo proyecto

trabajan desarrolladores, testers, QA (Quality Assurance), entre otros, sería muy difícil manejar los cambios que se realizan a cada momento, es por esto que, con GIT se puede definir una rama o *branch* para cada tipo de involucrado con el fin de llevar un orden y no afectar el trabajo de los demás. A eso se le conoce como *branching*.

Figura 46. **Ejemplo de ramificación de un repositorio**



Fuente: *branching and merging*. https://git-scm.com/about. Consulta: marzo de 2016.

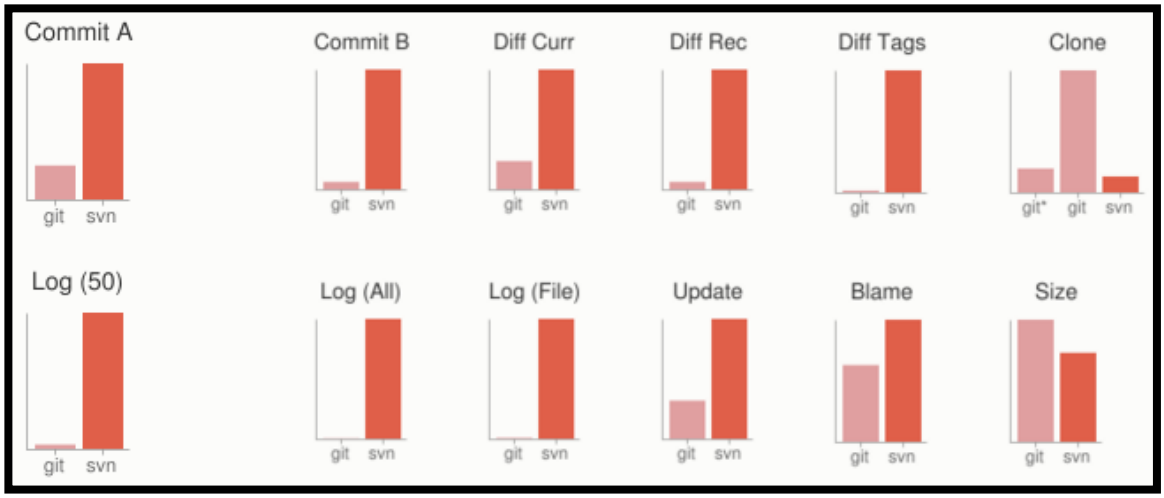
Cuando se requiere unificar el trabajo de todas las ramas se combinan o mezclan las ramas seleccionadas, a esto se le conoce como *merging*.

4.1.3.2. Comparación con otros sistemas de gestión de versiones

GIT es rápido y liviano, ya que fue construido para trabajar sobre un kernel de Linux (funciona también para Windows) y fue desarrollado en lenguaje C, por

lo que reduce mucho el tiempo de ejecución comparado con un leguaje de mucho más alto nivel (Java, C#, entre otros.).

Figura 47. **Performance de GIT contra SVN**



Fuente: *small and fast*. https://git-scm.com/about/small-and-fast. Consulta: marzo de 2016.

Figura 48. **Tiempo en segundos de GIT versus SVN**

Operation		Git	SVN	
Commit Files (A)	Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Commit Images (B)	Add, commit and push 1000 1k images	1.53	24.70	16x
Diff Current	Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff Recent	Diff against 4 commits back (269 changed/3609+,6898-)	0.25	3.99	16x
Diff Tags	Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1.17	83.57	71x
Log (50)	Log of the last 50 commits (19k of output)	0.01	0.38	31x
Log (All)	Log of all commits (26,056 commits - 9.4M of output)	0.52	169.20	325x
Log (File)	Log of the history of a single file (array.c - 483 revs)	0.60	82.84	138x
Update	Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Blame	Line annotation of a single file (array.c)	1.91	3.04	1x

Fuente: https://git-scm.com/about/small-and-fast. Consulta: marzo de 2016.

Se puede apreciar en la figuras 47 y 48 que GIT, debido a su sistema distribuido, es muy rápido comparado con Subversion (SVN propiedad de Apache).

Cabe resaltar que GIT ofrece integridad criptográfica a cada bit que está contenido en el proyecto. Debido a esto se garantiza que no se pueda cambiar nada del repositorio desde afuera, únicamente utilizando GIT.

4.1.3.3. Conceptos básicos y CLI

A continuación se detallan los comandos más utilizados en la terminal de usuario.

Tabla VI. **Conceptos básicos de un software de gestión de versiones**

Concepto	**Significado**
Commit	Persiste un conjunto de cambios realizados en un repositorio
Pull	Actualiza cambios hechos en un repositorio remoto a uno local.
Clone	Crea una copia de un repositorio determinada rama
Checkout	Permite a un usuario cambiarse entre ramas de un repositorio
Push	Permite enviar cambios realizados (commit) a un repositorio local hacia un repositorio remoto
Merge	Combina o mezcla el contenido de una rama a otra

Fuente: elaboración propia.

Por otra parte, la CLI que ofrece GIT es muy efectiva y se pueden listar sus comandos más básicos y funcionales.

Tabla VII. **Comandos básicos de GIT**

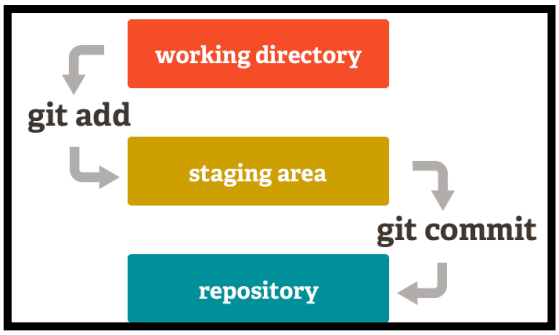
Acción	Comando
Iniciar un nuevo repositorio	$ git init
Agregar un archivo al repositorio	$ git add [nombre de archivo]
Agregar todos los archivos al repositorio	$ git add –A
Agregar archivos con cierta extensión	$ git add *.[extensión]
Realizar cambios en el repositorio	$ git commit –m ‘comentario’
Clonar repositorio	$ git clone [URL]
Actualizar repositorio	$ git pull
Ver información del repositorio	$ git status
Agregar repositorio remoto	$ git remote add origin [URL]
Realizar cambios en repositorio remoto	$ git push [nombre remoto] [nombre de rama]
Crear rama y cambiarse a la misma	$ git checkout –b [nombre de nueva rama]
Cambiar de rama	$ git checkout [nombre de rama]
Combinar rama con la rama actual	$ git merge [nombre de la rama]

Fuente: elaboración propia.

4.1.3.4. Área de ensayo

Una de las diferencias de GIT es que antes de persistir los cambios mediante un Commit, pasan a un área de ensayo o *staging* área en donde los Commits son formateados y pueden ser revisados antes de realizar dicho Commit.

Figura 49. **Flujo de acciones con GIT**



Fuente: *Staging Area*. https://git-scm.com/about/staging-area. Consulta: marzo de 2016.

4.1.4. Github

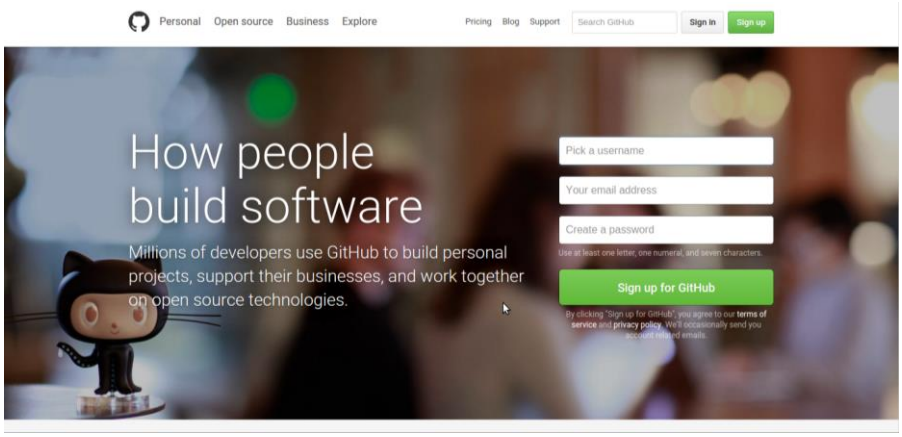
Es una plataforma que permite almacenar proyectos de desarrollo utilizando el control de versiones GIT. El código de los proyectos se almacena de forma pública o privada, dependiendo del tipo de cuenta que se esté utilizando.

Github proporciona herramientas para el desarrollo en equipo como: varios usuarios colaborando en un repositorio, ramas de un repositorio, wiki para el mantenimiento de versiones, sección de seguimiento para problemas, historial de commits con detalle de cambios en el código, entre otros.

Github almacena muchos proyectos de código abierto muy importantes para el desarrollo a nivel mundial como: Bootstrap, NodeJS, Jquery, entre otros.

Para hacer uso de Github hay que registrarse en su página web, como se muestra en la figura 50.

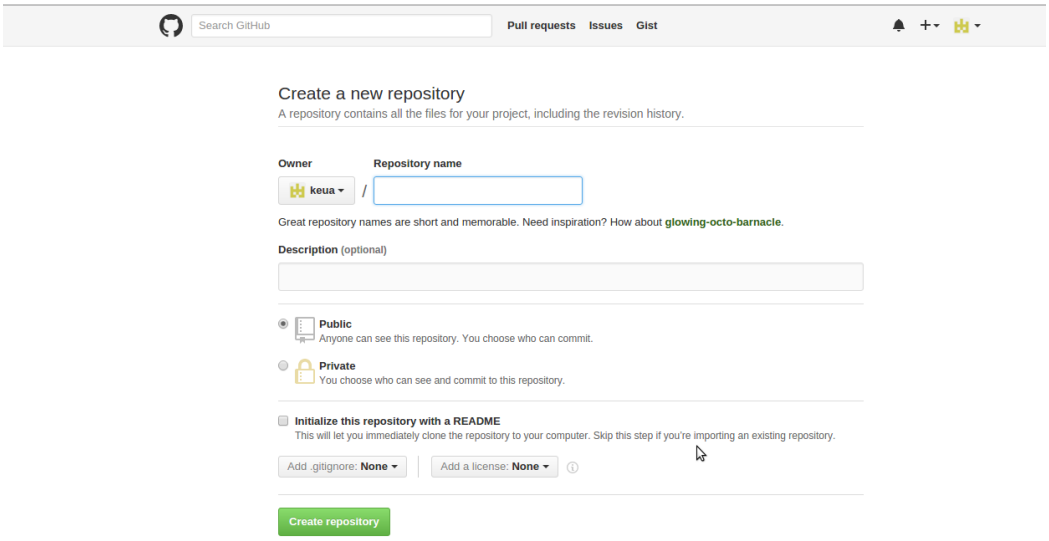
Figura 50. **Registro Github**



Fuente: elaboración propia, empleando página oficial de Github.

Una vez registrado se debe ingresar desde el mismo sitio web y ya se puede crear un nuevo repositorio, como se muestra en la figura 51.

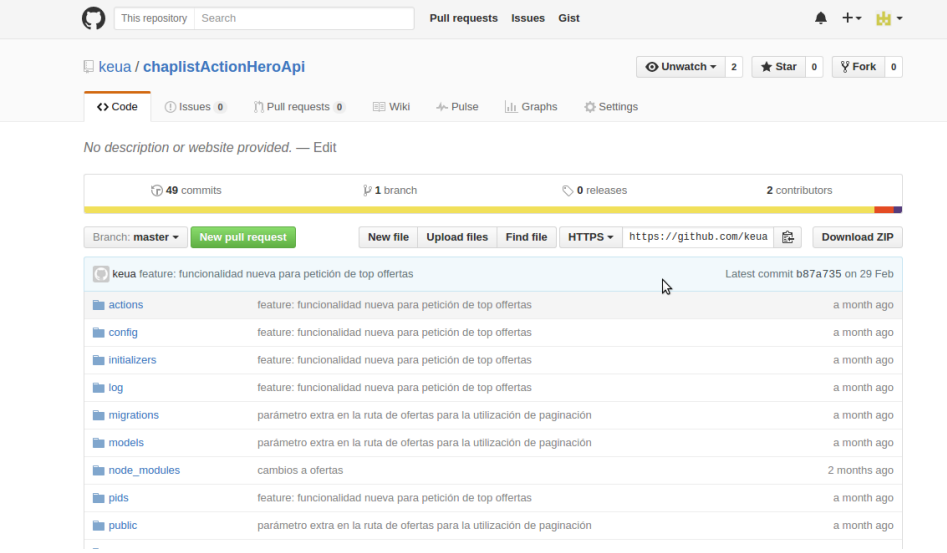
Figura 51. **Crear repositorio Github**



Fuente: elaboración propia, empleando página oficial de Github.

Una vez creado el repositorio ya se puede hacer uso del manejador de versión GIT desde una consola si se trata de Linux, el software que ofrece Github para hacerlo desde Windows. El repositorio puede tener varios colaboradores, se puede clonar, hacer nuevas ramas entre otros. Además, Github provee una serie de opciones como: estadísticas, sección de seguimiento de problemas, área de documentación, configuraciones entre otras. En la figura 52 se muestra como se ve un repositorio con sus opciones en la página oficial de Github.

Figura 52. **Repositorio en Github**



Fuente: elaboración propia.

4.1.5. *Brackets*

Es un editor de texto para desarrolladores, reconoce la sintaxis de muchos lenguajes como: php, JavaScript, html, java, python, entre otros. Posee vista previa y se puede integrar una serie de funcionalidades por medio de *plugins*. El editor es desarrollado por Adobe® y se puede encontrar su código fuente en Github.

Para el desarrollo de ChapList se utilizó este editor de texto tanto para la API como para la App. Utilizar este editor ayudó a la organización y agilidad de programación gracias a sus predicciones y *plugins*.

4.1.5.1. *Plugins*

Una parte fundamental del editor es la integración de nuevas funcionalidades ya que esto permite personalizar el desarrollo en función de la necesidad del desarrollador. Es importante que el desarrollador se sienta cómodo y que el aspecto visual del código sea agradable. *Brackets* presenta diversas opciones de temas y *plugin*; los que se utilizaron para el desarrollo son:

- AngularJS Code Hints

Este *plugin* contribuye a la agilidad de programación de AngularJS, ya que mientras el programador está escribiendo el código, le proporciona en una lista las opciones que más coincidan para no seguir escribiendo y ahorrar tiempo.

- *Beautify*

Este *plugin* contribuye a la organización del código, ya que su función es formatear el código, respetando los saltos de línea y tabulaciones. Con ello se tiene una mejor visualización del código en el momento de explorarlo y permite encontrar con mayor rapidez la porción de código buscada.

- *Brackets Git*

Para la integración del código es fundamental el manejo de versiones y este *plugin* es muy útil, ya que en el mismo editor se pueden ver los cambios que se han hecho con respecto de la última versión y pueden deshacerse. También puede hacerse commit del código para posteriormente hacer *push* a la rama máster, y actualizar el código local con un *pull request*. El único

requerimiento extra para instalarlo es, tener GIT y cada vez que se realiza un *pull* o *push* son requeridas las credenciales.

