

UNIVERSIDAD CATÓLICA BOLIVIANA “SAN PABLO”

UNIDAD ACADÉMICA REGIONAL COCHABAMBA

Departamento de Ciencias Exactas e Ingenierías

Ingeniería de Sistemas



**Sistema basado en el modelo SOA para la oferta y venta
en línea de productos**

Proyecto de Grado de Licenciatura en Ingeniería de Sistemas

Alexander Peredo Torossova

Cochabamba – Bolivia

Junio 2016

Agradecimientos

*Agradezco a Lia, Erik, Nana, Isabela, Kim e Irina
por el apoyo durante estos años*

RESUMEN

El presente documento muestra el desarrollo de un sistema basado en el modelo SOA para la oferta y venta en línea de productos. El caso de estudio para este sistema es la Importadora C&H, la cual decidió ampliar su fuerza de ventas, viendo por conveniente hacer esto mediante un sistema web.

El desarrollo del sistema se hizo bajo la metodología *Scrum*, la cual permite adaptarse rápidamente a los cambios y requiere de un trabajo continuo con el cliente.

Además, el sistema se desarrolló basándose en el modelo SOA, el cual, mediante el uso de servicios web, como ser APIs, permite la creación de sistemas con gran capacidad de escalabilidad y adaptación al crecimiento de una empresa.

Una vez finalizado el desarrollo del sistema, se pudo concluir que los objetivos específicos fueron cumplidos satisfactoriamente, además que la elección de la metodología permitió adaptarse a cambios durante el desarrollo y se pudo evidenciar que el uso del modelo SOA permite escalar el sistema sin presentar mayores dificultades.

ABSTRACT

This document shows the development of a system based on the SOA model to offer and sell products online. The case study for this system is the importer C & H, which decided to expand its sales force, deciding the best way to do this was through a web system.

The development of the system was done under the Scrum methodology, which allows to adapt quickly to changes and requires continuous work with the client.

In addition, the system was developed based on the SOA model, which, through the use of web services, such as APIs, allows the creation of systems with high scalability and adaptation to the growth of a company.

Upon completion of the system, it was concluded that the specific objectives were met satisfactorily, in addition, the methodology chosen allowed to adapt to changes during development and was evident that the use of SOA model allows scaling the system without major difficulties.

ÍNDICE GENERAL

INTRODUCCIÓN	1
1. MARCO TEÓRICO	14
1.1. COMERCIO ELECTRÓNICO	14
1.1.1. Generalidades.....	14
1.1.2. Historia.....	15
1.1.3. Actualidad.....	15
1.1.4. Problemas con el Comercio Electrónico.....	16
1.1.5. Proyecciones Hacia el Futuro.....	18
1.1.6. Intercambio electrónico de datos	19
1.2. SOA.....	20
1.2.1. Definición.....	20
1.2.2. Principios	21
1.2.3. Capas de SOA.....	22
1.2.4. Heterogeneidad.....	23
1.2.5. Servicios Web	24
1.2.6. SOAP.....	25
1.2.7. REST.....	25
1.2.8. Arquitectura basada en microservicios.....	27
1.3. METODOLOGÍA DE DESARROLLO DE SOFTWARE.....	28
1.3.1. Selección de Metodología de Desarrollo.....	29
1.3.2. Scrum	32
1.4. TECNOLOGÍAS.....	39
1.4.1. Frameworks web	39
1.4.2. Ruby.....	42
1.4.3. Ruby On Rails.....	43
1.4.4. MySQL	45
1.4.5. EmberJS.....	46
1.4.6. Twilio.....	48
1.4.7. Apipie.....	48
2. INGENIERÍA DEL PROYECTO	49
2.1. PROPUESTA DE UN MODELO DE PROCEDIMIENTOS DE OFERTA Y VENTA PARA EL MERCADO BOLIVIANO A TRAVÉS DE INTERNET	49
2.1.1. Problemas encontrados.....	49
2.1.2. Propuesta del modelo.....	50
2.2. DESARROLLO DEL SISTEMA PARA LA VENTA EN LÍNEA DE PRODUCTOS	52
2.2.1. Product backlog.....	54
2.2.2. Sprint 1. Desarrollo de categorías y subcategorías	56
2.2.3. Sprint 2. Desarrollo de fabricante, proveedores y productos	59

<i>2.2.4. Sprint 3. Desarrollo del módulo de clientes.....</i>	<i>63</i>
<i>2.2.5. Sprint 4. Desarrollo del módulo de ventas.....</i>	<i>65</i>
<i>2.2.6. Sprint 5. Desarrollo del módulo de reportes.....</i>	<i>71</i>
CONCLUSIONES	73
RECOMENDACIONES	74
BIBLIOGRAFÍA	75
ANEXOS	77
ANEXO 1. ESTUDIO SOBRE EL COMERCIO ELECTRÓNICO EN LATINOAMÉRICA.....	77

ÍNDICE DE TABLAS Y FIGURAS

Tabla 1: Acciones y fundamentos teóricos para cada objetivo específico	9
Tabla 2: Tabla de ventajas y desventajas de metodologías.	30
Tabla 3: Product Backlog	54
Tabla 4: Navegadores y LocalStorage	70
Figura 1: Razones para no usar comercio electrónico	17
Figura 2: Zonas a las que pertenecen los usuarios de Internet	17
Figura 3: Porcentaje de crecimiento en el uso de comercio electrónico en el mundo	19
Figura 4: Implementación dependiente del estado	26
Figura 5: Implementación independiente del estado	27
Figura 6: Flujo de trabajo dentro de la metodología SCRUM	33
Figura 7: Popularidad de distintos frameworks	41
Figura 8: Flujo para confirmar un pedido mediante un mensaje de texto	51
Figura 9: Flujo para confirmar un pedido mediante foto de extracto bancario	52
Figura 10: Diagrama de contexto del sistema	53
Figura 11: Caso de uso para categorías	56
Figura 12: Casos de uso para subcategorías	56
Figura 13: Diagrama de clases para categorías y subcategorías	57
Figura 14: Diagrama de bases de datos de categorías y subcategorías	57
Figura 15: Casos de uso para proveedores	59
Figura 16: Casos de uso para fabricantes	60
Figura 17: Casos de uso para productos	60
Figura 18: Diagrama de clase de proveedores, fabricante y productos	61
Figura 19: Diagrama de bases de datos de proveedores, fabricante y productos	62
Figura 20: Diagrama de casos de uso para clientes	63
Figura 21: Diagrama de clases para clientes	64
Figura 22: Diagrama de clase de clientes	64
Figura 23: Diagrama de casos de uso para ventas	66
Figura 24: Diagrama de clases para ventas	66
Figura 25: Diagrama de clases para movimientos	67
Figura 26: Diagrama de bases de datos para ventas	68
Figura 27: Diagrama de base de datos para movimientos	69
Figura 28: Tabla utilizada para almacenar productos en LocalStorage	70
Figura 29: Diagrama de casos de uso para reportes	72

INTRODUCCIÓN

Actualmente muchas empresas en el exterior realizan sus ventas por medio de páginas web, donde además de ofertar y vender sus productos tienen una comunicación y retroalimentación con los usuarios.

En nuestro país el comercio en línea aún no tiene la magnitud que otros países muestran, pero paulatinamente las empresas e instituciones están optando por ofrecer este servicio, aunque el uso de tarjetas crédito para compras en línea aún no es aceptado por la mayoría de los usuarios, siendo el uso de este medio de pago tan solo el 2.2% del total de los métodos de pago en nuestro país, tal como se observa en Anexo 1. Sin embargo, es necesario resaltar el hecho de que a pesar de que la mayoría de los usuarios en Bolivia aun no confían en los pagos por Internet la gran mayoría si hace uso de Internet. Es por esta razón que se deben desarrollar otras maneras de realizar las transacciones.

Además, muchas organizaciones en el país aún tienen procesos y métodos para el control de sus operaciones que no llegan a satisfacer todas las necesidades de las mismas, por lo que muchas de estas empresas buscan adecuar sus procesos y representarlos en sistemas informáticos, sin embargo, existen casos donde los procesos presentan tantas falencias que la mejor solución tanto económica como operativa es reformular los procesos y posteriormente implementarlos en sistemas informáticos.

Cabe mencionar que la mayoría de los sistemas desarrollados para distintas empresas suelen empezar con un conjunto reducido de requerimientos, pero conforme avanza el tiempo y las empresas van creciendo también crecen los requerimientos de los sistemas, por lo que es necesario tomar en cuenta este potencial crecimiento a la hora de empezar a desarrollar los sistemas. El uso del modelo SOA (Service Oriented Architecture)

brinda soluciones eficaces a los puntos mencionados anteriormente, permitiendo realizar sistemas altamente escalables, reduciendo costos de implementación y permitiendo a los negocios poder comunicar distintos sistemas desarrollados en plataformas y lenguajes diferentes. Esta arquitectura es usada ampliamente en el ámbito del comercio en línea y permite tener separadas las unidades de negocio y los servicios requeridos.

Antecedentes

Antecedentes organizacionales

La empresa C&H Importaciones se dedica a la importación de material de construcción y jardinería. Esta empresa cuenta con dos unidades de negocio, una enfocada a la venta por mayor y otra que se dedica a la venta por menor de los productos importados.

Para ambas unidades de negocio la importadora adquiere sus productos principalmente de tres empresas, las cuales son:

- Amig, empresa española que se dedica a la venta de artículos de ferretería, bricolaje y suministros industriales.
- Coflex, empresa mexicana dedicada a la venta de conectores flexibles, brida flexible, trampas flexibles entre otros.
- Pabovi, empresa brasileña especializada en la venta de todo tipo de mangueras.

Estas tres empresas ofrecen productos de primera calidad, razón por la cual los precios que ofrece la importadora C&H están por encima del promedio del mercado nacional.

La primera unidad de negocio se enfoca en la venta al por mayor de sus productos, siendo sus principales clientes las empresas constructoras, ferreterías y casas de construcción.

La segunda unidad de negocio, llamada TodoConstrucción está enfocada en la venta de los productos importados, pero al por menor, por lo cual sus principales clientes son carpinteros, dueños de casa, cerrajeros, etc. Los precios de los artículos son distintos en ambas unidades de negocio.

Se cuenta con tres almacenes en Cochabamba donde ingresan los productos una vez importados. A mediano plazo la empresa contará con almacenes tanto en Santa Cruz como en Sucre. Cada almacén tiene distintos productos, ya que no pueden almacenarse artículos de plomería junto con material de construcción como cemento u otros. Lo mismo ocurre en el caso de las mangueras. Estos almacenes se encuentran en distintos puntos de la ciudad de Cochabamba.

Las ventas se realizan actualmente de tres maneras, las cuales son:

- Existen vendedores a nivel nacional que visitan los departamentos de La Paz, Sucre, Potosí y Santa Cruz, ofreciendo los productos en ferreterías, casas de construcción y otros negocios del rubro de la construcción y jardinería. Una vez que se logra conseguir un cliente, el vendedor envía a la empresa el requerimiento del cliente a través de correo electrónico o si es muy urgente para el cliente recibir los artículos rápidamente, el vendedor llama a la importadora, la cual prepara el pedido. El vendedor se encarga de realizar los cobros respectivos en el departamento que esté y a final de cada mes se realiza una rendición de cuentas entre la importadora y el vendedor. Se han dado casos en los cuales el vendedor ofrece una serie de productos al cliente, pero al enviar los

requerimientos a la importadora esta le comunica que los productos solicitados no están disponibles.

- Por otro lado, en Cochabamba, La Paz y Sucre, un vendedor recorre la ciudad en busca de ferreterías a las cuales ofrece los distintos productos. Si consigue un cliente, el vendedor se encarga de llevarle al mismo los artículos requeridos, recopila datos del cliente en hojas de papel, cobra el dinero y cierra la transacción. Debido a que algunos vendedores volvían frecuentemente diciendo que no pudieron encontrar clientes, se decidió instalar en los vehículos de los mismos un sistema de posicionamiento global para poder conocer el recorrido que hacían durante su jornada laboral. Este sistema permitió darse cuenta que existían vendedores que salían de la empresa y retornaban a sus hogares, donde pasaban el resto de la jornada laboral, luego volvían a la importadora e informaban que no pudieron conseguir clientes.
- También se cuenta con venta directa, por correo electrónico y teléfono, siendo estas dos últimas solo con clientes antiguos que ya conocen la empresa debido a que anteriormente ya realizaron compras en la misma.

Para la importadora existen dos formas de cobrar la venta de una gama de productos, el cobro al contado y a crédito. El cobro a crédito solo se realiza con clientes antiguos y además se toma en cuenta el comportamiento crediticio que han tenido. Los registros de los créditos se realizan de forma manual y en papel u hojas excel. Hubo casos en los cuales se le negó un crédito a un cliente que debería haber podido acceder al mismo y también se ha otorgado crédito a clientes que no tenían un buen historial crediticio.

Antecedentes tecnológicos

El comercio electrónico existe desde hace varios años, pero dentro de nuestro país su uso aun es escaso, ya que los usuarios prefieren usar los métodos de pago tradicionales como depósitos a bancos y pagos al recibir los productos (Anexo 1).

Existen empresas dentro del país que se dedican al comercio en línea, un ejemplo es la empresa TuMomo (<http://www.tumomo.com/>), la cual funciona como una página de clasificados virtual, donde los usuarios pueden publicar anuncios para ofrecer los productos que quieren vender, dejando sus datos personales como referencia para poder ser contactados posteriormente por los interesados en el anuncio publicado.

El publicar un anuncio en la página web de TuMomo tiene un costo de Bs. 25 por anuncio, pero también ofrecen la opción de obtener distintos tipos de membresías anuales.

Para realizar los pagos por concepto de publicación de un anuncio o para adquirir una membresía, TuMomo ofrece 3 formas, una es a través del portal PagosNet, el cual permite realizar los pagos en distintos bancos y empresas como FarmaCorp.

La segunda y tercera manera es realizar los pagos a través del banco de crédito BCP o banco mercantil Santa Cruz.

Otra organización dentro del país que realiza comercio en línea es NIC Bolivia (<https://www.nic.bo>), la cual ofrece nombres de dominio dentro del territorio nacional.

Para adquirir un dominio NIC Bolivia ofrece las siguientes opciones:

- Mediante depósito bancario al Banco Central de Bolivia o al Banco Unión.
- Mediante cheque al Banco Central de Bolivia o al Banco Unión.

- Mediante Banca por Internet del Banco Unión, la cual permite transferencias monetarias a cuentas de terceros dentro del mismo banco.
- Para instituciones públicas mediante transferencia SIGMA. Esta forma de pago requiere tener instalado el software de SIGMA, el cual es distribuido por el gobierno nacional y permite realizar distintos tipos de transferencias monetarias.

Fuera del país existen muchas más empresas que se dedican al comercio electrónico, como el caso de E-Bay (<http://www.ebay.com>) o Amazon (<https://www.amazon.com>). Estas empresas realizan las transferencias monetarias mediante el uso de tarjetas de crédito o a través de PayPal. El servicio de PayPal mediante bancos nacionales aún no existe, por lo que la única manera de depositar dinero en PayPal es a través de una tarjeta de crédito y como vimos anteriormente en el Anexo 1, el uso de tarjetas de crédito en nuestro país no supera el 2% de todas las transacciones monetarias.

Problema

A continuación, se mencionará cada uno de los problemas encontrados.

Situación Problemática

- La necesidad de vendedores que viajan por todo el país para ofrecer los productos genera un gasto considerable para la empresa ya que además de sus pasajes se debe pagar los viáticos del mismo.
- Debido a la distancia y continuo movimiento de los vendedores, la importadora no conoce a detalle si los mismos cumplen sus jornadas laborales y obligaciones con la empresa.

- Al enviar los pedidos por correo electrónico en la mayoría de los casos, la importadora no siempre prepara el pedido el día mismo que este se realizó lo que genera molestias en los clientes de la empresa debido a la urgencia de algunos pedidos.
- Los vendedores no tienen a mano el stock que la importadora maneja, lo que lleva al mismo a ofrecer productos que no siempre están disponibles.
- También sucede que los encargados de ventas muchas veces no conocen las preferencias y datos de clientes antiguos, por lo que vuelven a recopilar esta información, lo que lleva a inconsistencias en la información y ocasiona problemas a la hora de revisar el historial de los clientes.
- La venta de productos por correo electrónico y teléfono no es accesible para todos los clientes ya que al hacer compras de esta manera el cliente no cuenta con un listado de los productos, razón por la cual estas formas de venta están limitadas a cierto sector de los clientes que ya conocen los artículos que necesitan y que ofrece la importadora.
- El buscar en hojas de papel o dentro de archivos excel el historial crediticio de un cliente ha generado irregularidades a la hora de aceptar o rechazar un crédito a los mismos.
- Para poder realizar la verificación de deudas pendientes para un cliente, la importadora debe realizar la búsqueda de estos datos manualmente en los archivos físicos y Excel, lo cual ha ocasionado retrasos en cobros y malentendidos con los clientes.
- El uso de tarjetas de crédito en nuestro país al solo representar el 2% del total de las transacciones monetarias no es un método factible para realizar comercio en línea dentro del territorio nacional.
- La necesidad de interactuar directamente con el cliente; debido a que muchas veces los pedidos de material son urgentes, no permite el uso de métodos de pago por medio de terceros como bancos y otras empresas.

Formulación del problema

El porcentaje de población en Bolivia que utiliza actualmente tarjetas de crédito o PayPal como forma de pago para compras por Internet, la aplicación de métodos tradicionales¹ de oferta y venta de productos, y las dificultades que se encuentran en la logística de la empresa no permiten a la misma contar con una herramienta para gestionar y dar a conocer al público en general los productos que se ofrecen.

Objetivos

Objetivo General

Desarrollar un sistema basado en el modelo SOA para la oferta y venta en línea de productos.

Objetivos Específicos

A continuación, se mencionarán los objetivos específicos:

- Proponer un modelo que permita adecuar y/o reformular los procedimientos de la importadora para la oferta y venta de sus productos mediante un método factible en el mercado boliviano para el comercio a través de Internet.
- Desarrollar el módulo de control de stock de los productos.
- Desarrollar el módulo de control de la información sobre los clientes de la empresa.
- Desarrollar el módulo de ventas por Internet.

Metodos Tradicionales¹. En este contexto se definen como venta directa, venta puerta a puerta, venta por teléfono y venta por correo electrónico.

- Desarrollar el módulo de reportes.

Tabla 1: *Acciones y fundamentos teóricos para cada objetivo específico*

Objetivo Específico	Acciones	Fundamento Teórico
1. Proponer un modelo que permita adecuar y/o reformular los procedimientos de la importadora para la oferta y venta de sus productos mediante un método factible en el mercado boliviano para el comercio a través de Internet.	<ul style="list-style-type: none"> • Realizar entrevistas al personal encargado de las ventas en la importadora. • Analizar los datos que relacionan a los clientes y las ventas. • Realizar la captura de requerimientos de los clientes a la hora de realizar pedidos por Internet. Esto se realizará mediante entrevistas a potenciales clientes. • Realizar una propuesta para el comercio en línea. • Entrevistar a distintos empleados de la empresa en busca de problemas operativos en sus áreas respectivas. • Conjuntamente con los encargados de la empresa identificar las falencias en las operaciones. • Realizar una propuesta para los nuevos procesos a modificar. 	<ul style="list-style-type: none"> • Captura de requerimientos para el desarrollo de software

<p>2. Desarrollar el módulo de control de stock de los productos.</p>	<ul style="list-style-type: none"> • Capturar los requerimientos pertinentes a este módulo en base a los requerimientos obtenidos anteriormente. • Realizar prototipos de la interfaz de usuario. • Implementar la lógica de este módulo. • Diseñar e implementar las pruebas necesarias para asegurar el correcto funcionamiento de este módulo. • Implementar los mecanismos de intercambio de mensajes para la comunicación entre este módulo y futuros módulos. 	<ul style="list-style-type: none"> • Metodologías de desarrollo ágil. • Ingeniería de Calidad. • Sistemas Distribuidos. • Mecanismos de intercambio de mensajes para sistemas distribuidos. • Modelo SOA
<p>3. Desarrollar el módulo de control de la información sobre los clientes de la empresa.</p>	<ul style="list-style-type: none"> • Capturar los requerimientos pertinentes a este módulo en base a los requerimientos obtenidos anteriormente. • Conjuntamente con los encargados de la empresa seleccionar la información relevante para representar en el sistema a los clientes. • Realizar prototipos de la interfaz de usuario. • Implementar la lógica de negocio de este módulo. • Diseñar e implementar las pruebas necesarias para asegurar el correcto funcionamiento del módulo. • Implementar y probar los mecanismos de intercambio de mensajes para acoplar el módulo con futuros módulos 	<ul style="list-style-type: none"> • Metodologías de desarrollo ágil. • Ingeniería de Calidad. • Sistemas Distribuidos. • Mecanismos de intercambio de mensajes para sistemas distribuidos. • Modelo SOA.

<p>4. Desarrollar el módulo de ventas por Internet.</p>	<ul style="list-style-type: none"> • Capturar los requerimientos pertinentes a este módulo en base a los requerimientos obtenidos anteriormente. • Diseñar prototipos de la interfaz de usuario. • Implementar los mecanismos de intercambio de mensajes para obtener información tanto del módulo de clientes como del módulo de stock de productos. • Diseñar e implementar las pruebas para confirmar que el intercambio de información entre los módulos es correcto. • Implementar la lógica de negocio de este módulo. • Diseñar e implementar las pruebas para confirmar el correcto funcionamiento de la lógica de negocio de este módulo. • Implementar los mecanismos para intercambio de mensajes con módulos futuros. 	<ul style="list-style-type: none"> • Metodologías de desarrollo ágil. • Ingeniería de Calidad. • Sistemas Distribuidos. • Mecanismos de intercambio de mensajes para sistemas distribuidos. • Modelo SOA.
<p>5. Desarrollar el módulo de reportes.</p>	<ul style="list-style-type: none"> • Identificar conjuntamente a los encargados de la empresa que reportes son necesarios. • Implementar los reportes identificados anteriormente. • Diseñar e implementar las pruebas necesarias para asegurar el funcionamiento correcto de este módulo. 	<ul style="list-style-type: none"> • Metodologías de desarrollo ágil. • Ingeniería de Calidad. • Sistemas Distribuidos. • Mecanismos de intercambio de mensajes para sistemas distribuidos. • Modelo SOA. • API de Google para generación de gráficas.

Fuente: Elaboración propia.

Alcances

A continuación, se mencionarán los alcances del sistema:

- Mediante cuentas de usuario, los distintos encargados de las áreas de la empresa podrán acceder al sistema distribuido para realizar las acciones pertinentes a su área y acceder a la información necesaria para sus tareas.
- El sistema permitirá ingresar y actualizar continuamente la información de los productos en stock.
- Se podrá crear, editar, actualizar y borrar la información necesaria de los clientes, para tener un registro adecuado de estos, además contara con datos sobre sus preferencias respecto a los productos que adquieren.
- El sistema permitirá recibir los pedidos, confirmarlos y acceder a la información de los clientes para poder entregar sus pedidos.
- El sistema ofrecerá datos sobre el tráfico que recibe módulo de ventas por Internet, brindando analíticas y métricas.
- El sistema se adaptará a distintas resoluciones de pantalla, para que este pueda ser accedidos desde computadoras de escritorio, computadoras portátiles y distintos dispositivos móviles como celulares y tablets.