- *Emmet*

Este *plugin* sirve para agilizar la creación de código HTML, por medio de atajos y abreviaciones puede ser generado código repetitivo sin necesidad de perder tiempo en ello. Eso reduce el tiempo de desarrollo y permite que el desarrollador invierta ese tiempo en la lógica del sistema.

4.1.6. Apis de terceros

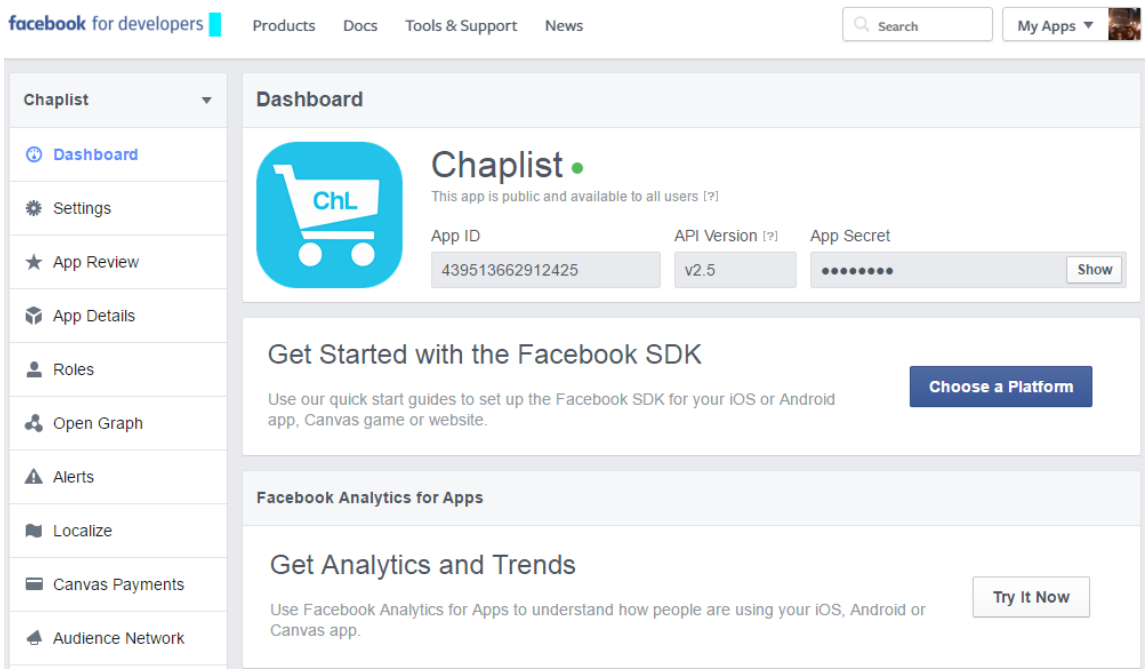
Existen distintos tipos de APIs para diferentes tipos de uso, dentro de las más utilizadas están las que ofrecen las redes sociales más populares.

4.1.6.1. API de Facebook

Facebook provee una serie de servicios y herramientas mediante su API para desarrolladores, actualmente está en su versión 2.5. Entre los que destacan:

- Facebook *login*
- *Sharing on* Facebook
- Facebook *social plugins*
- Facebook *Analytics for apps*
- *Mobile monetization*

Figura 53. **Panel de administración de Apps**



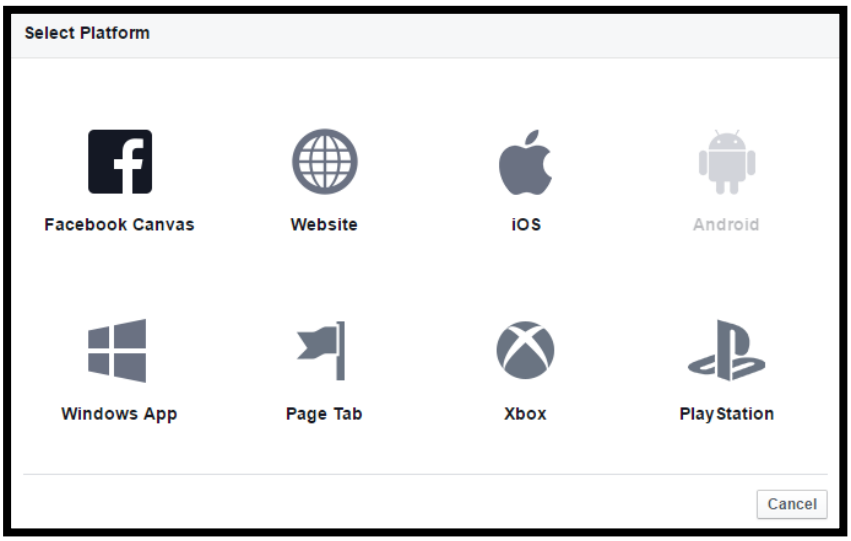
Fuente: elaboración propia, empleando información de Facebook *developers*.

Facebook provee una plataforma para la gestión de las Apps que un desarrollador crea, esto con el fin de analizar el uso de la misma por parte de los usuarios, como definir información que se le muestra al usuario cuando utiliza una App determinada.

- Plataformas que soporta la API de Facebook

Cuando se crea una App o se utiliza una ya existente, Facebook soporta una gran cantidad de plataformas para el uso de la API.

Figura 54. **Plataformas en API de Facebook**



Fuente: elaboración propia, con información de Facebook *developers*.

- Permisos de la API de Facebook

Cuando se crea una App, está por defecto tiene activados tres permisos para autenticación, que son:

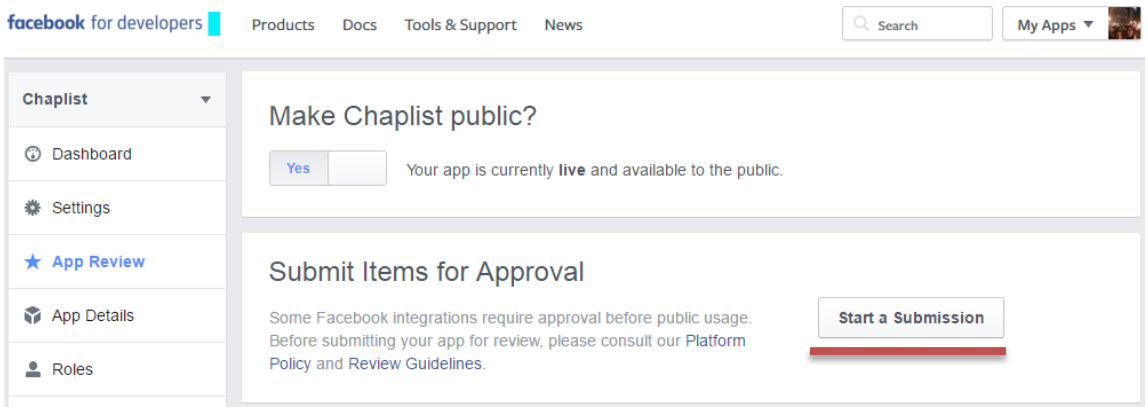
Tabla VIII. **Permisos por defecto de API de Facebook**

Permiso	**Acción que realiza**
Email	Provee la dirección de email del usuario que se loguea con Facebook utilizando determinada App de desarrollador
public_profile	Provee información básica como la foto de perfil, nombre y apellido de un determinado usuario.
user_friends	Provee un listado de personas que utilizan determinada App en común con el usuario que se está autenticando.

Fuente: elaboración propia.

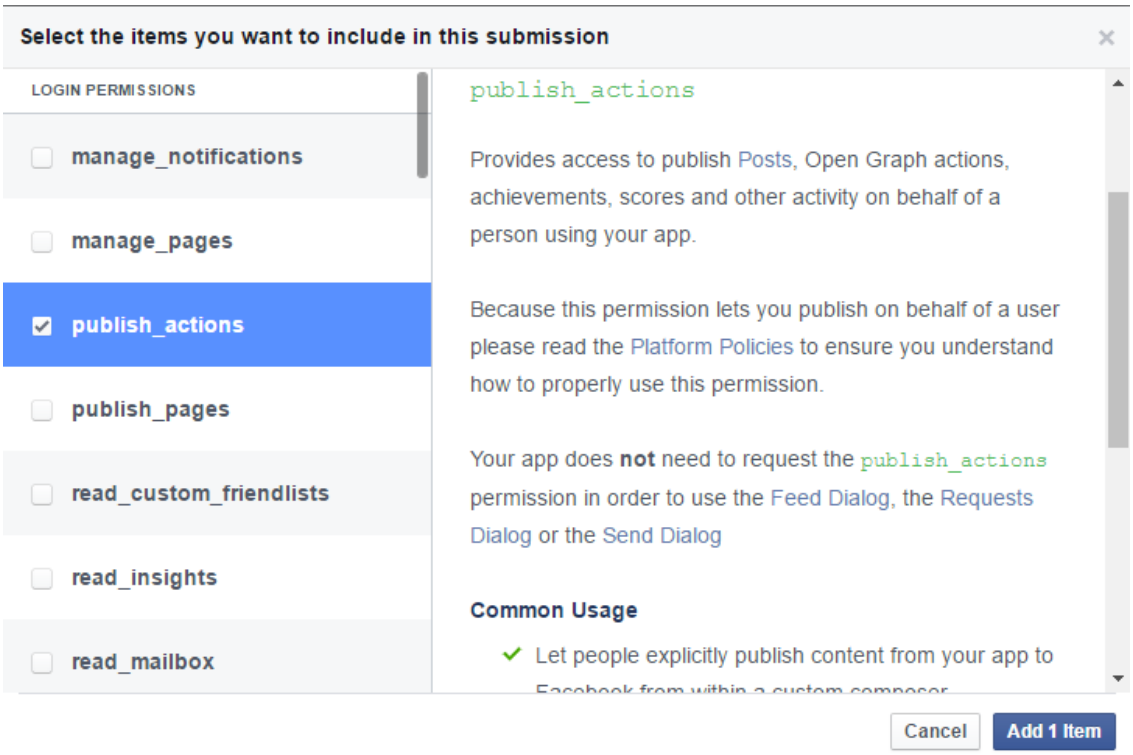
Si se llegara a requerir más permisos, como que la App pueda hacer publicaciones sobre el muro de un usuario u obtener un listado con las publicaciones que tiene un usuario en su perfil; por ejemplo, se tiene que elaborar un documento en el cual se hace una petición de los permisos a utilizar y debe ir acompañada por una política de privacidad, un video donde se muestra al usuario la forma en que afectará ese permiso a su información privada, *screenshots* de lo que se hará con dicho permiso. Toda la información debe ir en inglés y se advierte que en un máximo de siete días hábiles se aprueba o no la petición.

Figura 55. **Ubicación para petición de habilitación de permisos**



Fuente: elaboración propia, empleando información de Facebook *developers*.

Figura 56. **Listado de permisos a agregar en una petición**



Fuente: elaboración propia, empleando información de Facebook *developers*.

- Graph API

Es una API HTTP de bajo nivel, en donde se pueden hacer peticiones mediante consultas colocadas en la URL, permite muchas acciones como postear, subir fotos, entre otros.

Figura 57. **Ejemplo de una petición vía *Graph* API**

Fuente: elaboración propia, empleando información de Facebook *developers*.

La mayor parte de la funcionalidad de Graph necesita un *token* de acceso para procesar peticiones, de lo contrario se rechazan dichas peticiones, dicho *token* se obtiene cuando se autentica con Facebook.

La forma en que está estructurado se basa en los tres componentes principales de la misma:

- *Nodes*: son objetos como un usuario, una foto, un comentario, entre otros.
- *Edges*: son los vínculos entre los nodes, como los comentarios de una foto, las fotos de un usuario, entre otros.
- *Fields*: son atributos que hacen referencia a los nodes, como el nombre de un usuario, el email de un usuario, el nombre de una foto, entre otros.

La forma en cómo se construye una petición que se envía por medio la URL a *Graph* API se realiza como lo detalla la figura 58.

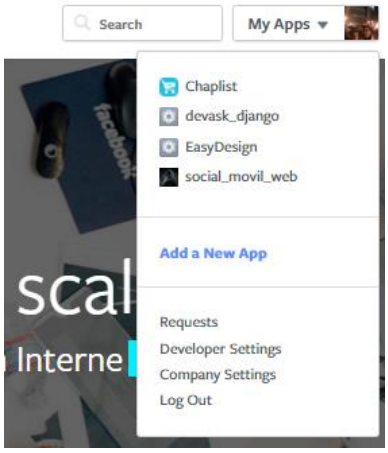
Figura 58. **Estructura de peticiones a *Graph* API**

Fuente: elaboración propia, empleando información de Facebook *developers*.

- Creación de una App en la API de Facebook

En la página principal de desarrolladores de Facebook (Facebook *developers*) se muestra una pequeña pestaña donde se pueden ver las Apps que se han creado, incluso permite crear nuevas.

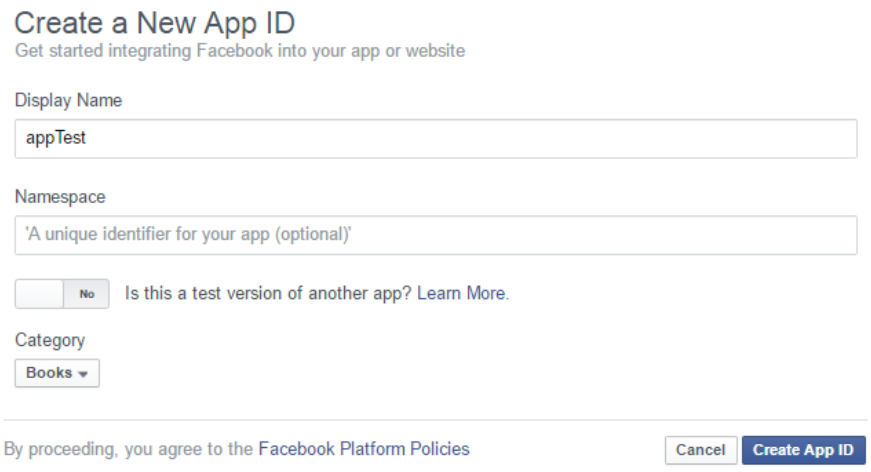
Figura 59. **Lista de aplicaciones existentes**



Fuente: elaboración propia, empleando información de Facebook *developers*.

Facebook genera un App ID para identificar cada App que se crea dentro de su API.

Figura 60. **Creación de App ID**



Fuente: elaboración propia, empleando información de Facebook *developers*.

Luego que se crea el App ID, Facebook provee una llave llamada App Secret, que es la que se utiliza mediante otros medios de autenticación para obtener un *token* válido y empezar a utilizar los servicios de la API.

Al agregar nuevas plataformas a la App, va a depender mucho de la que se elija, ya que si es una plataforma web se solicitarán las URLs mediante las cuales se puede utilizar la App, o si es plataforma Android, Facebook pedirá el nombre del APK, un *HashKey*, entre otros.

4.1.6.2. API de Google

De forma similar a Facebook, Google ofrece una mayor cantidad de servicios en la nube mediante distintas APIs. Dentro de las distintas APIs que nos ofrece Google se pueden mencionar: Drive API, Gmail API, Google Analytics API, Google + API, entre otros.