Límites

El sistema desarrollado no contará con servicios para el control de inventario, solo control de stock, además que no se realizará un control de la parte contable (Impuesto a las transacciones, IVA, etc.) en el módulo de ventas. Además, el sistema requerirá de acceso continuo a Internet para poder funcionar.

Justificación

Justificación Técnica

La utilización de SOA permitirá contar con un sistema distribuido fácilmente escalable,

en el cual será posible no desechar sistemas ya existentes, sino comunicarlos con el nuevo sistema. También sistemas a ser desarrollados en el futuro podrán ser acoplados al sistema distribuido.

Tampoco existirán restricciones respecto a los lenguajes de programación y las tecnologías que se usen para desarrollar los sistemas específicos a cada tarea, lo que permitirá en un futuro inclusive comunicar el sistema distribuido con aplicaciones móviles.

También es necesario resaltar el hecho de que el mantenimiento del sistema distribuido será más sencillo, ya que cada servicio podrá ser mantenido de manera separada.

Justificación Económica

La implementación de este sistema permitirá un mejor control de la oferta, venta y distribución de productos, además que permitirá a las empresas mostrarse al público en general, algo que, si las empresas quisieran lograr sin el uso de un sistema adecuado, requerirían contratar una cantidad considerable de personal para ir a ofrecer sus productos, lo que implicaría gastos en sueldos, viáticos, viajes y otros.

Justificación Social

El sistema distribuido ofrecerá a las empresas una manera moderna, eficaz y altamente escalable de manejar su oferta, distribución y venta de productos, lo que permitirá a las mismas enfocarse más en otros aspectos que permitan el crecimiento sostenido a lo largo del tiempo, siendo el sistema distribuido la herramienta que les permita dar el siguiente salto hacia un uso más adecuado de la tecnología.

Además, los clientes tendrán un medio seguro y confiable para realizar compras, permitiendo a los mismos ahorrar tiempo y tener la certeza de que recibirán un servicio

de calidad.

1. MARCO TEÓRICO

1.1. Comercio Electrónico

1.1.1. Generalidades

El comercio electrónico es la compra y venta de productos o servicios a través de redes de computación como por ejemplo el Internet. Inicialmente el comercio electrónico se refería a transacciones e intercambio electrónico de datos, pero a mediados de los años 90 debido al crecimiento masivo de Internet y *World Wide Web* (WWW) se empezó a usar el término de comercio electrónico para la venta de servicios y productos usando como forma de pago medios electrónicos como las tarjetas de crédito.

El comercio electrónico ha crecido de manera extraordinaria desde su concepción, lo que ha llevado a que prácticamente cualquier bien o servicio pueda ser adquirido a través de Internet.

1.1.2. Historia

A pesar de no ser un dato comprobado se presume que los primeros indicios de comercio electrónico se dieron entre 1971 y 1972. Alumnos del instituto de inteligencia artificial de Stanford y alumnos del instituto de tecnología de Massachusetts a través de la ARPANET organizaban venta de cannabis (Wikipedia, 2016).

En 1979 Michael Aldrich mostro al mundo el primer sistema de comercio electrónico. Este consistía en una televisión conectada a través de una línea telefónica a una computadora que procesaba transacciones en tiempo real. Michael Aldrich llamó a este sistema *Teleshopping*. A partir de este sistema surgieron los primeros sistemas *business-to-business* como Thomson Holiday en Gran Bretaña o Minitel en Francia (Aldrich, 2011).

En abril de 1984 *CompuServe* lanzó al mundo el primer servicio comercial masivo en Estados Unidos. Este servicio fue lanzado casi de manera clandestina y fue utilizado inicialmente en la cadena de tiendas *Radio Shack*. A partir de este punto el comercio electrónico empieza a expandirse, por ejemplo, en 1992 *Books Stacks Unlimited* puso en línea el primer sitio de ventas en línea (www.books.com) que usó tarjetas de crédito como forma de pago. Además, en 1994 *Ipswithc IMail Server* se convirtió en el primer software disponible para comprar y descargar inmediatamente a través de Internet (Wikipedia, 2016).

1.1.3. Actualidad

Hoy en día el comercio electrónico es mundial, puede ser accedido desde prácticamente cualquier lugar del mundo y genera sumas de dinero muy grandes. En 2013 el departamento de comercio electrónico de Estados Unidos informó que ese año se generó un movimiento de 294 mil millones de dólares solo en el territorio norteamericano (Wikipedia, 2016).

El comercio electrónico actualmente no solo se usa para la compra de productos o bienes, también es usado para acceder a servicios, por ejemplo, Spotify, servicio electrónico para escuchar música por internet, el cual permite el uso de transacciones electrónicas para acceder a contenido *Premium*, algo que muchas otras empresas también proveen. Lo mismo ocurre con servicios de Microsoft como Azure, que permite acceder a servidores virtuales después de realizar una transacción en línea.

Otro ámbito donde se utiliza el comercio electrónico es en casas de apuestas deportivas, donde los usuarios pueden realizar distintos tipos de apuestas en prácticamente cualquier deporte.

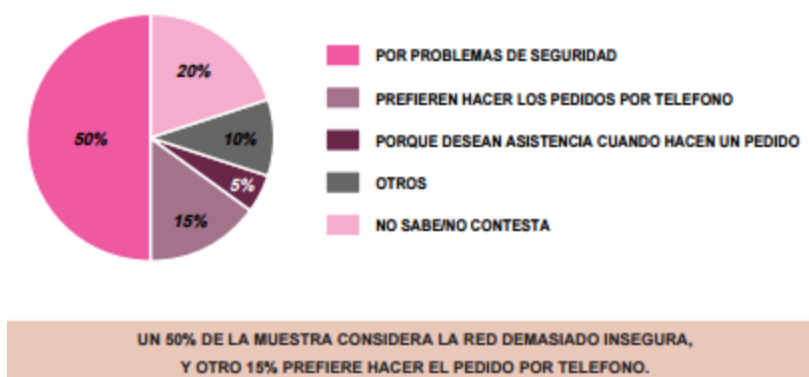
En cuanto a la adquisición de productos a través de Internet hoy en día se cuentan con muchas páginas web que ofrecen este servicio, como ser Ebay, Alibaba, Amazon y otros. Estas páginas web permiten a los usuarios seleccionar productos para poder adquirirlos a través de Internet, usando como métodos de pago distintos tipos de tarjetas, como ser tarjetas de crédito o tarjetas de débito, las cuales realizan los cobros de manera inmediata.

1.1.4. Problemas con el Comercio Electrónico

A pesar que el comercio electrónico es usado de manera masiva en todo el mundo aún existen personas e instituciones que no confían totalmente en esta forma de realizar transacciones. Las razones para desconfiar de este servicio pueden ser muchas, como por ejemplo el temor a ser víctima de un fraude, la preferencia de algunas personas de tener un contacto directo con el vendedor y otras. En la siguiente figura podemos observar algunos de estos motivos:

Figura 1: Razones para no usar comercio electrónico

¿Por qué no se realizan compras por INTERNET?



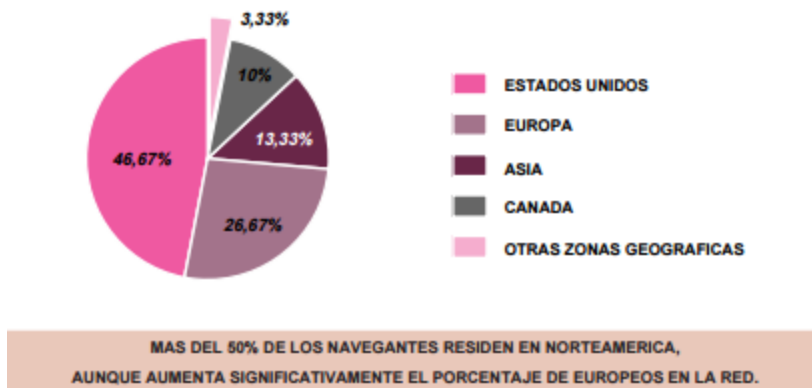
Fuente: (Teresa Ferreiro, 2008)

Como se observa en los resultados el 50% argumentaron que no realizan compras por Internet debido a problemas de seguridad y el 15% porque desean asistencia al hacer un pedido.

También se sabe que la mayoría de las transacciones electrónicas en el mundo son realizadas en Norteamérica y que en países en vías de desarrollo no suelen usarse de manera masiva las tarjetas de crédito, las cuales son el principal medio para la compra y venta por Internet. La siguiente figura nos muestra además el porcentaje de los usuarios de Internet por zonas geográficas:

Figura 2: Zonas a las que pertenecen los usuarios de Internet

¿A qué zonas pertenecen mayoritariamente los usuarios?



Fuente: (Teresa Ferreiro, 2008)

En la gráfica podemos ver que más del 50% de los usuarios de Internet se encuentran en Norteamérica y casi el 30% se encuentran en Europa, lo que deja alrededor de un 20% entre Asia, África y Latinoamérica. Esto podría explicar porque el comercio electrónico en Latinoamérica no tiene la relevancia que el mismo tiene en Europa y Norteamérica.

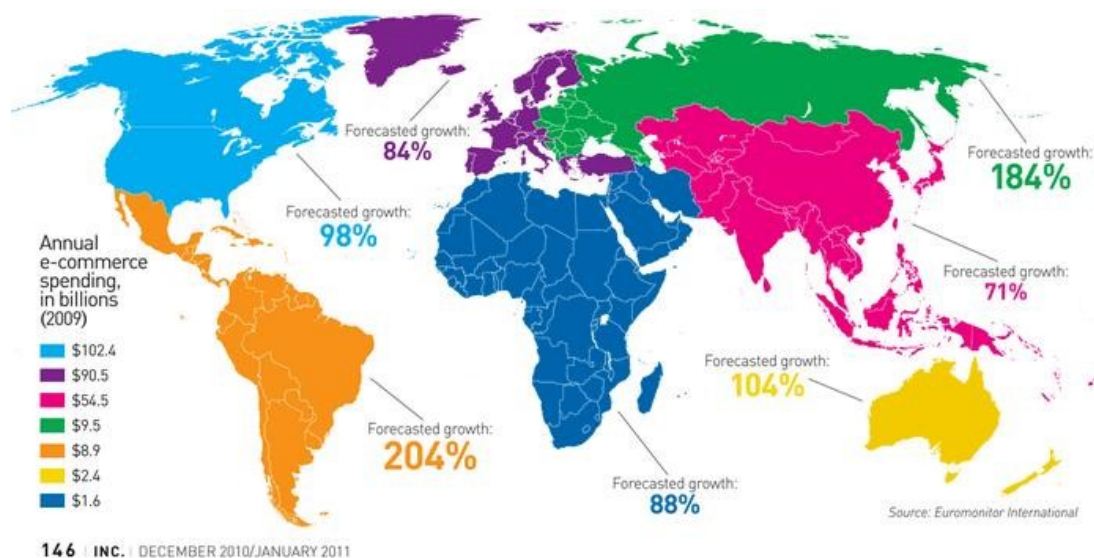
Otro factor que influye en el uso de comercio en línea es el uso de tarjetas de crédito, principal medio para realizar transacciones electrónicas. En Sudamérica el uso de estas aun es reducido, especialmente en países como Ecuador, Perú, Bolivia, Paraguay y Uruguay.