Específicamente, la API de Google+ es muy utilizada por sitios y aplicaciones móviles para que los usuarios puedan autenticarse a determinado sistema y utilizar la información de Google sobre un usuario.

Figura 61. **Servicios que ofrece Google+**

Services > Google+ API v1

Authorize requests using OAuth 2.0: OFF

plus.activities.get	Get an activity.
plus.activities.list	List all of the activities in the specified collection for a particular user.
plus.activities.search	Search public activities.
plus.comments.get	Get a comment.
plus.comments.list	List all of the comments for an activity.
plus.people.get	Get a person's profile. If your app uses scope https://www.googleapis.com/auth/plus.login, this method is guaranteed to return ageRange and language.
plus.people.list	List all of the people in the specified collection.
plus.people.listByActivity	List all of the people in the specified collection for a particular activity.
plus.people.search	Search all public profiles.

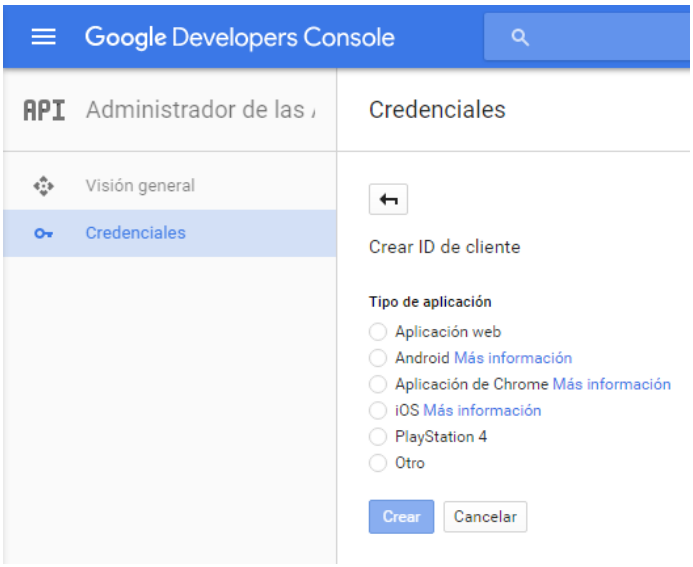
Fuente: elaboración propia, empleando información de Google *developers*.

En estas distintas APIs, los desarrolladores de igual forma crean Apps para utilizar, por ejemplo, Google+. Todo esto se realiza en la consola de desarrolladores de Google que tiene un panel de administración para las distintas Apps que un desarrollador utilice.

- Plataformas que soporta la API de Google

Google al igual que Facebook, tiene soporte para varias plataformas en las cuales se pretende desarrollar.

Figura 62. **Plataformas en API de Google**



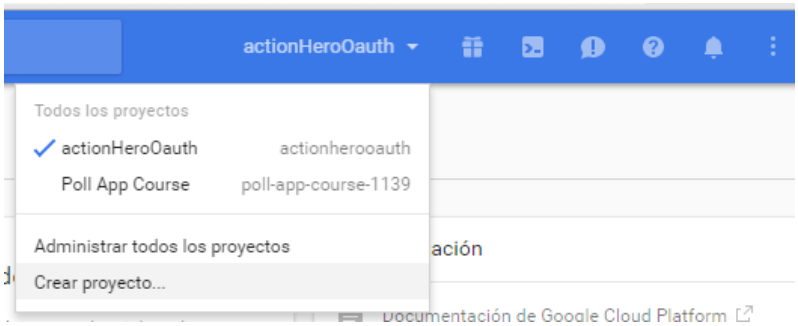
Fuente: elaboración propia, empleando información de Google *developers*.

La cantidad de información y de documentación que ofrece Google para sus distintas APIs es muy extensa y funcional, por lo que se puede aprender mucho sobre una API determinada directamente desde su documentación.

- Creación de una App en la API de Google

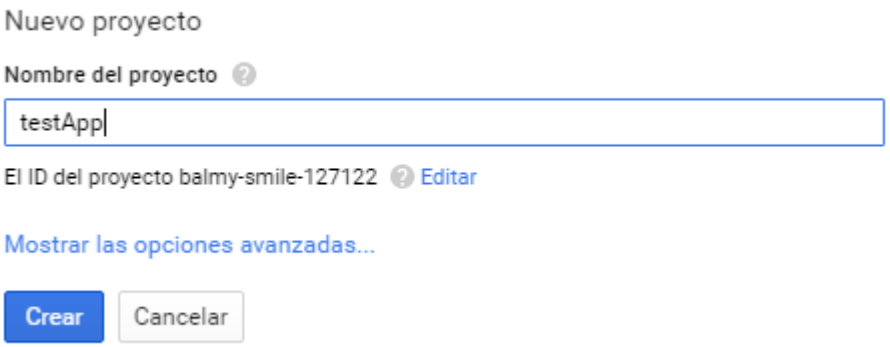
Al igual que Facebook, en el listado de aplicaciones de Apps se puede crear una nueva de forma muy sencilla.

Figura 63. **Listado de apps en API de Google**



Fuente: elaboración propia.

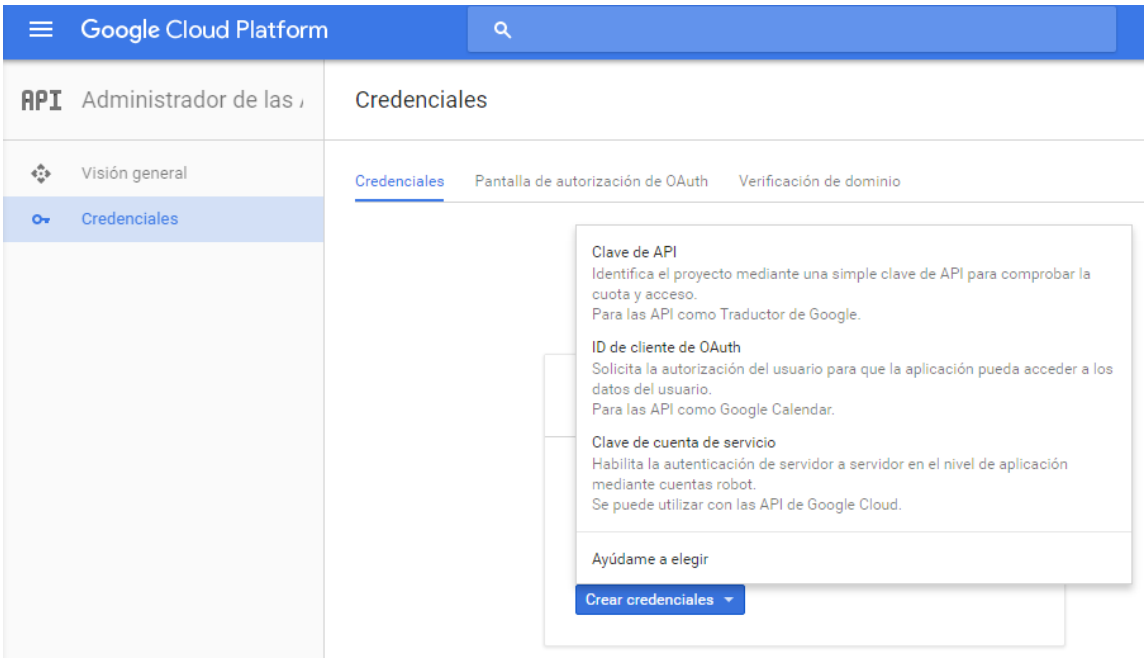
Figura 64. **Generación de ID para nueva App**



Fuente: elaboración propia.

En el panel de administración para las Apps, específicamente en el apartado de credenciales, es donde se define las plataformas y el tipo de credencial a utilizar.

Figura 65. **Creación de credenciales**



Fuente: elaboración propia, empleando herramienta de recortes.

4.1.7. SDK Android

Es el *kit* de desarrollo de software (Software Development Kit) y permite desarrollar aplicaciones móviles para sistemas Android. Una de las principales características es que se puede ejecutar un emulador de dicho SO para poder probar las aplicaciones que se desarrollen.

El SDK, además provee distintas versiones de Android y da acceso a sus múltiples APIs para lograr una mayor interacción entre la aplicación y el SO. Para el desarrollo de dichas aplicaciones móviles se utiliza Java como su lenguaje de programación.

Google, que es propietaria de Android, genera un SDK cada vez que se crea una nueva versión de Android con el fin de que los desarrolladores puedan construir aplicaciones que se adapten a los dispositivos que salen actualmente al mercado.

4.2. Hardware

Varía dependiendo la tecnología aplicada, por ejemplo, la Nube.

4.2.1. Máquina virtual en Cloud 9

Cloud 9 es una plataforma que provee entornos virtuales en la nube para el desarrollo en equipo, ya que se pueden compartir los entornos entre usuarios para desarrollar en paralelo e ir ejecutando el código para realizar pruebas en tiempo real.

Cloud 9 tiene máquinas pre configuradas con las herramientas y lenguajes de programación más comunes como: Nodejs para JavaScript, Django para Python, Rails para Ruby, entre otros. Esta herramienta es muy útil, ya que tiene pre configurados los entornos. Ahorra tiempo y problemas que le llevan al desarrollador cuando preparara su entorno. Otra alternativa es crear un entorno a partir de un repositorio en Github, lo cual es muy útil, ya que se puede estar integrando el código una vez probado.

Esta herramienta se utilizó para el desarrollo de la API RESTful. Las especificaciones técnicas de la máquina que se utilizó se muestran en la tabla IX.

Tabla IX. **Máquina API RESTful**

Característica	**Máquina virtual Cloud 9**
Sistema operativo	Ubuntu 14.04.3 LTS
CPU	Procesado con 1 núcleo
Memoria RAM	1 GB
Disco duro	5 GB
Herramienta de desarrollo	Nodejs

Fuente: elaboración propia.

Para utilizar Cloud 9 hay que registrarse en su página oficial y tienen dos tipos de planes de pago y gratis. La versión gratis, que fue la que se utilizó para este caso, permite crear un número limitado de entornos y la máquina virtual del entorno de desarrollo se apaga después de un tiempo determinado.

4.2.2. Smartphone Android 4.x.x

Para realizar las respectivas pruebas durante el desarrollo se utilizó el dispositivo que se describe en la tabla X.

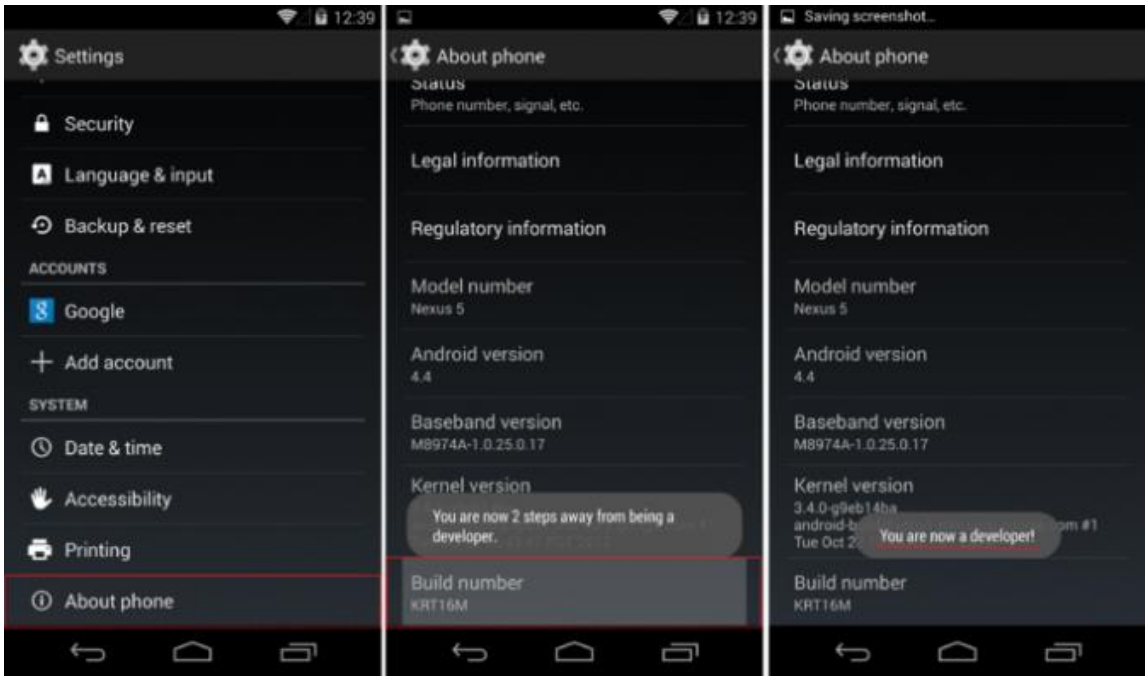
Tabla X. **Dispositivo móvil**

Característica	**UMI-emax-mini**
Sistema operativo	Android 4.4
CPU	Qualcomm Snapdragon 615 MSM8939 Octa-core (4x 1.5 GHz Cortex-A53 & 4x 1.0 GHz Cortex-A53) 64 bits
Procesador gráfico	Adreno 405
Almacenamiento interno	16 GB
Modo desarrollador	Activado

Fuente: elaboración propia.

Tomar en cuenta que, para utilizar el dispositivo y hacer pruebas de desarrollo es necesario activar el modo de desarrollador que se hace dirigiéndose a ajustes del teléfono, luego en la opción: acerca del teléfono y, debe pulsar varias veces el número de compilación, hasta que salga el mensaje de que se ha activado el modo de desarrollador.

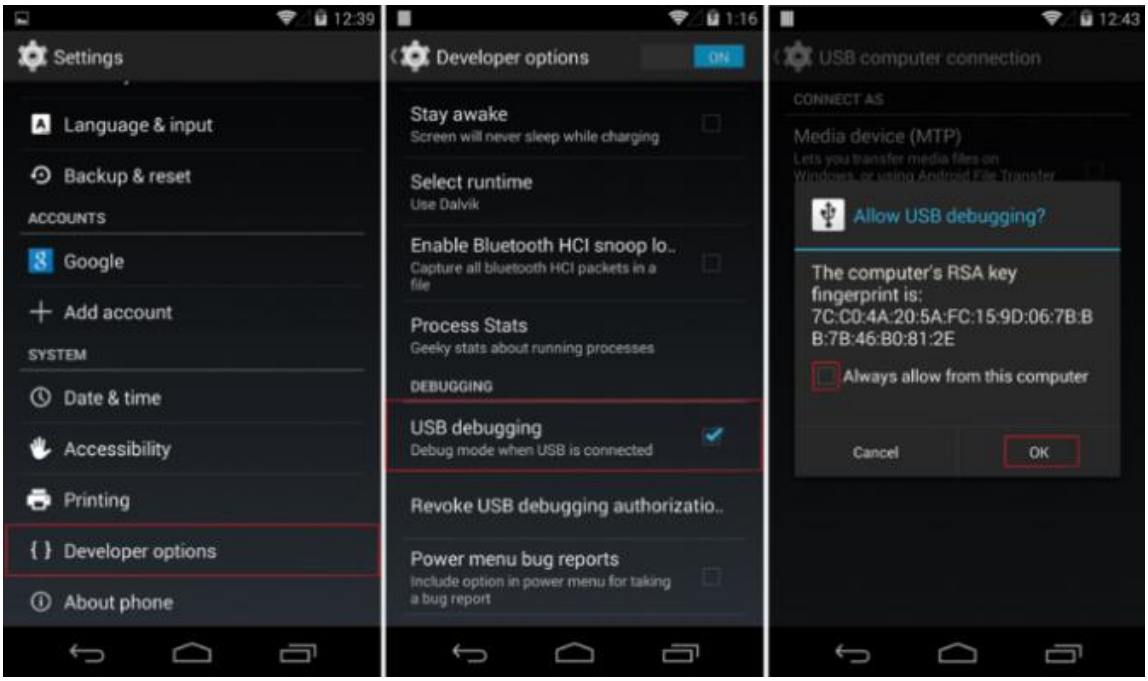
Figura 66. **Activar modo desarrollador**



Fuente: *Cómo activar el modo depuración USB en Android.* http://www.elandroidelibre.com/2015/01/como-activar-el-modo-depuracion-usb-en-android.html. Consulta: abril de 2016.