Específicamente en Bolivia solo el 2% de las transacciones comerciales son realizadas con tarjetas de crédito. (Anexo 1) y el uso de PayPal en el país es solo posible mediante tarjetas de crédito ya que los bancos dentro del territorio nacional no cuentan con acuerdos con PayPal.

1.1.5. Proyecciones Hacia el Futuro

A pesar de haber visto que el comercio electrónico en Sudamérica y especialmente Bolivia no es tan masivo como en otras zonas geográficas del mundo, las proyecciones son favorables e indican que en un futuro próximo el uso de esta forma de realizar transacciones llegará a los mismos niveles que en otros países.

Figura 3: Porcentaje de crecimiento en el uso de comercio electrónico en el mundo



Fuente: (Shivers, 2011)

1.1.6. Intercambio electrónico de datos

El intercambio electrónico de datos es un mecanismo de intercambio de información que hace posible la transmisión de datos comerciales entre los sistemas de información de una organización (Wikipedia, 2015).

La importancia del intercambio electrónico de datos en este proyecto surge cuando se observa las características de los clientes de la importadora. Aunque la importadora también apunta a la venta por menor de sus productos a clientes minoristas, la mayor parte de sus compradores son empresas con las que se ha tenido una relación prolongada

en el tiempo, creando confianza entre la importadora y estas, por lo cual los pagos se realizan en efectivo, algo que permite plantear una forma de comercio electrónica que no tenga que involucrar el uso de tarjetas de crédito, sino otro tipo de estrategias donde se puede evitar casos de fraude (pedidos de clientes inexistentes), pero que no se restrinja al cliente a contar con una tarjeta de crédito o un medio de pago electrónico.

Justamente esto es lo que se plantea con el intercambio electrónico de datos, una herramienta para que los clientes brinden toda la información respectiva, un intercambio de información, de modo que la importadora utilice estos datos para brindar sus servicios de manera más accesible para el mercado.

1.2. SOA

La arquitectura orientada a servicios, SOA por sus siglas en inglés (*Service Oriented Architecture*) es un patrón arquitectónico para desarrollar sistemas distribuidos. SOA fue creada para permitir a satisfacer las necesidades de negocio de manera flexible con los procesos de negocio. SOA permite reducir costos de implementación y además se adapta a cambios, permitiendo una reacción temprana ante la competitividad.

Esta arquitectura es especialmente utilizada para desarrollar sistemas grandes ya que tiene beneficios como el uso de distintos lenguajes de programación y tecnologías para desarrollar distintas partes del sistema global.

Un punto que debe quedar claro al momento de hablar de SOA es que esta no es una tecnología, es una arquitectura en la que pueden aplicarse distintas tecnologías.

1.2.1. Definición

Existen 2 definiciones formales de SOA, una ha sido creada por el grupo OASIS² y la otra por Open Group³.

La definición de OASIS es:

“La arquitectura orientada a servicios es un paradigma para organizar y utilizar componentes distribuidos que pueden estar bajo distintos dominios.” (Oasis Group, 2006)

La definición de Open Group es:

“La arquitectura orientada a servicios es un estilo arquitectural que soporta la orientación a servicios. La orientación a servicios es una manera de pensar en términos de servicios y desarrollo basado en servicios”. (The Open Group, 2015)

1.2.2. Principios

Aunque no existen estándares en relación con la composición exacta de la arquitectura orientada a servicios muchas fuentes de la industria han ido publicando distintos principios, entre los cuales se encuentran:

- Contrato de servicios estandarizados: los servicios adhieren a un acuerdo de comunicación, según se define en conjunto con uno o más documentos de descripción de servicios.
- Acoplamiento débil de sistemas: los servicios mantienen una relación que minimiza las dependencias y sólo requiere que mantengan un conocimiento de uno al otro.

² Organization for the Advancement of Structured Information Standards (OASIS) es un consorcio global que trabaja en el desarrollo y adopción de estándares de seguridad, energía, tecnología y otros.

³ Open Group es un consorcio dedicado a brindar servicios de estrategia, administración, innovación, investigación, estándares, certificaciones y desarrollo de tests.

- Abstracción de servicios: más allá de las descripciones del contrato de servicios, los servicios ocultan la lógica a los demás.
- Reutilización de servicios: la lógica se divide en servicios con la intención de promover la reutilización.
- Autonomía de servicios: los servicios tienen control sobre la lógica que encapsulan, desde una perspectiva de diseño y ejecución.
- La normalización de servicios: los servicios se descomponen a un nivel de forma normal para minimizar la redundancia. En algunos casos, los servicios se desnormalizan para fines específicos, como la optimización del rendimiento, el acceso y agregación.
- Optimización de servicios: los servicios de alta calidad son preferibles a los de baja calidad.
- Transparencia de ubicación de servicios: se refiere a la capacidad de un consumidor de servicios para invocar a un servicio independientemente de su ubicación en la red. Esto también reconoce la propiedad de descubrimiento (uno de los principios fundamentales de SOA) y el derecho de un consumidor para acceder al servicio. A menudo, la idea de la virtualización de servicios también se refiere a la transparencia de ubicación. Aquí es donde el consumidor simplemente llama a un servicio lógico, mientras que un SOA habilita la ejecución del componente de la infraestructura, normalmente un bus de servicios, que mapea este servicio lógico y llama al servicio físico.

1.2.3. Capas de SOA

SOA cuenta con cinco capas horizontales y cuatro capas verticales cuyo objetivo es apoyar las capas horizontales (The Open Group, 2015).

Las cinco capas horizontales son:

- Interfaz del consumidor: esta capa tiene las interfaces gráficas para el usuario final o las aplicaciones que utilizarán los servicios.
- Procesamiento del negocio: en esta capa se encuentran los servicios orquestados de manera que representen la lógica del negocio.
- Servicios: capa donde se encuentran los servicios como tal.
- Componentes de los servicios: se puede encontrar en esta capa los componentes utilizados para construir los servicios, como por ejemplo librerías.
- Sistemas operacionales: capa en la que reside los modelos de datos y bases de datos.

Las cuatro capas verticales son (Wikipedia, 2015):

- Integración: en esta capa se encuentra la integración de datos e integración de servicios.
- Calidad del servicio: capa en la que se encuentran los mecanismos de seguridad, disponibilidad, rendimiento y otros.
- Informativa: dentro de esta capa se provee información del negocio.
- Gobernanza: en esta capa se encuentra la estrategia de la tecnología de información.

1.2.4. Heterogeneidad

Durante muchos años el mayor problema entre sistemas grandes fue la integración entre los mismos. Para solucionar esto se manejó por mucho tiempo la idea de eliminar la heterogeneidad y tratar de llegar a un estándar que permita a todos desarrollar sistemas de forma que estos puedan comunicarse utilizando las mismas tecnologías, pero este enfoque no llegó a funcionar y es en este punto donde SOA provee una solución eficaz a este problema.

Al usar servicios, SOA permite pasar información entre los sistemas utilizando archivos JSON, XML y otros. Esto permite tener una heterogeneidad en cuanto a los componentes de los sistemas. En pocas palabras, SOA no ve como un problema la heterogeneidad, sino como un atributo que es parte del desarrollo.

1.2.5. Servicios Web

El termino servicios web designa una tecnología que permite que las aplicaciones se comuniquen de una forma que no dependan de la plataforma ni del lenguaje de programación (IBM, 2016). Un servicio web es una interfaz de software que describe un conjunto de operaciones a las cuales se puede acceder por la red.

Los servicios web están basados en un grupo de estándares que cubren la interoperabilidad de los sistemas. Estos estándares definen tanto los protocolos a ser usados como para la comunicación y el formato de las interfaces que serán usadas para los servicios.

Los estándares más utilizados son:

- XML: es usado para describir tipos de datos, modelos y formatos.
- JSON: es usado de la misma forma que XML.
- HTTP y HTTPS: es usado para mandar servicios web a través de Internet o redes de comunicación.
- WDSL: utilizado para describir las interfaces de los servicios, especialmente dos puntos, la firma (nombre y parámetros) y los detalles de publicación como ser los protocolos y la localización del servicio.
- UDDI: es un estándar para administrar los servicios web. Permite registrarlos y encontrarlos.
- SOAP: es un protocolo estándar para el intercambio de mensajes entre aplicaciones.

Muchos de estos servicios web pueden ser usados de manera conjunta por lo que el desarrollo de este proyecto será realizado siguiendo el estándar REST, debido a las ventajas que este ofrece en la simplicidad de la implementación, además que es un estándar que al ser utilizado con SOA ha mostrado buenos resultados.

1.2.6. SOAP

SOAP es un protocolo para el intercambio de información en un ambiente descentralizado y distribuido. Utiliza interfaces de servicios para exponer la lógica de negocio, a diferencia de REST que realiza esto mediante URIs (Javatpoint, 2015).

Una característica importante de SOAP es que solo permite el formato XML para representar datos.

Para el desarrollo de este proyecto se vio por conveniente no usar SOAP, ya que requiere mayor ancho de banda y recursos en el sistema para funcionar de manera óptima, además que define estándares que deben ser seguidos de manera muy estricta.

1.2.7. REST

REST (*Representational State Transfer*) define una serie de principios arquitectónicos enfocados en acceder a recursos de un sistema y enviarlos a través de algún protocolo por una red para que estos recursos puedan ser accedidos por otros sistemas (IBM, 2015).

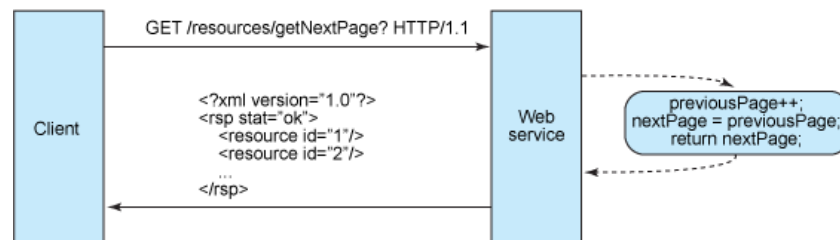
Aunque REST no depende de ningún protocolo en específico, hoy en día es prácticamente un estándar en la industria de software el uso de HTTP junto con REST, pero también puede utilizarse REST con SOAP.

REST fue presentado al mundo en el año 2000 por Roy Fielding en la universidad de California y a partir de entonces el crecimiento en su uso ha crecido de manera muy rápida, llegando inclusive a convertirse en una parte integral de Java6 a través de JSR-311⁴.

Una implementación de un servicio web REST sigue los siguientes cuatro principios (Fielding, 2000):

- Uso del estándar HTTP. Las convenciones de HTTP deben ser cumplidas, lo que significa que para crear un recurso en el servidor se debe usar POST, para obtener un recurso debe usarse GET, para actualizar un recurso se utiliza PUT y para eliminar un recurso se hace uso de DELETE.
- No tener estado. Esto se refiere a que un servicio REST no debe almacenar el contexto. Para entender mejor este punto a continuación se puede observar dos figuras que muestran una implementación dependiente del estado y otra que no.

Figura 4: Implementación dependiente del estado

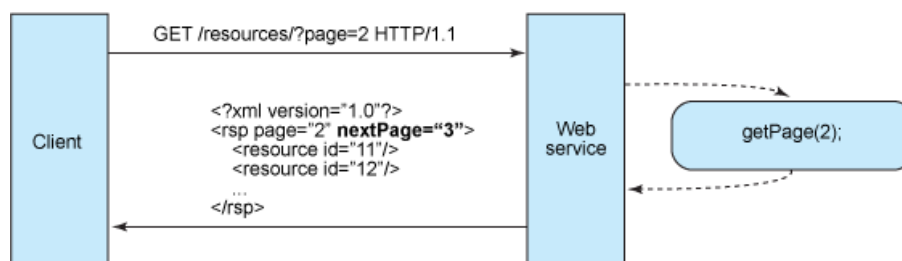


Fuente: (Rodriguez, 2015)

En la figura 4 podemos ver que cuando un cliente solicita la siguiente página a un servicio web que depende del estado, este debe incrementar la página actual, almacenar este incremento en una variable nueva y finalmente retornar la siguiente página.