Una vez activado el modo desarrollador en los ajustes del teléfono, se tendrá una nueva opción para desarrolladores en la que se debe activar el modo de depuración y autorizar la computadora a la que está conectado el dispositivo.

Figura 67. **Activar modo de depuración**



Fuente: *Cómo activar el modo depuración USB en Android.* http://www.elandroidelibre.com/2015/01/como-activar-el-modo-depuracion-usb-en-android.html. Consulta: abril de 2016.

La versión Android para correr la App, sin ningún problema por recursos es la versión 4.4 con un procesador de 1 núcleo y 512 MB de memoria RAM.

4.2.3. Computadora para desarrollo

Para el desarrollo de la App se utilizaron dos computadoras portátiles con las especificaciones que se describen en la tabla XI.

Tabla XI. **Especificaciones computadoras de desarrollo**

Característica	**Máquina núm. 1**	**Máquina núm. 2**
Sistema operativo	Crunchbang ++ (Debian)	Ubuntu 14.04
CPU	Procesador I7 3ra generación	Procesador I3 3ra generación
Memoria RAM	8 GB	4 GB
Disco duro	750 GB	640 GB
Herramienta de desarrollo	Nodejs v4, Ionic framwerok v1.7.14, Java development kit (JDK), Android Development Kit (SDK).	Nodejs v4, Ionic framwerok v1.7.14, Java development kit (JDK), Android Development Kit (SDK).
Modelo	Toshiba Satelite s855-7352	Toshiba Satelite c845

Fuente: elaboración propia.

Un punto importante para la configuración del entorno de desarrollo es declarar la variable del SDK de Android en las variables de entorno del sistema. Para ello en una distribución Linux basada en Debian se debe modificar el siguiente archivo:

- /etc/bash.bashrc

Al final del archivo se agrega la siguiente línea:

- export PATH=${PATH}:/home/USER/sdk/tools:/home/USER/sdk/platform-tools.

Donde “home/user” es la dirección donde se guardó el SDK de Android.

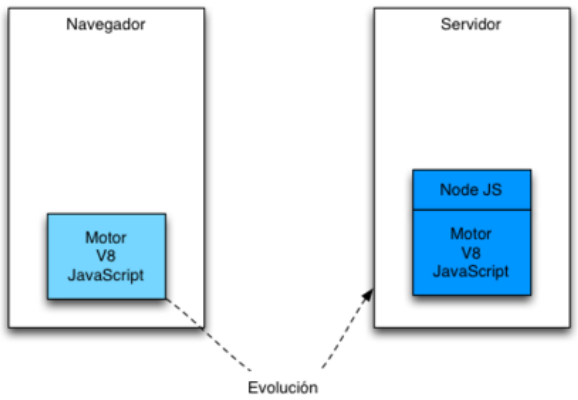
4.3. Software

Existen gran cantidad de tecnologías nuevas que ayudan a la creación de software de mejor calidad.

4.3.1. NodeJS

Google posee el motor V8 que fue diseñado para ejecutar código JavaScript de forma rápida en un navegador web, por ende, NodeJS se basa en dicho motor para crear aplicaciones web (y otras), utilizando JavaScript del lado del servidor.

Figura 68. **Diseño de NodeJS**



Fuente: ÁLVAREZ, Cecilio. *¿Cómo funciona NodeJS?.* http://www.genbetadev.com/*frameworks*/como-funciona-node-js. Consulta: marzo de 2016.

NodeJS tuvo su lanzamiento el 27 de mayo de 2009, y actualmente se encuentra su última versión estable, la 5.9.1. Tiene licenciamiento MIT y está disponible en varios sistemas operativos como: Linux, Windows, Mac OS, entre otros.

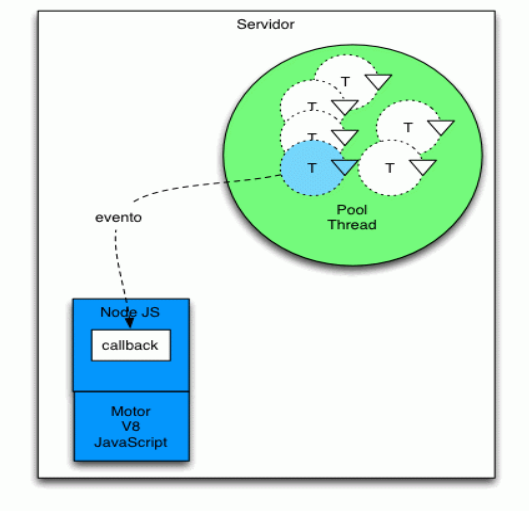
NodeJS está enfocado a programación orientada a eventos, en donde su especialidad es ser utilizado para el desarrollo de aplicaciones de red escalables.

- Eventos asíncronos

Una de las particularidades de NodeJS es que utiliza un único hilo de ejecución que gestiona todo el flujo de peticiones y trabajo que realiza.

Es por este motivo que dicho hilo de ejecución delega las tareas a realizar a un pool de hilos que resuelven dichas tareas de forma independiente, en tiempos independientes (asíncrona), y que retornan su respuesta mediante una función Callback a NodeJS.

Figura 69. **Manejo de eventos asíncronos en NodeJS**



Fuente: ÁLVAREZ, Cecilio. *¿Cómo funciona NodeJS?.* http://www.genbetadev.com/*frameworks*/como-funciona-node-js. Consulta: marzo de 2016.

- Node modules y NPM

Los módulos también son manejados en NodeJS y lo hace muy liviano, ya que se pueden ir agregando módulos según sea la necesidad. Por ejemplo

existen módulos para la gestión de bases de datos como MySQL, o para generar cadenas cifradas, o para lectura y escritura de archivos, entre otros.

NodeJS permite crear módulos personalizados para exportarlos, y además existen módulos que ya están cargados y compilados dentro de NodeJS, por lo que solo es necesario importarlos.

En cuanto a importar módulos externos se refiere, existe NPMm (Node Package Manager) que es un gestor de paquetes para NodeJS que permite compartir y reutilizar código hecho por otras personas que trabajan en JavaScript.

La comunidad de NPM constantemente libera diversos módulos para ser utilizados en proyectos de NodeJS, estos están documentados y, dependiendo del autor del mismo, puede brindar soporte sobre su implementación. Además, cuando se instala un nuevo módulo este automáticamente incluye todas sus dependencias para ser utilizado sin ningún problema.

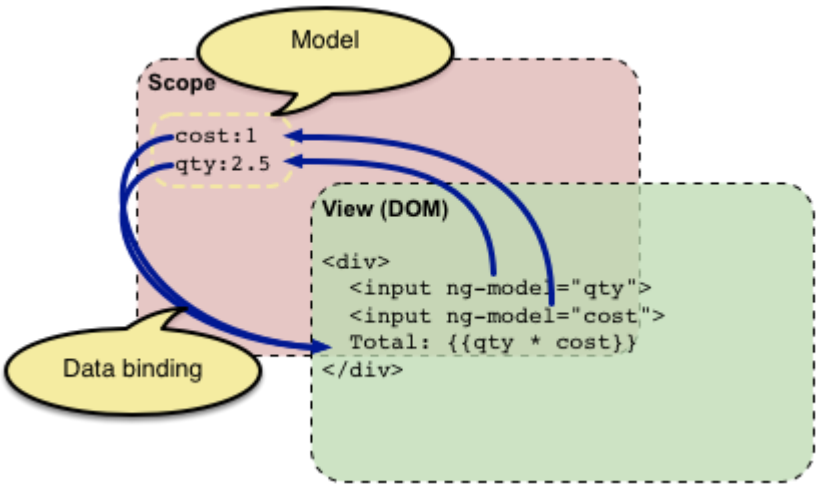
4.3.2. AngularJS

Es un conjunto de librerías hechas en JavaScript, cuyo objetivo principal es realizar aplicaciones web dinámicas. AngularJS se basa en el patrón de diseño MVW (modelo vista y cualquier alternativa). Los desarrolladores, por lo regular implementan el MVC (Modelo Vista Controlador).

AngularJS utiliza programación declarativa, que significa que el desarrollador se encarga de los datos y AngularJS de manipular el árbol DOM. Para una mejor comprensión AngularJS no busca elementos como JavaScript convencional sino más bien declara dónde deben incluirse las variables para

que pueda hacer el trabajo de más bajo nivel. Para una mejor compresión se ilustra en la figura 70.

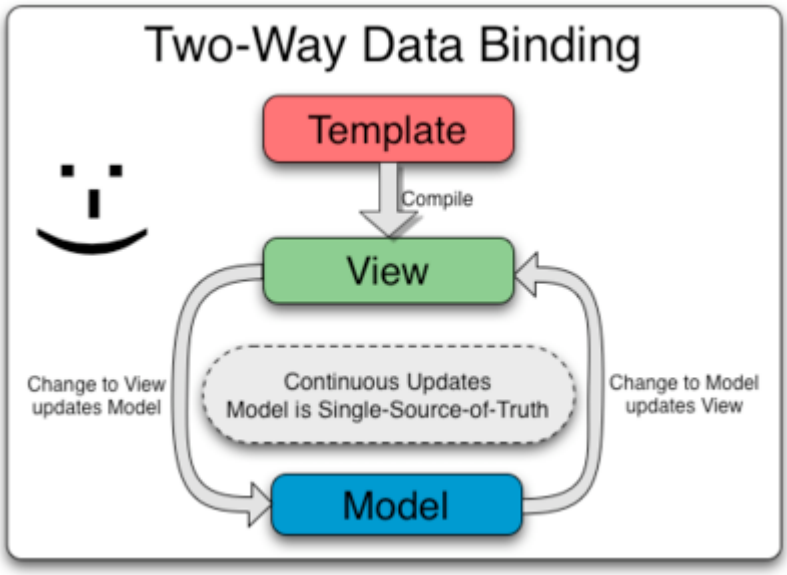
Figura 70. **DOM AngularJS**



Fuente: *Conceptual Overview. https://docs.angularjs.org/guide/concepts.* Consulta: abril de 2016.

AngularJS utiliza el concepto de *data binding* mejorado, ya que primero compila el *template* y está en continua actualización con el modelo.

Figura 71. **AngularJS *Data Binding***



Fuente: *Data Binding. https://docs.angularjs.org/guide/databinding.* Consulta: abril de 2016.

Es importante tener claro el concepto de cómo se ejecuta AngularJS al momento de carga de la página web. Cuando se incluye AngularJS dentro de las páginas HTML, este se ejecutará cuando toda la página se termine de cargar. Una vez cargada, la página AngularJS explorará todos los elementos del DOM en busca de alguna directiva, al encontrar alguna se definirá un comportamiento en función de la misma.

AngularJS debe saber si está siendo instanciado, de lo contrario no se ejecutará, por lo que la primera etiqueta que debe encontrar es ngApp y solo dará funcionalidad a los elementos que dentro de la jerarquía del DOM, sean hijos del elemento en donde se encuentra la etiqueta mencionada. Podría ser una ventaja, ya que se puede implementar otros *framework*s dentro del mismo proyecto sin generar conflictos.

Para entender mejor el funcionamiento de AngularJS es necesario profundizar en sus componentes fundamentales.

- Directivas

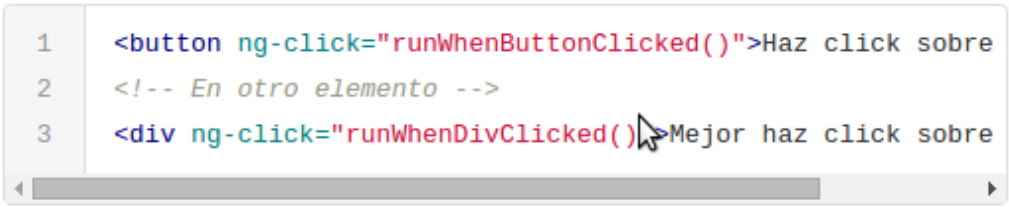
Para entender que es una directiva, y como funciona, se puede hacer una analogía. Para el desarrollo de páginas web se usa el lenguaje HTML que se basa en etiquetas. Cada una tiene sus propiedades y representan una porción diferente de la página web, por ejemplo, la etiqueta <href=””> hace referencia a un link, otro ejemplo es la etiqueta de video o de audio, el navegador web interpretará las etiquetas según su función y mostrará el contenido respectivo.

Sin embargo, el navegador tiene un número finito de etiquetas HTML a interpretar, si se quisiera agregar nuevas etiquetas el navegador no las reconocería. AngularJS viene a solucionar este problema y permite crear elementos DOM nuevos que se pueden personalizar para tareas específicas, según la necesidad de la solución.

Entonces, una directiva es una función que retorna un elemento DOM para nuevas funcionalidades y permite extender el HTML con funcionalidades propias.

AngularJS trae un conjunto de directivas integradas que lo hacen poderoso. Estas directivas permiten mostrar contenido, ocultar contenido, hacer ciclos para no repetir código, capturar eventos, entre otros. Las directivas propias de AngularJS se identifican por utilizar la siguiente notación: “ng-identificador”.

Figura 72. **Directivas AngularJS**



Fuente: *Directivas.* http://raulexposito.com/documentos/como-aprender-angularjs/. Consulta: abril de 2016.

- Módulos

Como AngularJS es un *framework* basado en JavaScript se puede escribir código en JavaScript, pero hay que indicarle donde esté el nuevo código. Esto se hace por medio de módulos definiendo uno o bien instanciarlo.

Para definir un módulo se utiliza la API de AngularJS y se ve como se describe en la figura 73.

Figura 73. **Módulo AngularJS**

Fuente: *Módulos.* http://raulexposito.com/documentos/como-aprender-angularjs/. Consulta: abril de 2016.

En la figura 74 se declara myApp, que sería el módulo principal, y se accede a las funcionalidades desarrolladas, invocando el módulo por su nombre.

Figura 74. **Instancia de módulo AngularJS**

```
1  <html ng-app="myApp">
2  <head>
3  <!-- etc. -->
4  </head>
5  <body>
6  <!-- etc. -->
7  </body>
8  </html>
```

Fuente: *Módulos.* http://raulexposito.com/documentos/como-aprender-angularjs/. Consulta: abril de 2016.

Los módulos ayudan a organizar mejor las funcionalidades, ya que se puede incluir módulos dentro del principal para agregar funcionalidades.

- Ámbitos

Los ámbitos son representados en AngularJS con la siguiente variable: “$scope”, también hay otro tipo que es el “$rootScope”.