⁴ JSR-311 es la API de Java para servicios web RESTful.

Figura 5: Implementación independiente del estado



Fuente: (Rodriguez, 2015)

Por otro lado, una implementación de un servicio web que no depende del estado recibe la petición de un cliente para ir directamente a la página 2 y el servicio retorna esto, sin depender de ningún estado.

- Estructurar las URIs en forma de directorios. Este es un punto muy importante en REST, ya que determinará la simplicidad del servicio web. Las URIs en lo posible deben poder ser simples y descriptivas, un ejemplo de una URI correctamente formada para REST es:

`www.myservice.org/persons/1`

Esta URI retornará una persona con ID 1. Como se puede ver la estructura es simple y además es posible para un desarrollador deducir que al acceder a `www.myservice.org/persons` se obtendrá un listado de personas.

- Transferir XML, JSON o ambos. Otro punto muy importante en REST es la forma en la que los recursos serán enviados al cliente y para esto REST indica que deben ser enviados en formato XML, JSON o ambos.

1.2.8. Arquitectura basada en microservicios

La arquitectura basada en microservicios es un estilo arquitectónico de software en el cual aplicaciones complejas se dividen en procesos pequeños e independientes, los cuales se comunican entre sí mediante APIs.

Esta arquitectura se basa en las siguientes propiedades:

- Los servicios son fáciles de remplazar.
- Se organizan los servicios en cuanto a su competencia, por ejemplo, interfaz de usuario, logística, etc.
- Los servicios pueden ser implementados utilizando distintos lenguajes de programación, bases de datos y ambientes de desarrollo.

Aunque en esencia esta arquitectura se parece bastante a SOA, debe quedar claro que ambas arquitecturas no son lo mismo, ya que ambas buscan solucionar problemas distintos. SOA se enfoca en cambiar toda la estructura de tecnologías de información de una organización, en cambio los microservicios buscan estructurar una sola aplicación de manera que esta haga uso de servicios (Fowler, y otros, 2014).

1.3. Metodología de Desarrollo de Software

Para que un sistema tenga éxito y se obtenga al final del ciclo de desarrollo un producto de calidad que cumpla con todos los requerimientos de los usuarios, la elección de una metodología de desarrollo adecuada es fundamental.

Una metodología de desarrollo es la segmentación o agrupación de tareas a la hora de desarrollar un sistema en distintas fases que contienen varias actividades, todo en pos de tener una planificación y administración del proyecto lo más óptima posible.

Aunque existen muchas metodologías de desarrollo en la actualidad las metodologías ágiles son las más utilizadas en la industria, esto debido a la relativa simplicidad en aprenderlas y que la aplicación correcta de estas permite sortear distintos obstáculos comunes a la hora de desarrollar un sistema.

Debido a las limitaciones de tiempo y la falta de requerimientos claros de parte del usuario para el desarrollo de este proyecto se vio por conveniente utilizar una metodología de desarrollo ágil.

1.3.1. Selección de Metodología de Desarrollo

Las metodologías de desarrollo ágil se basan en procesos iterativos e incrementales, donde la interacción continua entre el usuario y el equipo de desarrollo es fundamental para el éxito del proyecto.

Aunque las críticas hacia las metodologías ágiles están dirigidas a la poca importancia que estas le dan a la documentación de los sistemas, la realidad es posible incluir más documentación de la que se detalla en cada metodología ya que estas son en realidad marcos de trabajo referenciales, que pueden ser modificadas para adaptarse al requerimiento de los clientes y de los proyectos. Por esta razón en este proyecto la documentación será más extensa de lo que las metodologías mínimamente requieran.

Una característica fundamental de estas metodologías ágiles es que al terminar cada iteración el usuario recibe un sistema capaz de ser usado, obviamente en las etapas tempranas del desarrollo estos sistemas serán reducidos e irán expandiéndose a lo largo de todo el ciclo de desarrollo hasta cumplir los requerimientos.

Algunas metodologías ágiles con las que es posible desarrollar este proyecto son (Association of Modern Technologies Professionals, 2015):

- *Scrum*. La calidad del sistema está basada en el conocimiento y experiencia de los desarrolladores y los clientes. Esta metodología permite adaptarse a cambios durante el ciclo de desarrollo y es especialmente útil al no tener requerimientos claros de parte del cliente.
- Programación Extrema. Se caracteriza por la programación en pares, donde dos programadores comparten un equipo y trabajan de manera conjunta, además que al final del ciclo de desarrollo todos los programadores han trabajado por lo menos en una pequeña parte de cada componente del sistema, lo que garantiza que el equipo de desarrollo conozca de manera más amplia el sistema.
- *Crystal*. Es una de las metodologías más livianas y adaptables a distintos proyectos. Aunque cuenta con ciertas características comunes con otras

metodologías ágiles; como iteraciones, entregas de software funcional y una alta comunicación con el usuario, aborda cada proyecto de manera diferente, adaptándose al tipo de sistema, el tamaño del equipo de desarrollo y además durante todo el ciclo se va modificando la metodología para adaptarse y mejorar según el proyecto.

Tabla 2: Tabla de ventajas y desventajas de metodologías.

METODOLOGIA	VENTAJAS	DESVENTAJAS
Scrum	<p>Suele ayudar a las organizaciones a reducir tiempo y costos de desarrollo.</p> <p>Desarrollo continuo e incremental permite encontrar errores de manera más rápida.</p> <p>Reuniones continuas del equipo de desarrollo permiten estar siempre al tanto del estado del proyecto.</p> <p>Debido a iteraciones cortas resulta más fácil lidiar con cambios.</p>	<p>Si una tarea o especificación no es bien definida la estimación de costo y tiempo no será adecuada.</p> <p>Funciona mejor con equipos de trabajo pequeños.</p> <p>Se requiere miembros con experiencia en la metodología.</p>
	Adaptable a cambios sin mayores inconvenientes.	La falta de documentación o diseño previo puede llevar a problemas en las

XP	<p>Permite ahorrar recursos, especialmente los humanos, ya que los programadores se enfocan más en el desarrollo del producto y no tanto en papeleo y reuniones.</p> <p>Al separar las tareas en módulos se reduce el riesgo de fallas.</p>	<p>aplicaciones, especialmente en las más grandes.</p> <p>La documentación escasa puede llevar a problemas a la hora de realizar las pruebas en el sistema.</p> <p>La aplicación de programación en pares, aunque puede brindar buenos resultados, también lleva a código repetido y la necesidad de refactorización constante.</p> <p>La falta de documentación ocasiona problemas cuando uno o varios miembros de equipo de desarrollo dejan el proyecto y llegan nuevos programadores.</p>
Crystal	<p>Apropiada para entornos ligeros y proyecto pequeños.</p> <p>Adaptable a cambios.</p>	<p>No se cuenta con una forma definida de trabajo, lo cual puede llevar al fracaso total de un proyecto.</p>

Fuente: Elaboración propia

Luego de ver las ventajas y desventajas de distintas metodologías se vio conveniente desarrollar este proyecto bajo el uso del marco de trabajo *Scrum*, el cual permite abordar proyectos complejos de manera ágil, de forma que el producto final a ser entregado pueda ir adaptándose y moldeándose según los cambios y requerimientos nuevos que vayan surgiendo a lo largo del ciclo de desarrollo.

Debido a su naturaleza incremental, *Scrum* además resulta muy útil a la hora de encarar proyectos que no cuentan con todos los requerimientos claros desde el inicio del desarrollo.

Scrum además es un marco de trabajo probado y que ha brindado grandes resultados, no solo en el mundo de la informática, también en muchas otras áreas y disciplinas en las que se trabaja en proyectos de toda índole.

En el desarrollo de este proyecto se usará *Scrum*, pero modificando ciertos aspectos para adaptar el marco de trabajo a una sola persona, por lo cual se prescindirá de las reuniones que *Scrum* requiere, esto debido al hecho de que el proyecto es desarrollado por una sola persona y el objetivo principal de las reuniones dentro de *Scrum* es el de comunicar al resto del equipo de desarrollo distintos aspectos dentro del mismo, lo cual no es necesario en el caso de este proyecto. A pesar de prescindir de las reuniones, se tomarán en cuenta los objetivos específicos de cada reunión dentro de *Scrum* para no perder la esencia misma del marco de trabajo.

1.3.2. Scrum

Scrum es un marco de trabajo con el cual se puede abordar problemas complejos en pos de entregar un producto del máximo valor y calidad (Schwaber, y otros, 2013).

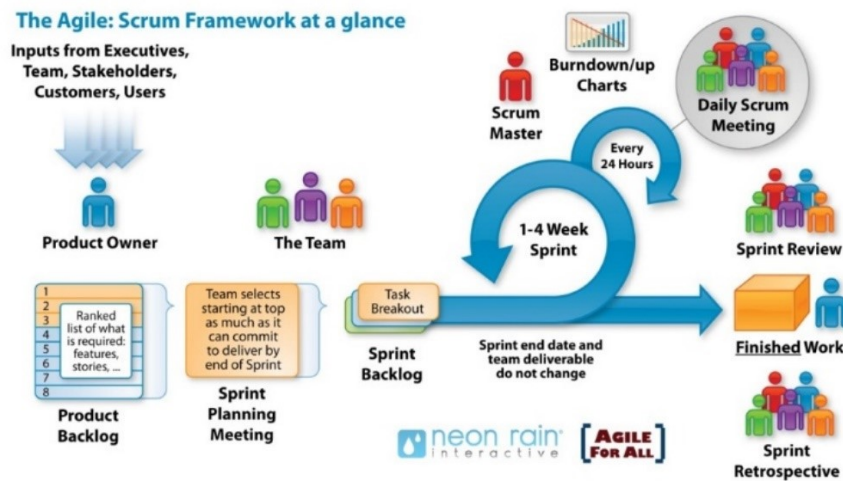
Scrum es:

- Ligero
- Fácil de entender

- Dificil de llegar a dominar

Desde los principios de los años 90 se ha usado este marco de trabajo para gestionar proyectos complejos. Vale la pena resaltar que *scrum* no es un proceso o una técnica para construir productos, en realidad es un marco de trabajo en el cual se pueden emplear varias técnicas y procesos.

Figura 6: Flujo de trabajo dentro de la metodología SCRUM



Fuente: (Agile For All, 2016)

Scrum está basado en la teoría de control de procesos empírica o empirismo. Esto significa que *scrum* asume que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce, además que se emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control de riesgos (Schwaber, y otros, 2013).

En la implementación del control de procesos empíricos existen tres pilares fundamentales, los cuales son transparencia, inspección y adaptación (Schwaber, y otros, 2013).

La transparencia hace referencia a que todos los aspectos significativos del proceso deben ser visibles para los responsables del resultado. Estos aspectos deben ser definidos por un estándar común, de manera que los observadores compartan un entendimiento

común de lo que se está viendo, en otras palabras, todos los participantes deben compartir un lenguaje común para referirse al proceso y tanto quienes desarrollan el producto como quienes lo aceptan deben compartir una definición común de “terminado”.

La inspección se refiere a que los usuarios de *scrum* deben inspeccionar los artefactos del marco de trabajo y el progreso hacia un objetivo, de manera que puedan identificarse falencias y estas puedan ser corregidas, esto se conoce como adaptación.

La adaptación determina que, si uno o más aspectos del proceso se desvían de límites aceptables y que el producto resultante no será aceptable, el proceso debe ser ajustado. Dicho ajuste debe ser realizado cuanto antes de manera que puedan minimizarse desviaciones mayores.

El equipo de *scrum* está formado por el dueño del producto, el equipo de desarrollo y un *scrum master*. Los equipos en *scrum* son autoorganizados y multifuncionales, esto ya que los equipos pueden elegir mejor la forma de trabajar y además tienen todas las competencias necesarias para llevar a cabo del trabajo sin depender de otras personas que no sean del equipo.

Los equipos en *scrum* entregan productos de forma iterativa e incremental en pos de obtener la mayor cantidad de retroalimentación posible, además que de esta forma se asegura contar siempre con versiones del producto potencialmente útiles.

1.3.2.1. El dueño de producto (Product Owner)

El dueño de producto es el responsable de maximizar el valor del producto y del trabajo del equipo de desarrollo. El cómo se lleva a cabo esto podría variar ampliamente entre distintas organizaciones, equipos *scrum* e individuos (Schwaber, y otros, 2013).

El dueño de producto es la única persona responsable de gestionar la lista del producto (*Product backlog*). La gestión de la lista del producto incluye:

- Expresar claramente los elementos de la lista del producto.

- Ordenar los elementos en la lista del producto para alcanzar los objetivos y misiones de la mejor manera posible.
- Optimizar el valor del trabajo desempeñado por el equipo de desarrollo.
- Asegurar que la lista del producto es visible, transparente y clara para todos, y que muestra aquello en lo que el equipo trabajará a continuación.
- Asegurar que el equipo de desarrollo entiende los elementos de la lista del producto al nivel necesario.

1.3.2.2. El equipo de desarrollo (Development Team)

El equipo de desarrollo consiste en los profesionales que desempeñan el trabajo de entregar un incremento de producto “terminado”, que potencialmente se pueda poner en producción, al final de cada *sprint*. Solo los miembros del equipo de desarrollo participan en la creación del incremento.

Los equipos de desarrollo son estructurados y empoderados por la organización para organizar y gestionar su propio trabajo. La sinergia resultante optimiza la eficiencia y efectividad del equipo de desarrollo (Schwaber, y otros, 2013).

1.3.2.3. El scrum master

El *scrum master* es el responsable de asegurar que *scrum* es entendido y adoptado. Los *scrum masters* hacen esto asegurándose de que el equipo *scrum* trabaja ajustándose a la teoría, prácticas y reglas del marco de trabajo.

El *scrum master* es un líder que está al servicio del equipo. Este ayuda a las personas externas al equipo a entender qué interacciones con el equipo *scrum* pueden ser de ayuda y cuáles no. Además, ayuda a todos a modificar estas interacciones para maximizar el valor del producto creado por el equipo (Schwaber, y otros, 2013).

1.3.2.4. El sprint

El corazón de *scrum* es el *sprint*, es un bloque de tiempo (time-box) de un mes o menos durante el cual se crea un incremento de producto “terminado”, utilizable y potencialmente desplegable. Es más conveniente si la duración del *sprint* es consistente a lo largo del esfuerzo de desarrollo. Cada nuevo *sprint* comienza inmediatamente después de la finalización del *sprint* previo.

Durante el *sprint*:

- No se realizan cambios que puedan afectar al objetivo del *sprint*.
- Los objetivos de calidad no disminuyen.
- El alcance puede ser clarificado y renegociado entre el dueño de producto y el equipo de desarrollo a medida que se va aprendiendo más.

Cada *sprint* puede considerarse un proyecto con un horizonte no mayor de un mes. Al igual que los proyectos, los *sprint* se usan para lograr algo. Cada *sprint* tiene una definición de qué se va a construir, un diseño y un plan flexible que guiará la construcción y el trabajo y el producto resultante (Schwaber, y otros, 2013).

1.3.2.5. Reunión de planificación de sprint (Sprint Planning Meeting)

El trabajo a realizar durante el *sprint* se planifica en la reunión de planificación de *sprint*. Este plan se crea mediante el trabajo colaborativo del equipo *scrum* completo.

La reunión de planificación de *sprint* tiene un máximo de duración de ocho horas para un *sprint* de un mes. Para *sprints* más cortos, el evento es usualmente más corto.

El *scrum master* se asegura de que el evento se lleve a cabo y que los asistentes entiendan su propósito. Además, enseña al equipo a mantenerse dentro del bloque de tiempo.

La reunión de planificación de *sprint* responde a las siguientes preguntas:

- ¿Qué puede entregarse en el incremento resultante del *sprint* que comienza?
- ¿Cómo se conseguirá hacer el trabajo necesario para entregar el incremento?

El equipo de desarrollo trabaja para proyectar la funcionalidad que se desarrollará durante el *sprint*. El dueño de producto discute el objetivo que el *sprint* debería lograr y los elementos de la lista de producto que, si se completan en el *sprint*, lograrían el objetivo del mismo.

A medida que el equipo de desarrollo trabaja, se mantiene el objetivo del *sprint* en mente. Con el fin de satisfacer el objetivo del *sprint* se implementa la funcionalidad y la tecnología. Si el trabajo resulta ser diferente de lo que el equipo de desarrollo espera, ellos colaboran con el dueño del producto para negociar el alcance de la lista de pendientes del *sprint* (*Sprint backlog*) (Schwaber, y otros, 2013).

1.3.2.6. Revisión de *sprint* (*Sprint Review*)

Al final del *sprint* se lleva a cabo una revisión de *sprint* para inspeccionar el incremento y adaptar la lista de producto si fuese necesario. Durante la revisión de *sprint*, el equipo *scrum* y los interesados colaboran acerca de lo que se hizo durante el *sprint*. Basándose en esto, y en cualquier cambio a la lista de producto durante el *sprint*, los asistentes colaboran para determinar las siguientes cosas que podrían hacerse para optimizar el valor. Se trata de una reunión informal, no una reunión de seguimiento, y la presentación del incremento tiene como objetivo facilitar la retroalimentación de información y fomentar la colaboración.

Se trata de una reunión restringida a un bloque de tiempo de cuatro horas para *sprint* de un mes. Para *sprint* más cortos, se reserva un tiempo proporcionalmente menor. El *scrum master* se asegura de que el evento se lleve a cabo y que los asistentes entiendan

su propósito. El *scrum* Master enseña a todos a mantener el evento dentro del bloque de tiempo fijado.

El resultado de la revisión de *sprint* es una lista de producto revisada, que define los elementos de la lista de producto posibles para el siguiente *sprint*. Es posible además que la lista de producto reciba un ajuste general para enfocarse en nuevas oportunidades (Schwaber, y otros, 2013).

1.3.2.7. Lista de producto (Product Backlog)

La lista de producto es una lista ordenada de todo lo que podría ser necesario en el producto, y es la única fuente de requisitos para cualquier cambio a realizarse en el producto. El dueño de producto (*Product owner*) es el responsable de la lista de producto, incluyendo su contenido, disponibilidad y ordenación (Schwaber, y otros, 2013).

Una lista de producto nunca está completa. El desarrollo más temprano de la misma solo refleja los requisitos conocidos y mejor entendidos al principio. La lista de producto evoluciona a medida que el producto y el entorno en el que se usará también lo hacen. La lista de producto es dinámica; cambia constantemente para identificar lo que el producto necesita para ser adecuado, competitivo y útil. Mientras el producto exista, su lista de producto también existe.

La lista de producto enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a ser hechos sobre el producto para entregas futuras. Los elementos de la lista de producto tienen como atributos la descripción, la ordenación, la estimación y el valor.

1.3.2.8. Lista de pendientes del sprint (Sprint Backlog)

La lista de pendientes del *sprint* es el conjunto de elementos de la lista de producto seleccionados para el *sprint*, más un plan para entregar el incremento de producto y conseguir el objetivo del *sprint*. La lista de pendientes del *sprint* es una predicción hecha por el equipo de desarrollo acerca de qué funcionalidad formará parte del próximo incremento y del trabajo necesario para entregar esa funcionalidad en un incremento “terminado” (Schwaber, y otros, 2013).

1.2.3.9. Definición de “terminado” (Definition of “Done”)

Cuando un elemento de la lista de producto o un incremento se describe como “terminado”, todo el mundo debe entender lo que significa “terminado”. Aunque esto varía significativamente para cada equipo *scrum*, los miembros del equipo deben tener un entendimiento compartido de lo que significa que el trabajo esté completado, para asegurar la transparencia. Esta es la definición de “terminado” para el equipo *scrum* y se utiliza para evaluar cuándo se ha completado el trabajo sobre el incremento de producto (Schwaber, y otros, 2013).

1.4. Tecnologías

1.4.1. Frameworks web

Actualmente existen muchos *frameworks* para el desarrollo de aplicaciones web, cada cual con características que brindan distintas ventajas y desventajas al momento de abordar un proyecto, por lo cual resulta importante seleccionar uno que permita abordar de manera eficaz el proyecto a desarrollar. A continuación, se listaran las principales características de distintos *frameworks* y sus mayores ventajas y desventajas:

- ASP.NET: desarrollado y distribuido por Microsoft, es un *framework* para desarrollar aplicaciones web. Es uno de los más usados en el medio, por lo cual se cuenta con documentación extensiva y una amplia comunidad que trabaja con el mismo. Además, es posible desarrollar servicios web a través de *Windows Communication Foundation*. Entre las principales ventajas se puede mencionar que existen muchos componentes ya desarrollados y distribuidos como librerías, además que el IDE *Visual Studio* es muy poderoso y permite agilizar el desarrollo de las aplicaciones. La principal desventaja es el consumo de recursos, este *framework* consume más recursos que otros, lo que traduce en la necesidad de adquirir servidores que resultan más caros que los que se requerirían con otras tecnologías (Wikipedia, 2015).
- AngularJS: es un *framework javascript* de código abierto, mantenido por Google, el cual se utiliza para desarrollar aplicaciones web, pero en su mayoría *frontend*, es decir la parte de la aplicación con la que interactuará el usuario. AngularJS está construido en base a la creencia de que la programación declarativa debe utilizarse para generar interfaces de usuario, en cambio la programación imperativa debe ser usada para la lógica de negocio. Entre sus principales ventajas están el gran soporte que cuenta de Google y la comunidad, muchas maneras de realizar las mismas tareas, menos líneas de código al momento de desarrollar las aplicaciones y un amplio soporte a servicios web (Novosiolov, 2015). La principal desventaja y motivo para no utilizar este *framework* en el desarrollo de este proyecto es que la versión actual de AngularJS dejará de tener soporte cuando surja AngularJS 2, pero esta nueva versión del *framework* fue desarrollada totalmente desde el inicio, lo que significa que muchas de las aplicaciones desarrolladas con versiones antiguas tendrán que ser reescritas y la nueva versión se encuentra aún en su etapa *beta*, por lo cual no es aconsejable utilizarla en ambientes de trabajo,
- Ruby On Rails: *framework* para desarrollar aplicaciones web mediante el lenguaje Ruby. Este *framework* es el seleccionado para desarrollar el *backend* de este proyecto, es decir los servicios web. Sus principales ventajas son la facilidad

para entender el código, gran cantidad de librerías (gemas) y una amplia documentación, además que la implementación de servicios web es simple y manejada casi en su totalidad por el *framework*. Entre las desventajas se debe mencionar que las aplicaciones no son tan rápidas como las desarrolladas en C o Java.