Los ámbitos son la unión entre el código propio y lo que muestra el navegador. AngularJS utiliza ámbitos para comunicarse entre componentes.

Puede manejar varios ámbitos, sin embargo, el “$rootScope” puede ser accedido por todos los componentes desde cualquier lugar de la vista.

Un detalle importante es que el enlace entre la comunicación es bidireccional, ya que desde la vista se puede cambiar el valor de la variable. Una desventaja que podría llegar a tener es que si se carga mucho el “$rootScope” puede llegar a ser difícil el acceso a los datos.

AngularJS maneja muchos ámbitos, por ejemplo, cada vez que se declara un controlador se crea un nuevo ámbito lo cual permite una mejor organización de las variables que se utilizan duran la ejecución.

- Controladores

AngularJS utiliza el concepto de controladores para organizar grupos de elementos que pueden ser accedidos desde la vista. Entonces un controlador es una porción de código donde se define la funcionalidad de un segmento de la página web.

Figura 75. **AngularJS controlador**

Fuente: *Controladores.* http://raulexposito.com/documentos/como-aprender-angularjs/. Consulta: abril de 2016.

Figura 76. **Vista controlador AngularJS**

```
<div ng-controller='HomeController'>
<!--
Aqui tenemos acceso al objeto
$scope definido en el HomeController
-->
</div>
```

Fuente: *Controladores.* http://raulexposito.com/documentos/como-aprender-angularjs/. Consulta abril de 2016.

- $http y promesas

$http es un objeto que ayuda a hacer peticiones HTTP y procesar las respuestas. Esto sirve para obtener la información de fuentes externas como APIs. La estructura de unan petición se muestra en la figura 77.

Figura 77. **Petición HTTP AngularJS**

```
$http.get('/someUrl', config).then(successCallback, errorCallback);
$http.post('/someUrl', data, config).then(successCallback, errorCallback);
```

Fuente: *peticiones HTTP.* https://docs.angularjs.org/api/ng/service/$http. Consulta: abril de 2016.

La estructura es sencilla, se maneja como un JSON y en el método se le indica si es *get*, *post*, *put* o *delete*. Se indica también la dirección a donde se harán las peticiones y los parámetros siempre con la notación JSON.

Las peticiones HTTP realizadas con el objeto $http de AngularJS son asíncronas lo cual permite que la página siga funcionando mientras se obtiene la información del servidor. Sin embargo, esto puede ser contraproducente y complicar el flujo de control de la información. Para solucionar ese problema se tienen dos opciones:

- Pasar una función que se ejecute como un *callback*
- Utilizar una promesa

AngularJS utiliza mucho las promesas para manejar el código asíncrono como si fuera síncrono. La estructura de una promesa se muestra en la figura 78.

Figura 78. **Promesa AngularJS**

```
promise
.then(function(data) {
  // Invocado cuando no hay errores con los datos
})
.catch(function(err) {
  // Invocado cuando surge un error
})
.finally(function(data) {
  // Invocado siempre, independientemente del result
})
```

Fuente: *Promesas.* http://raulexposito.com/documentos/como-aprender-angularjs/. Consulta: abril de 2016.

Las promesas tiene 3 métodos: *.then*, se cumple cuando la transacción se llevó a cabo sin ningún error, en ese método entonces se ejecuta la acción si todo fue correcto. El método *.catch*, se cumple cuando ocurre un error y se rechaza la petición. Y el método *finally*, se ejecuta siempre, haya o no error.

- Inyección de dependencias

La inyección de dependencias hace referencia a cómo definir las dependencias en el proyecto. Por ejemplo, NodeJS utiliza el *require* o bien si se habla de un proyecto en Java sería el *import*. Entonces la inyección de dependencias es la forma en que se deben incluir las dependencias que se necesitan para que el código se ejecute de forma correcta y totalmente funcional.

Por ejemplo, si dentro de un controlador se necesita incluir el objeto $scope debe definirse como se indica en la figura 79.

Figura 79. **Inyección AngularJS**

```
1  angular.module('myApp', [])
2  .controller('HomeController', function($scope, $q) {
3    // Tenemos los objetos $scope y $q
4    // disponibles en este punto
5  });
```

Fuente: *Inyección de dependencias.* http://raulexposito.com/documentos/como-aprender-angularjs/. Consulta: abril de 2016.

Se pueden inyectar varios objetos en cada controlador para diferentes funcionalidades. El orden de la definición no altera el resultado y no se puede tener dependencias cíclicas, de lo contrario no se ejecutara el código.

- Servicios, factorías y proveedores

Los servicios, factorías y proveedores en AngularJS son muy similares y se utilizan para situaciones parecidas. Una de las diferencias entre ellos es la forma en que se configuran. La idea de utilizarlos es una mejor organización del código manteniendo los controladores lo más fino que se pueda, ya que una buena práctica es que cada sección de código solo se encargue de una tarea en específico.

Figura 80. ***Factory, Service, Provider***

```
var myApp = angular.module('myApp', []);

//service style, probably the simplest one
myApp.service('helloWorldFromService', function() {
    this.sayHello = function() {
        return "Hello, World!";
    };
});

//factory style, more involved but more sophisticated
myApp.factory('helloWorldFromFactory', function() {
    return {
        sayHello: function() {
            return "Hello, World!";
        }
    };
});

//provider style, full blown, configurable version
myApp.provider('helloWorld', function() {

    this.name = 'Default';

    this.$get = function() {
        var name = this.name;
        return {
            sayHello: function() {
                return "Hello, " + name + "!";
            }
        }
    };

    this.setName = function(name) {
        this.name = name;
    };
});
```

Fuente: *AngularJS: Service vs Provider vs Factory.* http://stackoverflow.com/questions/15666048/angularjs-service-vs-provider-vs-factory/20613879#20613879. Consulta: abril de 2016.

Como se puede notar en la figura 80 los tres métodos para organizar el código cumplen con la misma función, sin embargo, el servicio retorna el valor de una función, mientras la factoría retorna una función y el proveedor retorna una función que retorna otro resultado. La diferencia no solo está en la definición sino en el acceso a los datos

4.3.3. Ionic *framework*

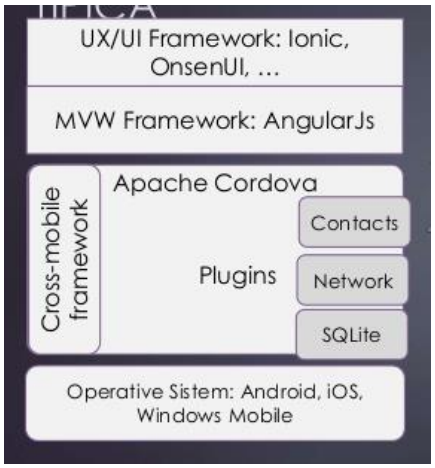
Es un AngularJS para desarrollo de aplicaciones híbridas, que actualmente está en la versión 1.2 Beta. Para el desarrollo de las aplicaciones Ionic utiliza AngularJS y SASS, por definición se utilizan componentes de HTML, CSS

SASS es un preprocesador para CSS y proporciona componentes de CSS que hacen que una vista HTML tenga funcionalidades (como pull-down), es muy similar a las interfaces que una aplicación nativa posee.

Ionic mejora su rendimiento dado que no utiliza jQuery para su desarrollo, y además provee una CLI muy funcional por medio de la cual se puede levantar un pequeño servidor web, se pueden agregar *plugin*s externos, entre otros.

Se ejecuta sobre Córdova, que explota las funcionalidades y *plugins* que se utilizan para el mismo. Está diseñado principalmente para el desarrollo de aplicaciones nativas como Android e iOS, siempre utilizando *Cordova* como base.

Figura 81. **Estructura de una aplicación Ionic**



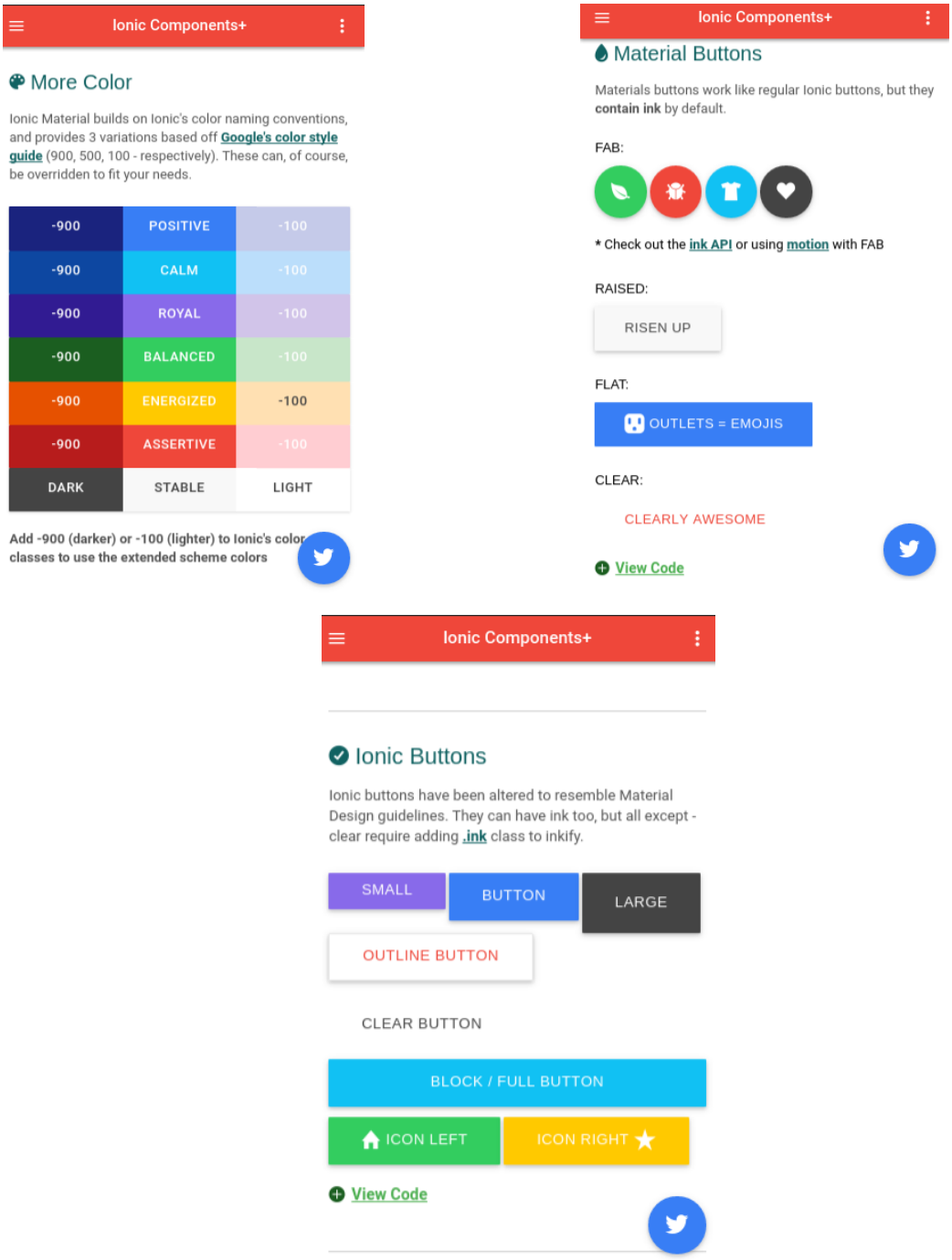
Fuente: *Desarrollo de aplicaciones móviles con Ionic y Apache Cordova.* http://es.slideshare.net/arignack/desarrollo-de-aplicaciones-mviles-con-ionic-y-apache-cordova. Consulta: abril de 2016.

4.3.3.1. Ionic material

Material Design es el más reciente paradigma de diseño visual de aplicaciones móviles y páginas responsivas. Fue propuesto por Google y se basa en objetos materiales que respetan leyes de física con respecto de posición, espacio y movimiento. En el diseño todo debe tener lógica, cada movimiento y posición de los objetos. Los colores y sombras juegan un papel importante en el aspecto visual realista.

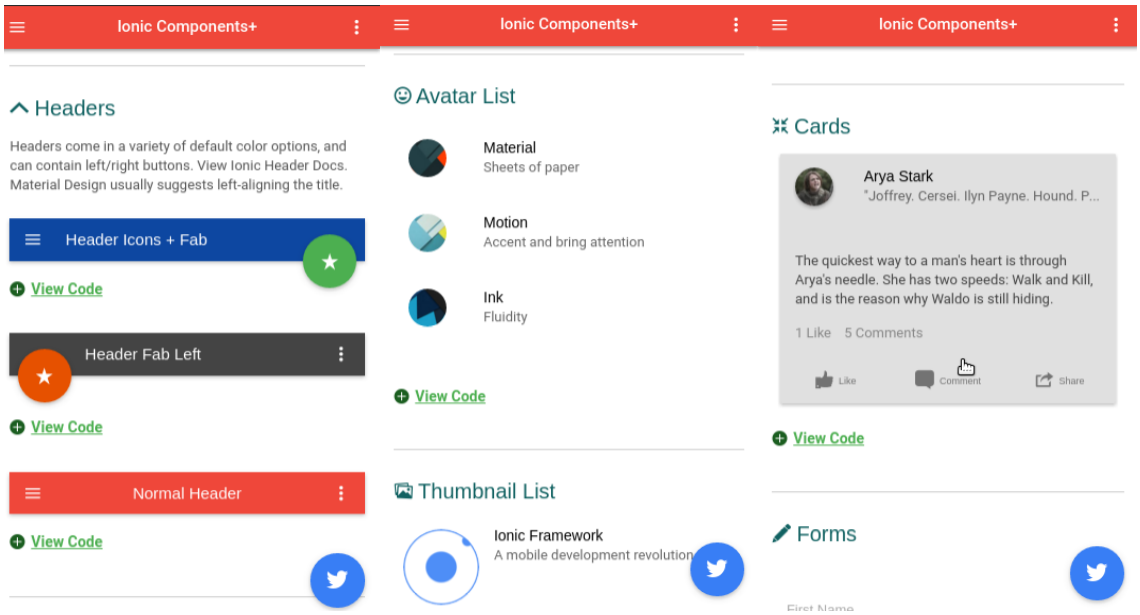
Ionic tiene su librería para el diseño con material basado en los principios que se mencionaron antes, que se nombró como Ionic material. Con esta librería se tiene acceso a los colores, botones, *headers*, *footers*, *Tabs*, iconos, listas, tarjetas entre otros.

Figura 82. **Paleta de colores y botones Ionic material**



Fuente: elaboración propia, empleando página oficial de Ionic material.

Figura 83. ***Headers*, listas y tarjetas Ionic material**



Fuente: elaboración propia, empleando página oficial de Ionic material.

4.3.3.2. Cordova *plugins*

Cordova ofrece una gran gama de *plugins* que pueden ser aprovechados para las aplicaciones nativas que se desarrollan con Ionic. Se detallan los *plugins* que se utilizaron para el desarrollo de la aplicación, que es parte de la solución planteada en este trabajo de graduación y son:

- Phonegap Facebook *plugin*

Es un *plugin* empleado para utilizar la API de Facebook mediante Cordova en su versión 3.5 en adelante. Este *plugin* ofrece una forma muy sencilla de conectar un dispositivo hacia la API de Facebook y provee funciones como login, logout, entre otras.