- EmberJS: *framework* seleccionado para implementar el *frontend* del proyecto. La principal razón para elegir EmberJS es la facilidad con la que este *framework* se conecta con APIs desarrolladas en Ruby on Rails además que el código fuente resulta fácil de entender, lo cual permite a cualquier programador abordar un proyecto EmberJS ya desarrollado de manera más rápida.

Además, se puede observar la popularidad de los distintos *frameworks* (Figura 7) de manera que se tenga una visión de la aceptación de los mismos.

Figura 7: Popularidad de distintos frameworks

Rankings

Framework	Github Score	Stack Overflow Score	Overall Score
ASP.NET		100	100
AngularJS	100	95	97
Ruby on Rails	95	98	96
ASP.NET MVC		93	93
Django	90	92	91
Meteor	96	79	87
Laravel	92	83	87
Spring	82	90	86
Express	93	79	86
CodeIgniter	85	85	85
Symfony	85	84	84
Ember.js	88	78	83
Flask	90	74	82

Fuente: (Hot Frameworks, 2016)

1.4.2. Ruby

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU (Wikipedia, 2015).

El creador del lenguaje, Yukihiro "Matz" Matsumoto, ha dicho que Ruby está diseñado para la productividad y la diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario.

Ruby se caracteriza por ser (Wikipedia, 2015):

- Orientado a objetos.
- Cuatro niveles de ámbito de variable: global, clase, instancia y local.

- Manejo de excepciones.
- Iteradores y clausuras.
- Expresiones regulares nativas similares a las de Perl a nivel del lenguaje.
- Posibilidad de redefinir los operadores (sobrecarga de operadores).
- Recolección de basura automática.
- Altamente portable.
- Hilos de ejecución simultáneos en todas las plataformas usando *green threads*.
- Carga dinámica de DLL/bibliotecas compartidas en la mayoría de las plataformas.
- Introspección, reflexión y metaprogramación.
- Amplia librería estándar.
- Soporta inyección de dependencias.
- Soporta alteración de objetos en tiempo de ejecución.

1.4.3. Ruby On Rails

Es un *framework* de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, que sigue la arquitectura Modelo Vista Controlador (MVC).

Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros *frameworks* y con un mínimo de configuración, este punto también conocido como *Convention Over Configuration*, el cual es un paradigma de diseño que busca disminuir el número de decisiones que debe tomar un programador en pos de simplificar su trabajo, pero sin perder flexibilidad (Wikipedia, 2016). En el caso de *Rails* un ejemplo es la asignación automática de nombres de tablas al momento de crear un modelo; suponiendo que tenemos un modelo *User*, la tabla correspondiente a este modelo se llamará *Users*, algo que en ningún momento el programador debe especificar en el código.

Otra característica importante de *Rails* es DRY (*Don't Repeat Yourself*) (Wikipedia, 2016), el cual es un principio de desarrollo de software que busca reducir la cantidad de

código duplicado que escribe un programador. En el caso de *Rails* la aplicación de esto puede verse al momento de crear un formulario; suponiendo que se desea crear y editar un modelo en lugar de desarrollar un formulario para la creación y otro formulario para editar el modelo, se puede usar el mismo formulario para ambas operaciones.

Otro punto muy importante dentro del *framework* es el mapeo automático de objetos al modelo relacional, es decir que solamente hace falta definir un modelo y *Rails* se encarga de llevar este modelo a una base de datos relacional.

También vale la pena resaltar se cuenta con herramientas para realizar distintos tipos de pruebas, desde las pruebas unitarias hasta pruebas de integración, todo esto a través del componente *Mini Test* que se encuentra acoplado a todo proyecto *Rails*.

En cuanto a opciones de validación, *Rails* ofrece muchas facilidades, ya que la sintaxis es bastante simple, por ejemplo, para validar la presencia de un atributo al momento de su creación solamente se requiere el siguiente código:

```
class Article < ActiveRecord::Base
  validates :title, presence: true
end
```

El lenguaje de programación Ruby permite la metaprogramación, de la cual *Rails* hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. *Rails* se distribuye a través de *RubyGems*, que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones Ruby.

Dentro del *framework* también se cuenta con generadores que facilitan y aceleran el proceso de desarrollo. El más utilizado dentro de este proyecto es el generador *Scaffold*, el cual permite generar toda la lógica para un CRUD (*Create, Read, Update, Delete*), creando el modelo, el controlador y las vistas correspondientes. Un ejemplo del uso de *Scaffold* es el siguiente:

```
rails generate scaffold User name:string last_name:string username:string
```

Este comando genera un modelo llamado *User*, un controlador denominado *UserController* en el cual se encuentra toda la lógica necesaria para realizar un CRUD y finalmente genera las vistas necesarias, es decir un índice de usuarios y los formularios para crear y editar. También genera una tabla denominada *Users*.

Finalmente, para el desarrollo de este proyecto se usó la gema *rails-api*, la cual facilita la implementación de una API. Esta gema modifica los generadores de *Rails*, de forma que ignora y no crea componentes innecesarios para una API, como por ejemplo la carpeta *Views* en la cual se encuentran las vistas.

1.4.4. MySQL

Es un sistema de gestión de bases de datos relacional, ampliamente usado en entornos de desarrollo web y considerada la base de datos de código abierto más popular del mundo. (Wikipedia, 2016).

Las características más importantes de *MySQL* son: (Oracle MySQL, 2016) (Search IT Channel, 2007)

- Desarrollado en C y C++.
- Funciona en distintas plataformas (*Windows, Linux y otros*).
- Sus principales tipos de datos son: *INTEGERS* (1,2,3,4 y 8 bytes de longitud), *FLOAT*, *DOUBLE*, *CHAR*, *VARCHAR*, *BINARY*, *VARBINARY*, *TEXT*, *BLOB*, *DATE*, *TIME*, *DATETIME*, *TIMESTAMP*, *YEAR*, *SET* y *ENUM*.
- Soporta bases de datos grandes. Sin problemas maneja 50 millones de datos, 200,000 tablas y alrededor de 5,000,000,000 de filas. Además, soporta hasta 64 índices por tabla.
- Control en las transacciones, es decir que *MySQL* se encarga de que todas las operaciones solicitadas sean realizadas correctamente o ninguna sea realizada, de esta manera se evitan problemas de concurrencia.

- Debido a un sistema de asignación basado en hilos, el rendimiento de *MySQL* es muy eficiente, inclusive en el caso de bases de datos muy amplias.

1.4.5. *EmberJS*

EmberJS es un *framework* javascript de código abierto, basado en la arquitectura MVC (*Model View Controller*) que permite crear aplicaciones web (Wikipedia, 2016).

Ember fue creado el año 2007 con el nombre de *SproutCore MVC* y fue desarrollado por la empresa *Apple*, pero el año 2011 Yehuda Katz, contribuyente en el proyecto de *Ruby On Rails*, se encargó de continuar con el desarrollo del *framework* (Wikipedia, 2016)k.

Igual que *Rails*, *Ember* se basa en COC (*Convention over Configuration*), lo que significa que un programador no debe pasar mucho tiempo configurando el sistema para que este funcione, sino que se enfoca en realizar los componentes únicos de cada sistema y reutilizar los que brinda el *framework* (Wikipedia, 2016).

Ember cuenta con 5 conceptos básicos (EmberJS, 2014):

- Rutas: el estado de una aplicación es representado por una URL, cada una de estas corresponde a un objeto *route* que controla lo que se mostrara al usuario.
- Modelos: cada ruta tiene un modelo asociado que contiene los datos asociados al estado del sistema.
- Plantillas: estas son usadas para construir el código HTML de la aplicación, es decir todo lo relacionado a la interfaz de usuario.
- Componentes: son etiquetas HTML creadas por el programador. Estas etiquetas permiten definir comportamientos y acciones que se activan desde la interfaz de usuario.
- Servicios: en *Ember* los servicios son objetos *singleton* que sirven para almacenar datos, como por ejemplo sesiones de usuario.

Una característica muy importante de *Ember* y la principal razón para haber elegido este *framework* es *ActiveModelAdapter*. Este es un adaptador que permite obtener datos desde una aplicación desarrollada en *Ruby On Rails* sin necesidad de hacer llamadas JSON de manera manual (EmberJS, 2014). Para aclarar este punto se tiene el siguiente código:

```
App.ApplicationRoute = Ember.Route.extend({
  model: function() {
    return Ember.$.getJSON('https://example.com/users').then(function(data) {
      return data.splice(0, 3);
    });
  }
});
```

En esta porción de código puede verse que se realiza una consulta para obtener un archivo JSON de manera manual, es decir que mediante la función *getJSON* mandamos una URL y obtenemos un archivo JSON. Esta es la manera manual de obtener datos desde un servidor, pero *ActiveModelAdapter* permite hacer lo siguiente:

```
export default DS.ActiveModelAdapter.extend({
  host: 'http://example.com'
});
```

En esta porción de código se define el *host* de la aplicación, es decir el servidor al que se realizaran las consultas.

```
export default Ember.Route.extend({
  model() {
    return this.get('store').findAll('user');
  }
});
```

Finalmente, para obtener los datos de todos los usuarios, en lugar de realizar una consulta esperando un archivo JSON, se accede al *store* de la aplicación, es decir al componente que accede al almacén de datos del sistema, el cual fue definido como

'http://example.com. Una vez que se accede al almacén, el programador tiene acceso a los modelos definidos en la aplicación *Ruby on Rails*.

1.4.6. Twilio

Twilio es un servicio alojado en la nube, el cual mediante sus distintas APIs permite el uso de mensajería instantánea, llamadas de voz, envío de mensajes SMS y autenticación de usuarios (Twilio, 2016).

En el desarrollo de este proyecto se ha utilizado este servicio para el envío de mensajes SMS a teléfonos móviles. Para integrar este servicio con el sistema se utilizó la gema *twilio-ruby* de *Ruby On Rails*, la cual permite integrar el servicio con una aplicación desarrollada con el *framework* mencionado (Twilio, 2016).

1.4.7. Apipie

Para desarrollar la documentación del sistema, es decir la documentación de la API, de forma que a futuro pueda ser utilizada y consumida por distintas aplicaciones, se utilizó la gema de *Ruby* denominada *apipie-rails*. Esta gema es un DSL y motor de *Rails* que permite documentar APIs RESTful (Apipie, 2016).

Para realizar la documentación de la API no se requiere aprender ningún lenguaje, ya que se usa *Ruby*. La manera de describir las distintas funciones es:

```
api :GET, '/users/:id'
  param :id, :number
  def show
    # .....
  end
```

2. INGENIERÍA DEL PROYECTO

2.1. Propuesta de un modelo de procedimientos de oferta y venta para el mercado boliviano a través de Internet

Para realizar una propuesta adecuada al mercado boliviano primeramente se debe entender el contexto en el cual se maneja el comercio electrónico en el país.

Como se mencionó anteriormente, en Bolivia solo el 2% de las transacciones son realizadas usando tarjeta de crédito (Anexo 1), por lo cual no resulta factible implementar pagos mediante este método.

Para poder plantear un modelo adecuado se realizaron entrevistas al personal de la importadora con el objetivo de encontrar fallencias en sus procesos y además entender cómo se realiza la oferta y venta de productos, de manera que se pueda proponer una forma de que los clientes puedan adquirir producto por Internet de manera que resulte cómodo y factible para ellos.

2.1.1. Problemas encontrados

Al realizar las entrevistas a los trabajadores de la importadora se encontraron los siguientes problemas:

- Existen pedidos de los cuales no se tiene seguimiento y muchas veces estos no son recogidos por los clientes.
- Existen clientes que realizan pedidos por correo electrónico, pero la importadora no tiene una política para revisar estos correos, es decir que muchas veces los correos recibidos no son revisados.
- Al momento de realizar la oferta de productos, el vendedor debe llevar muestras de los productos, ya que los clientes en la mayoría de los casos solicitan ver

estos, pero el manejo y transporte continuo de las muestras hace que estas se dañen.