Es capaz de utilizar los Show Dialogs para JavaScript o incluso utilizar directamente el Graph API de Facebook de forma fácil.

- Cordova Network Information

Este *plugin* provee información del estado de la red para una aplicación híbrida. Lo más común es utilizarlo para determinar si un dispositivo tiene o no conexión a internet.

- Cordova p*lugin device*

Este *plugin* funciona únicamente en una aplicación nativa (Android, iOS), ya que se encarga de obtener información sobre el hardware del mismo, como la versión o el UUID (identificador único para dispositivos móviles), por lo que, si se quiere utilizar en la aplicación web con Ionic mostrará, ciertos errores y excepciones.

- Cordova *plugin whitelist*

Permite definir una lista de URLs por las cuales se puede navegar, esta configuración se realiza en el archivo config.xml que posee una aplicación Ionic.

4.3.3.3. Componentes CSS

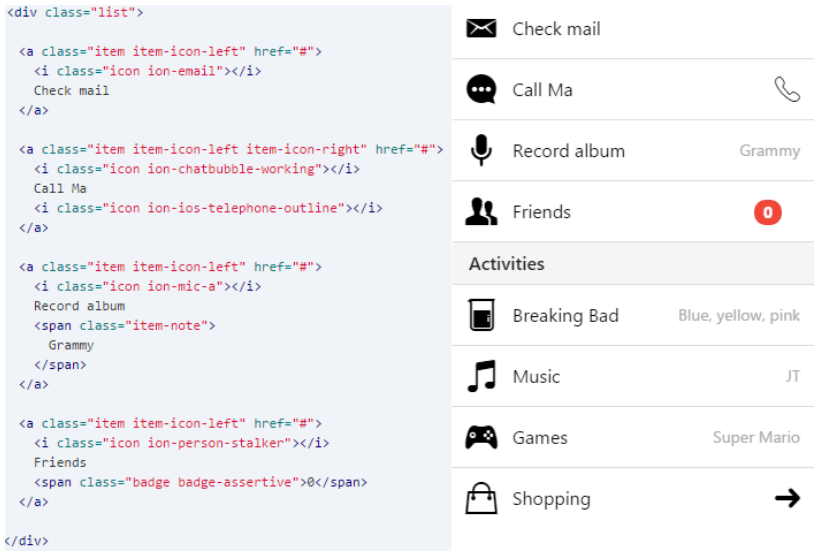
Ionic cuenta con sus propios componentes CSS para su uso dentro de vistas HTML, que le dan un aspecto muy similar a las vistas de una aplicación nativa.

Figura 84. **Estilos de botones en Ionic**

```
<button class="button">
  <i class="icon ion-loading-c"></i> Loading...
</button>

<button class="button icon-left ion-home">Home</button>

<button class="button icon-left ion-star button-positive">Favorites</button>

<a class="button icon-right ion-chevron-right button-calm">Learn More</a>

<a class="button icon-left ion-chevron-left button-clear button-dark">Back</a>

<button class="button icon ion-gear-a"></button>

<a class="button button-icon icon ion-settings"></a>

<a class="button button-outline icon-right ion-navicon button-balanced">Reorder</a
>
```


Fuente: elaboración propia, empleando página oficial de Ionic.

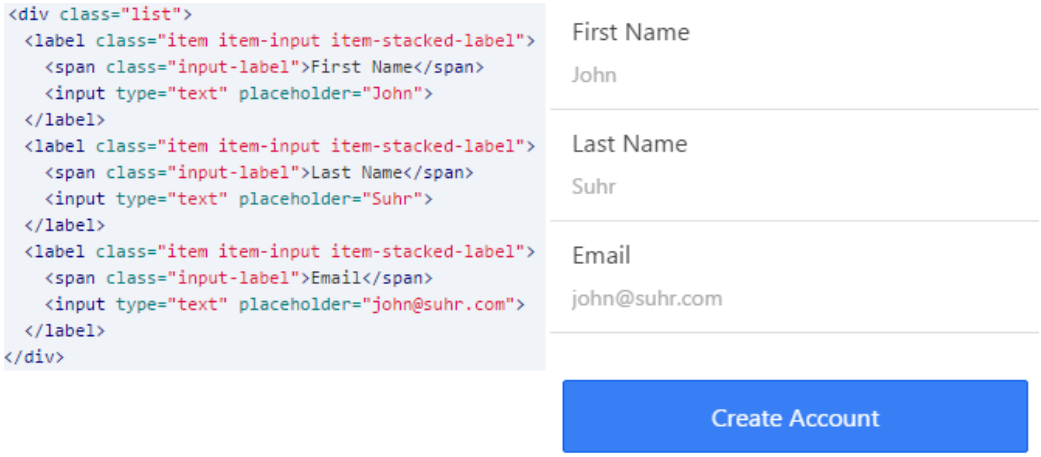
Figura 85. **Tipos de listas en Ionic**



```
<div class="list">

  <a class="item item-icon-left" href="#">
    <i class="icon ion-email"></i>
    Check mail
  </a>

  <a class="item item-icon-left item-icon-right" href="#">
    <i class="icon ion-chatbubble-working"></i>
    Call Ma
    <i class="icon ion-ios-telephone-outline"></i>
  </a>

  <a class="item item-icon-left" href="#">
    <i class="icon ion-mic-a"></i>
    Record album
    <span class="item-note">
      Grammy
    </span>
  </a>

  <a class="item item-icon-left" href="#">
    <i class="icon ion-person-stalker"></i>
    Friends
    <span class="badge badge-assertive">0</span>
  </a>

</div>
```


Fuente: elaboración propia, empleando página oficial de Ionic.

Figura 86. **Formularios en Ionic**



Fuente: elaboración propia, empleando página oficial de Ionic.

Ionic provee además, muchos más componentes como tablas, pestañas, encabezados, entre otros. Asimismo provee una serie de funciones que pueden ser ejecutadas desde algún controlador de AngularJS para definir objetos dinámicos o efectos a las vistas.

4.3.4. MySQL

Es un sistema administrador de bases de datos muy popular. Esta desarrollado, distribuido, y soportado por la corporación Oracle. Soporta transacciones que cumplen el estándar ACID, esto hace que sea una opción confiable para almacenar la información de manera estructurada y ordenada para su posterior consulta.

Es una base de datos relacional. Se basa en el leguaje estándar de Consultas (SQL). Que permite de una forma estándar obtener acceso a la información almacenada. Posee dos tipos de motores para diferentes

necesidades, por ejemplo, el motor MyISAM es para tablas no transaccionales y el motor InnoDB es para tablas relacionales.

El fin de usar este administrador de bases de datos es poder organizar y estructurar la información de manera que sea fácil acceder a ella y con la menor redundancia posible para no utilizar espacio ineficientemente.

4.4. Tutorial de desarrollo

Define la creación del entorno de trabajo y toda la configuración necesaria para la implementación de distintas tecnologías.

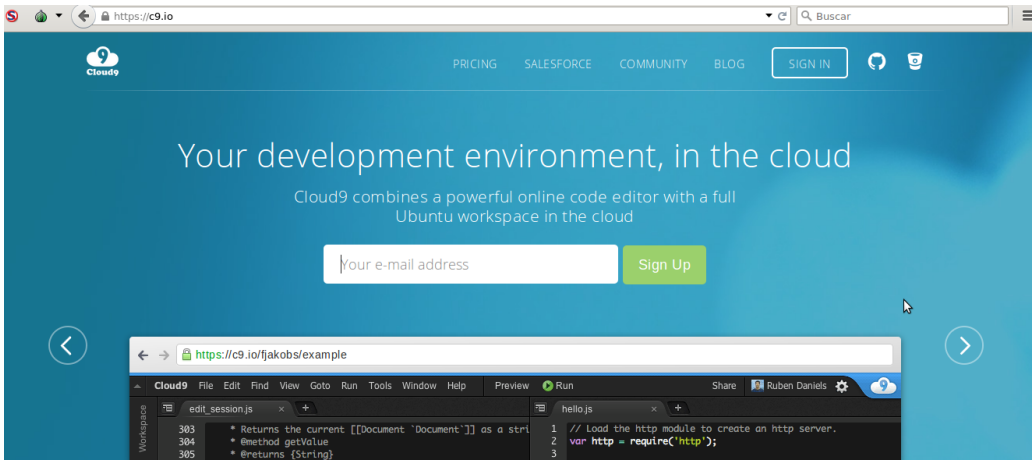
4.4.1. API RESTful

La API se desarrolló con nodejs utilizando ActionHero. Y como IDE para el desarrollo se utilizó C9.

4.4.1.1. Creación de máquina virtual en c9.io

Para utilizar C9 el primer paso es registrarse en la página oficial. La página se muestra en la figura 87.

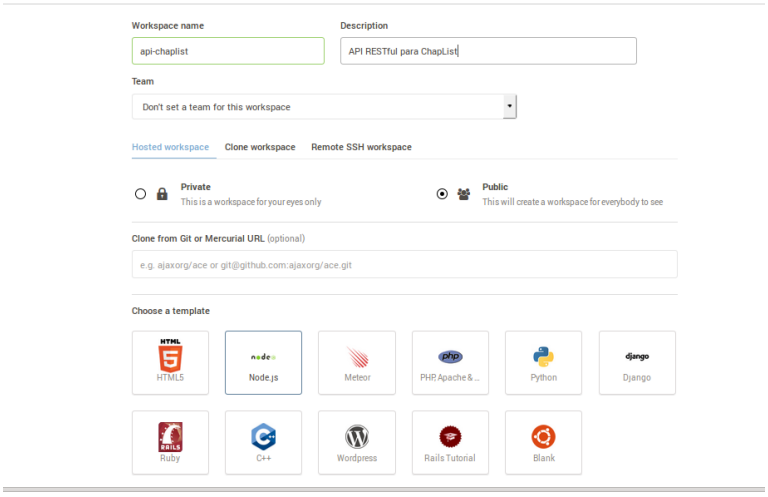
Figura 87. **Registro c9**



Fuente: elaboración propia, empleando página oficial de c9.

Luego se debe crear un espacio de trabajo al que se le indica el nombre y el lenguaje de programación con el que se trabajará.

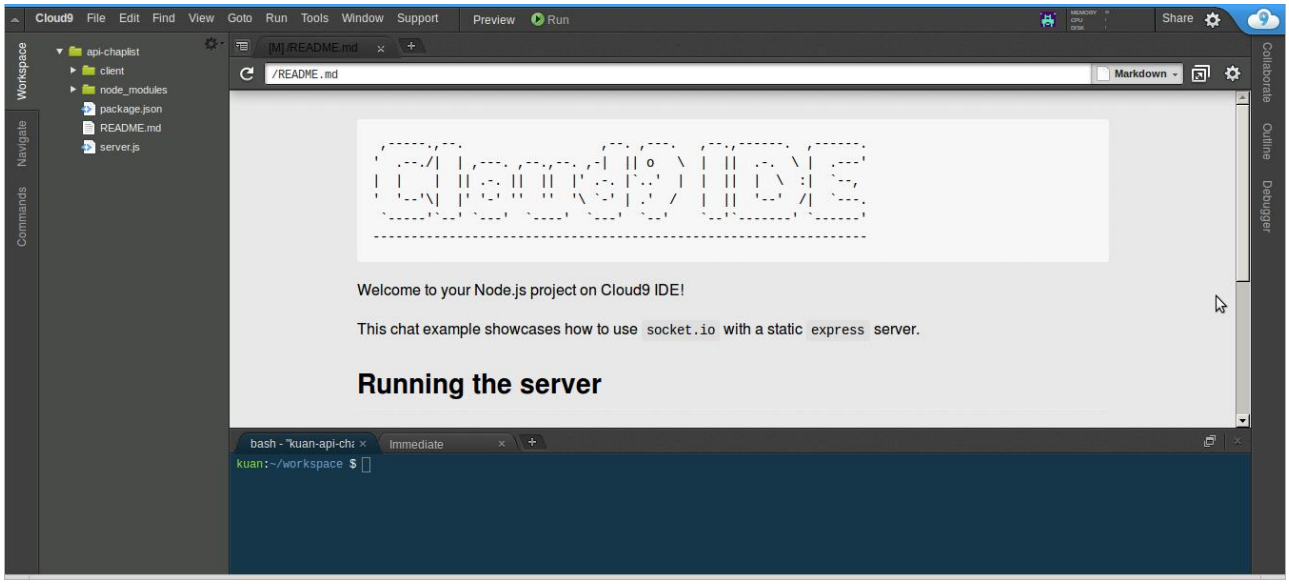
Figura 88. **Crear espacio de trabajo c9**



Fuente: elaboración propia, empleando página oficial de c9.

Una vez creado el espacio de trabajo se presentará el ambiente de desarrollo, en el que se tendrá un árbol de archivos y carpetas, una terminal Linux y un área donde se puede tener un editor, navegador entre otros.

Figura 89. **Ambiente c9**

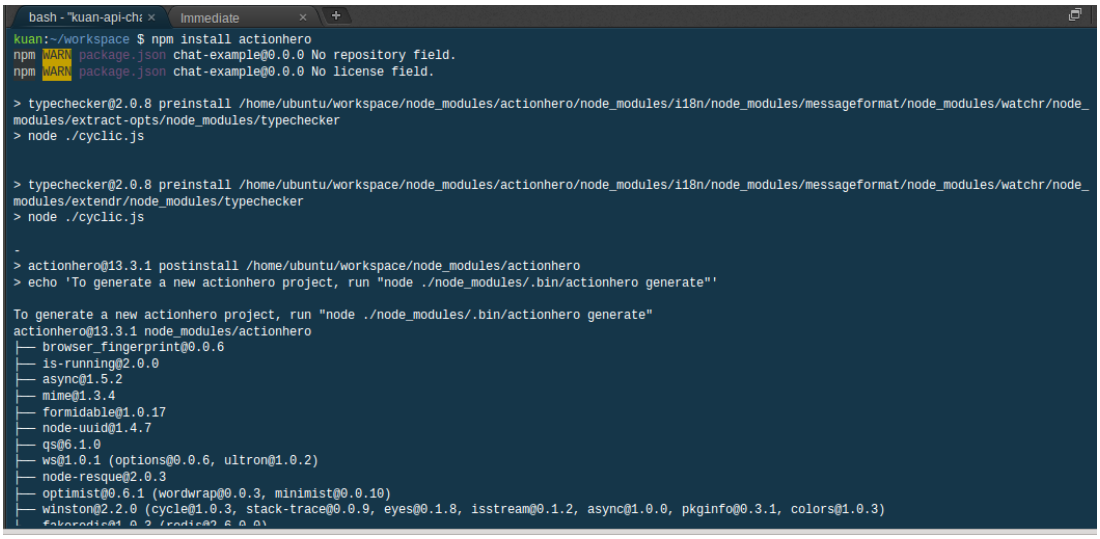


Fuente: elaboración propia, empleando página oficial de c9.

4.4.1.2. Configuración de entorno de desarrollo

Las herramientas que hay que configurar en la máquina de c9 para realizar el proyecto son: ActionHero, MySQL. Se hace por medio NPM.

Figura 90. **Instalación ActionHero c9**



Fuente: elaboración propia, empleando página oficial de c9.

La instalación de MySQL para NodeJS es con el siguiente comando: “npm install mysql”.C9 tiene instalado MySQL y la configuración para utilizarlo se describe en la figura 91.

Figura 91. **Mysql c9**

```
# start MySQL. Will create an empty database on first start
$ mysql-ctl start

# stop MySQL
$ mysql-ctl stop

# run the MySQL interactive shell
$ mysql-ctl cli
```

Fuente: *Using MySQL with Cloud9.* https://docs.c9.io/docs/setting-up-mysql. Consulta: abril de 2016.

- Para el nombre de *Host* se utiliza la variable “process.env.IP”.
- Puerto 3306 que trae por defecto MySQL.
- El usuario es el nombre de usuario con el que se registró la cuenta.
- No hay contraseña para la base de datos.
- La base de datos que se usa se llama c9, y ya existe.

Para manejar la base de datos ActionHero utiliza el ORM Sequelize que también se debe instalar con NPM.

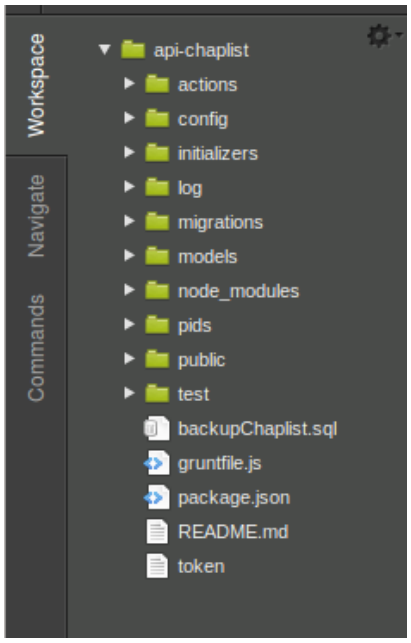
4.4.1.3. Creación y estructura de API RESTful en ActionHero

Para generar una nueva estructura de proyecto en ActionHero se utiliza el siguiente comando.

- $ node ./node_modules/.bin/actionhero generate

La estructura que se crea es la que se describe en la figura 91.

Figura 92. **Estructura ActionHero**



Fuente: elaboración propia, empleando página oficial de c9.

Una vez creada la estructura ya se procede a la programación. A continuación se explicará la estructura de la API de ChapList.

- Actions

En la carpeta actions se encuentran los servicios que la API expone. Para crear un nuevo Action se utiliza el siguiente comando.

  - $ node_modules/.bin/actionhero generateAction --name=” ”

Para el desarrollo de la API se utilizaron los siguientes *Actions*:

- appAction: para manejar los servicios de creación, eliminación, edición y actualización de Apps registradas.
- chaplistAction: se usó para la mayoría de servicios que consume la App móvil.
- offerAction: para la petición de productos ofertados que hace la App móvil a la API.
- storeAction: para los servicios que están relacionados con las Tiendas de los supermercados.
- userAction; para los servicios relacionados con los usuarios que registran sus Apps para consumir la API.

- Config

La carpeta Config contiene varios archivos de configuración importantes tales como:

- Api.js: sirve para las configuraciones de la API.
- Routes.js: sirve para definir las rutas con las que se expondrán los servicios.
- Sequelize.js: contiene la configuración de la base de datos.

- Initializers

En la carpeta Initializer se encuentran la lógica de los Actions que la API expone. Para crear un nuevo Initializer se utiliza el siguiente comando.

- $ node_modules/.bin/actionhero generateInitializer --name=” ”

Para el desarrollo de la API se utilizaron los siguientes Initializers:

- AppInit: contiene toda la lógica del registro de Apps.
- chaplistInit: contiene toda la lógica de los servicios que consume la App móvil
- modelsInit: contiene las relaciones entre tablas de la base de datos.
- offerInit: contiene la lógica de las solicitudes de ofertas.
- StoreInit: contiene la lógica de las solicitudes de las tiendas de los supermercados.
- *token*Init: contiene la lógica de la autenticación por *token* implementada.
- userInit: contiene la lógica de los usuarios que se registran en la plataforma de registro de Apps para consumo de API.

- *Log*

  Contiene un conjunto de archivos que corresponde al historial de actividad de la API.

- *Migrations*

  La carpeta Migrations contiene los archivos de migración a la base de datos. Para ejecutar las migraciones se ejecuta el siguiente comando.

- $ sudo node_modules/.bin/sequelize db:migrate

  AL ejecutar este comando las migraciones definidas se sincronizarán con la base de datos.

- *Models*

  La carpeta Models contiene los modelos creados, que equivalen a las tablas que se utilizan para guardar la información de forma persistente en la base de datos. Para crear un nuevo modelo se utiliza el siguiente comando.

  - $ model:create --name User --attributes id_user:string,firstName:string,lastName:string,email:string,image:string

Los modelos utilizados para Chaplist son los siguientes.

- App.js: modelo del registro de Apps para consumo de API.
- Offer.js: modelo para ofertas de supermercado.
- Product.js: modelo para productos de ofertas.
- productStore.js: relación entre las tiendas de supermercados y los productos.
- Store.js: modelo para la tienda de supermercado.
- Supermarket.js: modelo para supermercados.
- User.js: modelos de usuarios que se registran en la plataforma de registro de Apps.

- Node_modules

  Es la carpeta donde se encuentran todos los módulos que utiliza NodeJS para ejecutar correctamente la API.

- *Pids*

  Contiene los identificadores que servirían al momento de implementar un clúster de servidores.

- *Public*

  En esta carpeta está montada la plataforma de registro de Apps para el consumo de la API.

- Test

En esta carpeta se colocan los archivos para realizar pruebas a la API.

4.4.2. App móvil híbrida

Es la solución utilizada por los usuarios finales.

4.4.2.1. Configuración de entorno de desarrollo

Todas las herramientas que se utilizaron fueron instaladas sobre un SO Linux, específicamente la distribución Ubuntu 14.04.

- Instalación de java: se utilizó una versión libre del JDK y del JRE de Java.

  - $ sudo apt-get update
  - $ sudo apt-get install default-jre default-jdk

- Instalación SDK Android: existen varias formas de instalar el SDK de Android, por lo que se recomienda visitar el enlace de referencia.

- Instalación de NodeJS

    - $ sudo apt-get update
    - $ sudo apt-get install nodejs
    - $ sudo apt-get install npm (en caso de no estar instalado)

- Instalación de Ionic y Cordova: es necesario tener instalado NPM.

    - $ sudo npm install –g cordova
    - $ sudo npm install –g ionic

4.4.2.2. Creación y estructura de ChapList

Para crear y ejecutar una aplicación híbrida se puede realizar con simples comandos, teniendo preparado el entorno de desarrollo. Hay tres formas de crear una aplicación

Figura 93. **Creación de aplicación con Ionic**

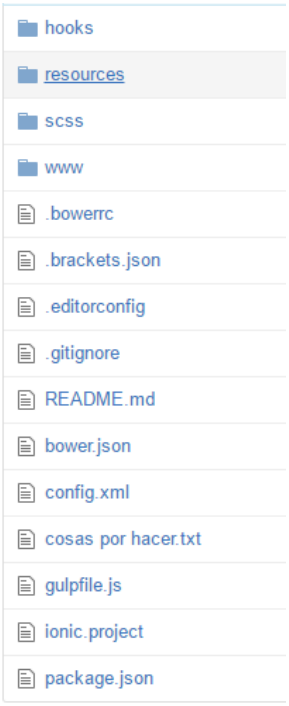
Fuente: elaboración propia, con información de Ionic *framework*.

La forma de ejecutar y de generar una aplicación nativa a partir de Ionic se realiza de la siguiente forma.

- Mover a la carpeta del proyecto desde la terminal con cd
- $ ionic platform add Android/iOS
- $ ionic build Android/iOS
- $ ionic run Android/iOS

Utilizando la CLI de Ionic es muy sencillo crear y generar aplicaciones móviles a partir de una híbrida en donde su estructura se compone de un conjunto de archivos y carpetas.

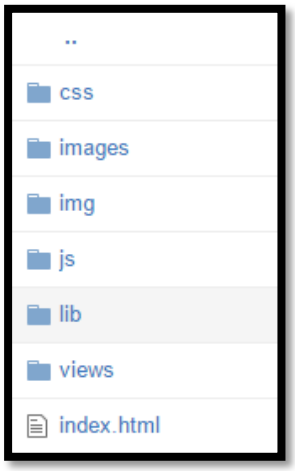
Figura 94. **Estructura de ChapList**



Fuente: elaboración propia.

En la figura 94 se muestra la estructura que se crea ante una nueva aplicación con Ionic. La carpeta de interés para el desarrollo es www que contiene todas las dependencias para utilizar AngularJS y el SASS de Ionic, para empezar a crear la funcionalidad de la aplicación.

Figura 95. **Contenido de carpeta www**



Fuente: elaboración propia.

- CSS: contiene todos los archivos de estilo que se utilizarán.
- Img/Images: imágenes o íconos que se utilizarán en las vistas.
- Js: ficheros JavaScript que utilizan AngularJS.
- Lib: contiene las librerías de AngularJS, Ionic, entre otras.
- Views: vistas que son embebidas dentro de la vista index.html.
- Index.html: vista principal en donde se importan todos los ficheros JS y CSS que se utilizarán, por medio de esta vista se muestran las otras.

La carpeta vital que contiene toda la lógica que maneja la aplicación es JS, que dentro posee otros archivos y carpetas que complementan todo el funcionamiento.

- Controllers
    - appCtrl.js: contiene varios controladores que son:

- AppCtrl: controlador encargado de definir las funciones que utilizan los otros controladores para el manejo de los efectos en las vistas es el controlador padre.

- HomeCtrl: es el controlador de la vista de inicio, se encarga de mostrar las ofertas favoritas en la vista de inicio y provee las funciones que redirigen a las demás partes y funcionalidades de la aplicación.

- MapCtrl: es el encargado de iniciar el proceso para mostrar los mapas de Google.

- OfferCtrl.js: este archivo también provee varios controladores.

  - OfferCtrl: controlador utilizado para obtener la lista de supermercados en la API y desplegarlos.

  - ProductCtrl: controlador utilizado para obtener productos de un supermercado seleccionado y los muestra mediante paginación de 10 elementos.

  - DetalleCtrl: este controlador provee información a detalle de un producto seleccionado y lo muestra en su respectiva vista.

  - FavCtrl: obtiene los favoritos de un usuario mediante un servicio que devuelve los que están almacenados localmente.

- profileCtrl.js: posee el controlador ProfileCtrl que se encarga de obtener la información de la cuenta de Facebook de un usuario, además de su listado de amigos.

- Factories: es una carpeta que contiene cuatro distintos factories que proveen una gama de servicios que son utilizados por los controladores, con el fin de ser reutilizables y genéricos.

  - actionFactory.js: es el Factory principal que provee servicios para la verificación de *token* con la API, hace uso de $http de AngularJS para comunicarse con la API mediante RESTful.

Contiene varios servicios como obtener supermercados y productos desde la API como productos favoritos localmente. Gestiona el almacenamiento y sustitución de *token* que devuelve la API.

  - FacebookFactory.js: es el Factory que utiliza Phonegap-Facebook-*Plugin* para comunicarse directamente con la API de Facebook. En este hay servicios para autenticación, obtención de perfil, obtención de amigos, y utilización de Show Dialogs para compartir estados y productos.

  - MapsFactory.js: provee servicios para renderizar Google Maps en la vista respectiva, también obtiene los distintos supermercados existentes en la API propia, para mostrar la ubicación de cada uno de los mismos en el mapa. Utiliza Cordova-Network-Information para verificar la conexión a internet.

- OfferFactory.js: provee servicios para la comunicación con la API para obtener supermercados, productos, *likes* entre otros que son solicitados por los controladores, que a su vez lo proveen a las vistas para ser mostrados.

- App.js: archivo principal de AngularJS que define todos los controladores y factories que se van a utilizar. Define además, las rutas y accesos a todas las funciones de la aplicación asociando un controlador a una única vista.

CONCLUSIONES

1. La construcción de APIs RESTful utilizando ActionHero simplifica la creación y permite tener un sistema escalable, además que mejora el rendimiento y no necesita hardware muy potente para funcionar.

2. El uso de máquinas virtuales en la nube es una opción que cada vez crece a mayor velocidad, debido a la reducción de costos y delega el soporte a terceros.

3. Herramientas para desarrollo híbrido, puede ser una forma de sustitución al desarrollo nativo convencional, ya que reducen tiempos y costos de desarrollo multiplataforma.

4. La combinación de APIs RESTful y aplicaciones híbridas ofrecen al usuario una serie de servicios que pueden ser beneficiosos para las tareas cotidianas, por lo que el estudio de nuevas tecnologías ayuda a crear mejores sistemas y experiencias para los usuarios.

RECOMENDACIONES

1. Para el desarrollo de APIs es importante elegir una herramienta especializada en ese tipo de proyecto, ya que la estructura es peculiar y afecta el rendimiento de la misma.

2. La seguridad en una API es fundamental, dado que se maneja una comunicación de dos vías. Por lo que se puede utilizar autenticación por *token* como medida de seguridad.

3. Para el manejo de proyectos de desarrollo; utilizar herramientas para versionar el código con el fin de controlar los cambios durante el desarrollo.

4. El diseño de una App móvil debe ser probado con anterioridad, ya que el flujo y aspecto son críticos para captar la atención del usuario.

5. Para el desarrollo de una App; tomar el tiempo necesario para preparar la arquitectura, ya que el sistema debe quedar abierto a mejoras, escalabilidad y buen manejo de concurrencia respecto de los usuarios que tenga.

BIBLIOGRAFÍA

1. ABERNETHY, Michael. *¿Simplemente qué es Node.js?* [en línea]. <https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/>. [Consulta: 29 de marzo de 2016].

2. ANGULARJS. *What Is Angular?.* [en línea].
< https://docs.angularjs.org/guide/introduction>. [Consulta: 7 de abril de 2016].

3. EXPÓSITO, Raúl. *Cómo Aprender AngularJS - Tu Sherpa de AngularJS.* [en línea]. < http://raulexposito.com/documentos/como-aprender-angularjs/>. [Consulta: 7 de abril de 2016].

4. IBM. *El desarrollo de aplicaciones móviles nativas, web o híbridas.* [en línea]. <ftp://ftp.software.ibm.com/la/documents/gb/commons/27754_IBM_WP_Native_Web_or_hybrid_2846853.pdf>. [consulta: 20 de marzo de 2016].

5. MYSQL. *What is MySQL?.* [en línea]. < http://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html>. [Consulta: 6 de abril de 2016].

6. PÉREZ ESTESO, Mario. *Cómo funciona HTTPS*. [en línea]. <https://geekytheory.com/como-funciona-https/>. [Consulta: 19 de marzo de 2016].

7. READ THE DOCS. *Documentación, conociendo GitHub.* [en línea] [ref. abril 2012]. < http://conociendogithub.readthedocs.org/en/latest/>. [Consulta: 6 de abril de 2016].

8. READ THE DOCS. *Sequelize* | *The Node.js / io.js ORM for PostgreSQL, MySQL, SQLite and MSSQL.* [en línea]. <http://sequelize.readthedocs.org/en/latest/>. [consulta 06 de abril de 2016].

9. SILES, Fernando. *Ionic, otro contendiente en la batalla de los frameworks para desarrollo de apps.* [en línea]. <http://www.genbetadev.com/desarrollo-aplicaciones-moviles/ionic-otro-contendiente-en-la-batalla-de-los-*framework*s-para-desarrollo-de-apps>. [Consulta: 29 de marzo de 2016].

10. SUBIRATS, Joan. *Qué es y para qué sirve el SDK.* [en línea]. <http://www.fandroides.com/que-es-y-para-que-sirve-el-sdk/>. [Consulta: 28 de marzo de 2016].

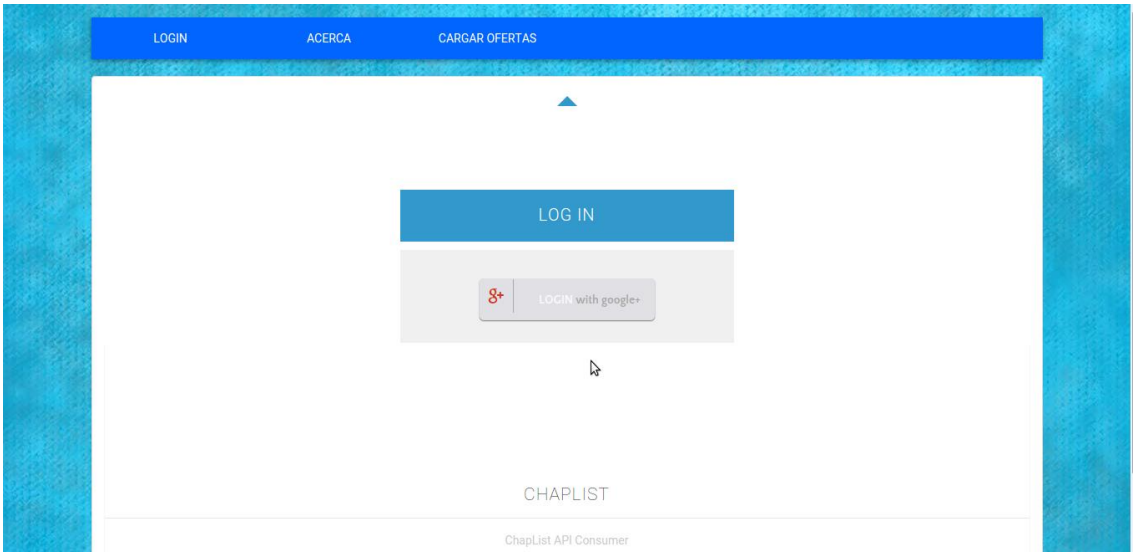
11. UMBARILA RUBIANO, Brayan Daniel. *Cloud Computing*. [en línea]. <http://www.fce.unal.edu.co/uifce/proyectos-de-estudio/pdf/Cloud_computing>. [Consulta: 19 de marzo de 2016].

APÉNDICES

Apéndice 1. Interfaz de registro API ChapList

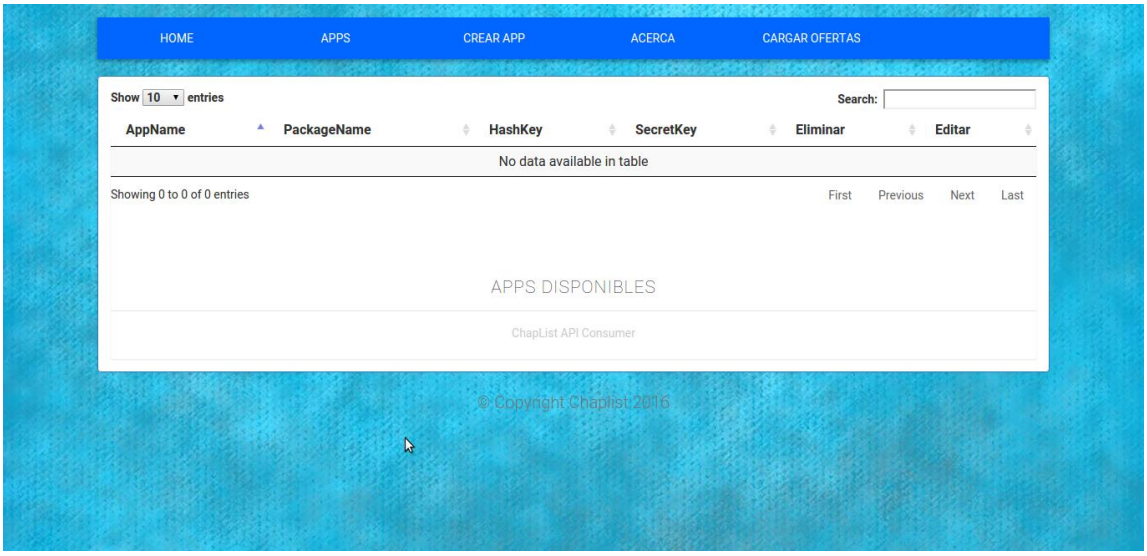
Para el registro de Apps que pudieran consumir la API se creó una plataforma de registro que se describe en las siguientes figuras.

Pantalla de ingreso a la plataforma

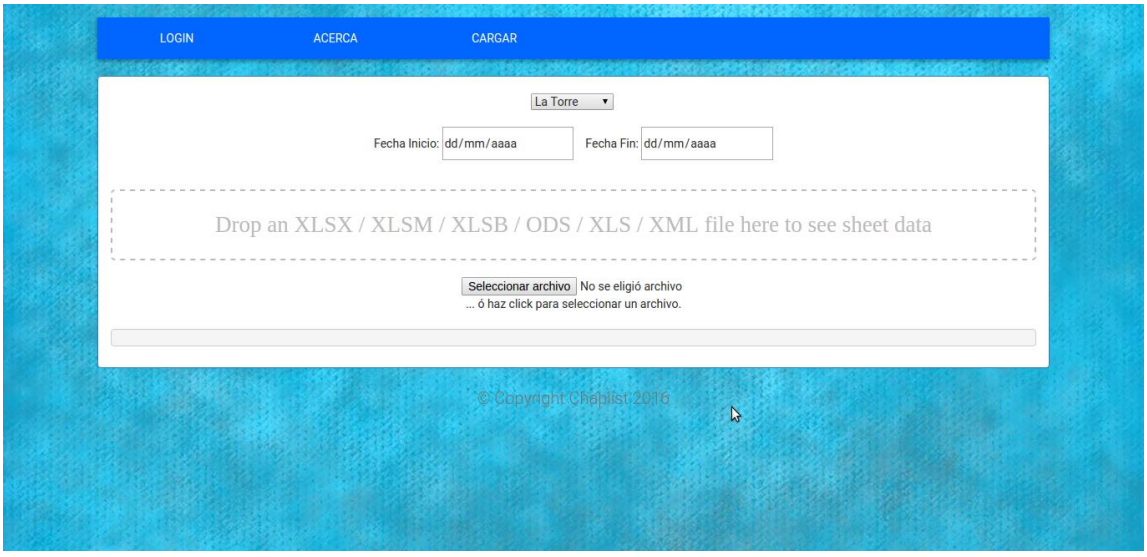


Continuación de apéndice 1.

Pantalla de gestion de Apps registradas que podrán consumir la API

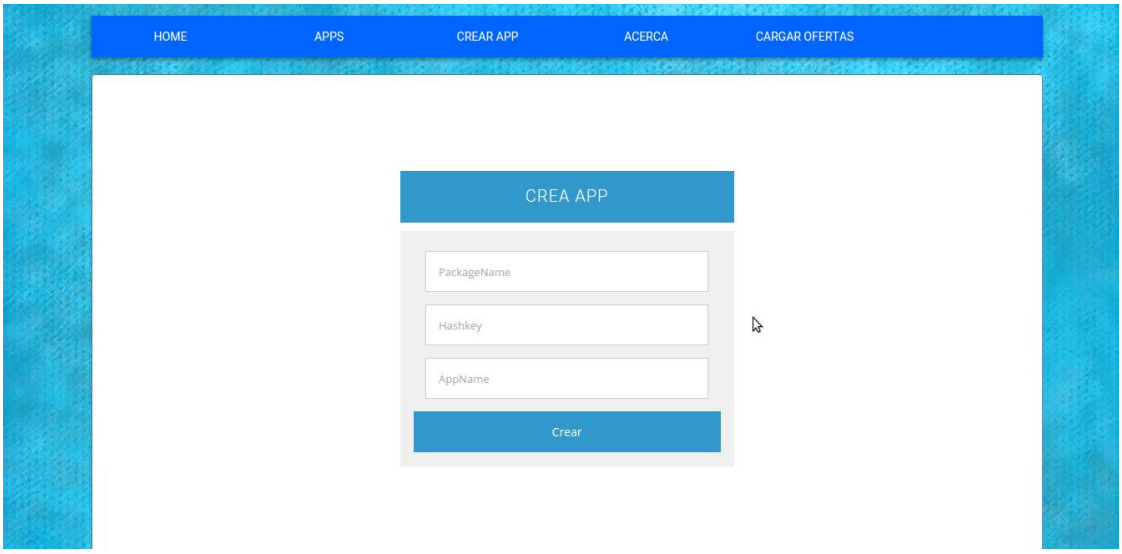


Pantalla de carga de ofertas



Continuación de apéndice 1.

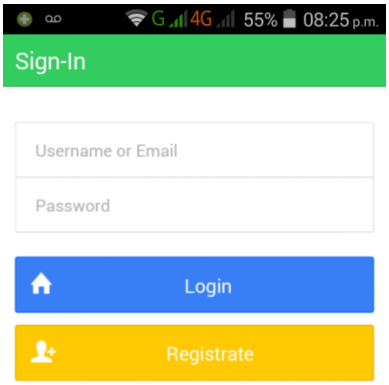
Pantalla de registro de Apps



Fuente: elaboración propia, empleando programa AngularJs.

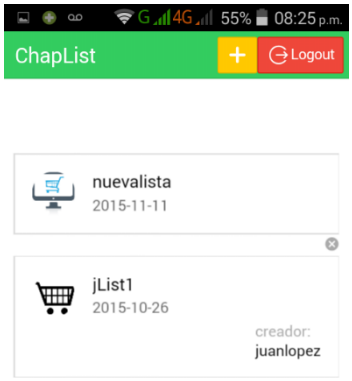
Apéndice 2. Prototipo inicial de ChapList

Pantalla inicial para autenticación o creación de usuarios

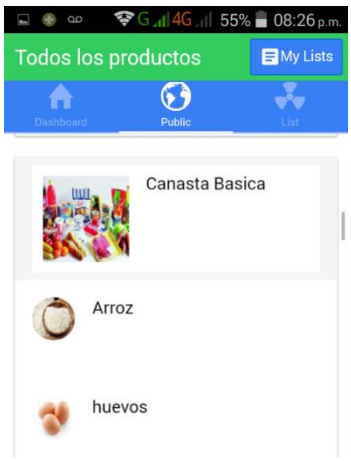


Continuación de apéndice 2.

Se muestra las listas que posee cada usuario en su respectiva sesión.



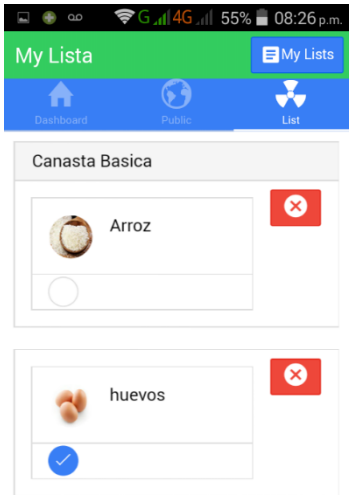
Pantalla que contiene el listado de productos generales.



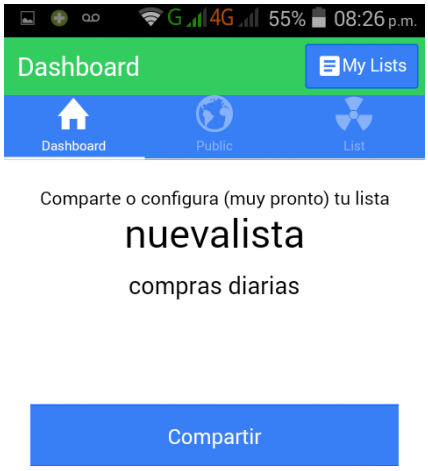
Para agregar un nuevo producto a la lista seleccionada, basta con presionar sobre dicho producto para que este se agregue automáticamente.

Continuación de apéndice 2.

Pantalla que contiene los productos seleccionados en una lista específica.

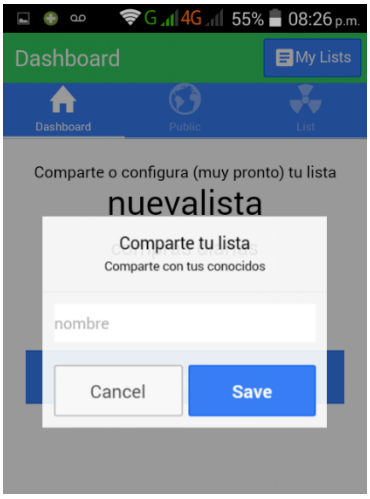


Pantalla para compartir la lista actual.



Continuación de apéndice 2.

Pantalla de ingreso de usuario para compartir una lista.



Fuente: elaboración propia, empleando programa Ionic.

ANEXOS

Anexo 1. Instalación de Phonegap Facebook *plugin* en Ionic

La instalación y desinstalación de distintos *plugin*s se realiza mediante la CLI de forma muy simple con las siguientes instrucciones.

- Estar dentro de la carpeta del proyecto Ionic
- $ cordova add "*nombre de plugin".* (Agregar *plugin*)
- $ cordova remove *"nombre de plugin".* (Eliminar *plugin*)
- $ cordova plubin list. (Muestra listado de *plugins* instalados)

Sin embargo, para la instalación del *plugin* de Phonegap para Facebook requiere de unos pasos previos para su uso.

- Estar dentro de la carpeta del proyecto Ionic

- $ git clone https://github.com/Wizcorp/phonegap-facebook-plugin.git. (Clonar repositorio del *plugin*)

- $ cordova -d plugin add phonegap-facebook-plugin --variable APP_ID="id de la app en la API de Facebook" --variable APP_NAME="nombre de la app en la API de Facebook"

Fuente: DABIF, *Dayana. How to add Facebook Native login to your Ionic app*. https://ionicthemes.com/tutorials/about/native-facebook-login-with-ionic-*framework*. Consulta: abril de 2016.

Anexo 2. Asignación de vista/controlador a estados en AngularJS

El uso de vistas dentro de una aplicación en AngularJS, requiere que se defina la tupla: vista/controlador y el estado en que se mostrará cada vista.

Dentro de cada archivo app.js que es el principal para un proyecto en AngularJS, se pueden definir las rutas utilizando ui-router. Esto hace que el manejo de rutas, vistas y controladores sea de forma dinámica y flexible.

Fuente: Angular, *UI Router: ui.router*. https://angular-ui.github.io/ui-router/site/#/api/ui.router. Consulta: mayo de 2016.