- Dentro de Cochabamba el vendedor tarda aproximadamente 7 días en realizar las rutas para llegar a todos los posibles clientes.
- Cuando un pedido es realizado no siempre se sabe la disponibilidad de este en stock, por lo cual se va demorando la entrega del mismo hasta tenerlo.
- No existe una documentación formal de los productos que salen de los almacenes, por lo cual no se tiene un control correcto.
- No se cuenta con una manera eficaz de buscar los productos que compro un cliente.

Como se puede ver los problemas de la empresa conllevan a retrasos en entregas de pedidos y a un control inadecuado de las ventas, por lo cual la importadora va perdiendo credibilidad de parte de su clientela.

2.1.2. Propuesta del modelo

Observando los problemas se plantea realizar un sistema web donde los clientes puedan acceder para buscar y ver los distintos productos ofrecidos por la importadora.

En este sistema el cliente contará con un carrito de compras donde podrá ir almacenando los productos que desea adquirir. Una vez que se termine de agregar productos en el carrito, el cliente procederá a seleccionar un método para confirmar su pedido. Debido a que no se usará tarjetas de crédito en el proceso de compra, el pago de los productos será en efectivo, pero el pedido podrá ser realizado a través de Internet y será confirmado de dos maneras, las cuales son:

- Confirmación de un pedido mediante un mensaje en el celular. Este método de pago propuesto permitirá al cliente confirmar su pedido recibiendo un mensaje de texto en su teléfono con un código, este código deberá ser ingresado en el

sistema web para confirmar el pedido y que la importadora empiece a preparar el mismo.

La ventaja de esta manera de confirmar un pedido es que al enviar un mensaje al celular se confirma la autenticidad del mismo, además que es una manera inmediata de adquirir productos.

Figura 8: Flujo para confirmar un pedido mediante un mensaje de texto



Fuente: Elaboración propia

- Confirmación de un pedido mediante envío de foto del comprobante de depósito. Esta manera de confirmar un pedido no es inmediata como la anterior, pero es especialmente útil para pedidos muy grandes y que sean realizados desde otros departamentos, ya que se confirmará la cancelación del monto necesario y la importadora podrá preparar el pedido.

Para utilizar esta forma de confirmar el cliente deberá subir una foto del extracto bancario al sistema web. Este extracto será el comprobante de que se realizó un depósito a la cuenta de la importadora, asegurando así la autenticidad del pedido.

Figura 9: Flujo para confirmar un pedido mediante foto de extracto bancario



Fuente: Elaboración propia

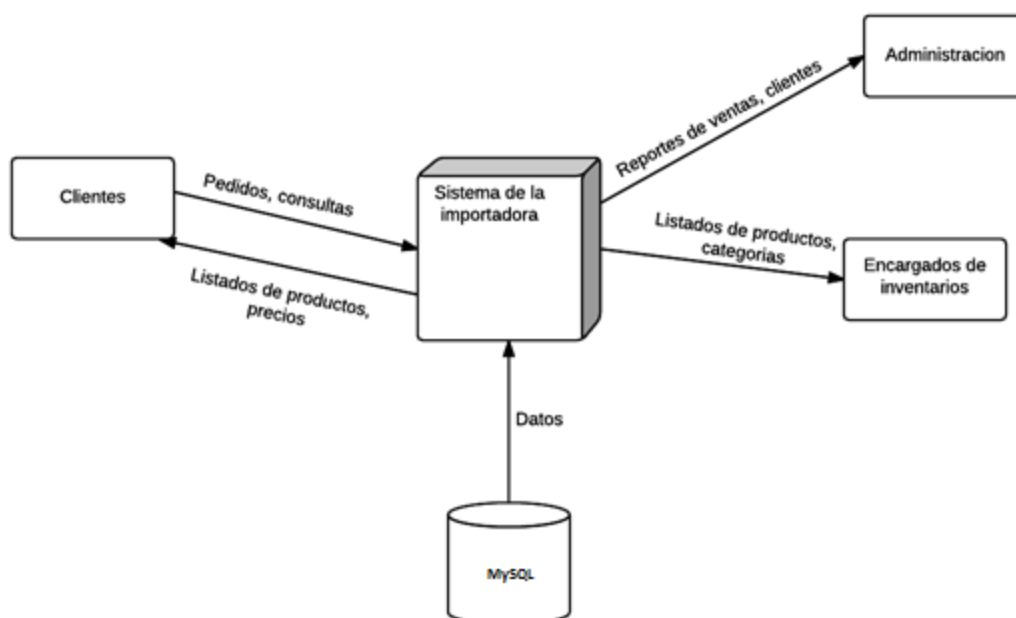
Como se puede ver, el mayor problema encontrado a la hora de plantear las formas de realizar el pago fue la autenticidad de los pedidos realizados por el sistema web, es por eso que se enfatizó en las propuestas un mecanismo de confirmar la veracidad del pedido. En el primer método esto se logra mediante el conocimiento del número de celular del cliente, lo que significa que en caso de algún problema este podrá ser contactado. En el segundo método se logra esto al tener ya pagado el pedido, lo que libera de sospechas de un pedido ficticio.

Además de afectar a la forma de interactuar con los clientes, la propuesta modificara ciertas operaciones y procesos de la importadora, ya que se tendrá un mejor control de los pedidos, se controlará las salidas y entradas de productos al stock, y se mejorara el alcance a nuevos clientes a través del sistema web.

2.2. Desarrollo del sistema para la venta en línea de productos

El sistema para la oferta y venta de productos se dividió en 5 iteraciones. Para una mejor abstracción del sistema se cuenta con la siguiente figura:

Figura 10: Diagrama de contexto del sistema



Fuente: Elaboración propia

Como se puede ver en la figura 10, el sistema se comunicará con clientes, administración y encargados de almacén. En el caso de los clientes, el sistema brindará listados de productos con la información respectiva para los clientes, como ser el precio, color y otros datos importantes. Para administración se brindará reportes sobre las ventas en general. Los encargados de inventarios usaran el sistema para obtener listados de productos, además de registrar los mismos.

2.2.1. Product backlog

Tabla 3: Product Backlog

Prioridad	Como	Necesito	Para	Esfuerzo
1	Administrador	Crear, modificar y borrar información de categorías de productos	Saber que subcategorías y productos asignar a estas categorías	2
2	Administrador	Crear, modificar y borrar información de subcategorías de productos	Saber que productos asignar a estas subcategorías	2
3	Administrador	Crear, modificar y borrar información de proveedores	Saber quiénes proveen los productos	2
4	Administrador	Crear, modificar y borrar información de fabricantes	Saber que productos pertenecen a cada fabricante	2
5	Administrador	Crear, modificar y borrar información de productos	Saber con qué productos se cuenta	2
6	Administrador	Crear, modificar y dar de baja a un cliente	Saber la información de los clientes	2
7	Administrador/Encargado de ventas	Buscar categorías de productos	Saber que productos pertenecen a cada categoría	1
8	Administrador/Encargado de ventas	Buscar subcategorías de productos	Saber que productos pertenecen a cada subcategoría y bajo que categoría están,	1
9	Administrador/Encargado de ventas	Buscar proveedores de productos	Saber quiénes proveen los productos	1
10	Administrador/Encargado de ventas	Buscar fabricantes de productos	Saber que empresas fabrican los productos	1
11	Administrador/Encargado de ventas	Buscar productos	Saber que productos existen y sus datos	1
12	Administrador/Encargado de ventas	Ver reportes de clientes	Acceder a información que pueda resultar útil para las ventas	2
13	Administrador/Encargado de ventas	Ver reportes de clientes	Acceder a información que pueda resultar útil para las ventas y el servicio a los clientes	2
14	Administrador/Encargado de ventas	Ver reportes de productos	Acceder a información que pueda resultar útil para las ventas y la adquisición de productos	2
15	Clientes	Seleccionar productos a un carrito de compras	Poder almacenar los productos que desean adquirir en un lugar de la pagina	2
16	Clientes	Confirmar reserva del carrito de compras	Confirmar los productos que desea adquirir	2

17	Cientes	Seleccionar método de confirmación	Seleccionar que método de confirmación desea utilizar	1
18	Cientes	Confirmar el pedido mediante imagen del depósito bancario	Subir una imagen del depósito bancario recibido al momento de realizar el pago	2
19	Cientes	Confirmar el pedido mediante un SMS	Recibir un SMS al celular con un código de confirmación de pedido	3
20	Administrador/Encargado de ventas	Confirmar reservas realizadas mediante imagen de depósito bancario	Poder confirmar las reservas realizadas con una imagen del depósito bancario	1
21	Administrador/Encargado de ventas	Ver reservas	Saber que productos se debe preparar para cada cliente	2
22	Administrador	Realizar entradas y salidas de productos	Realizar el control de los productos y almacenes	2
23	Administrador	Ver movimientos de entradas y salidas de productos	Ver el detalle de los movimientos realizados	2
24	Administrador	Buscar clientes y productos	Encontrar rápidamente tanto usuarios como productos	1

Fuente: Elaboración propia

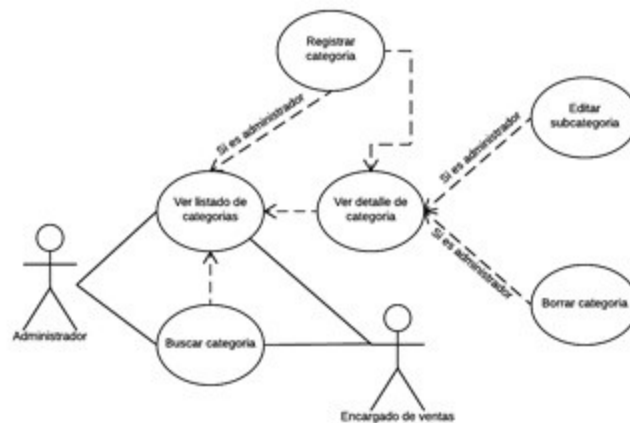
El *product backlog* contará con la historia de usuario y su respectiva dificultad, la cual será estimada entre 1 y 3, siendo 1 la dificultad más baja y 3 la más alta. Además, las historias de usuario están priorizadas, siendo las primeras aquellas que tienen prioridad más alta.

2.2.2. Sprint 1. Desarrollo de categorías y subcategorías

En este *sprint* se llevó a cabo el desarrollo de las funcionalidades de categorías y subcategorías. Estas funcionalidades cumplían las historias de usuario para registrar una categoría, una subcategoría y las demás operaciones referentes al CRUD.

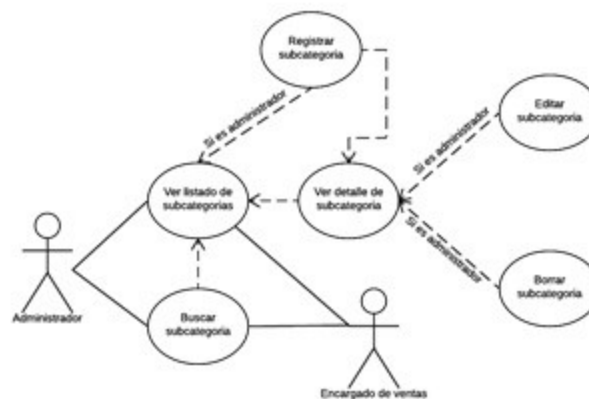
2.2.2.1. Casos de uso para categorías y subcategorías

Figura 11: Caso de uso para categorías



Fuente: Elaboración propia

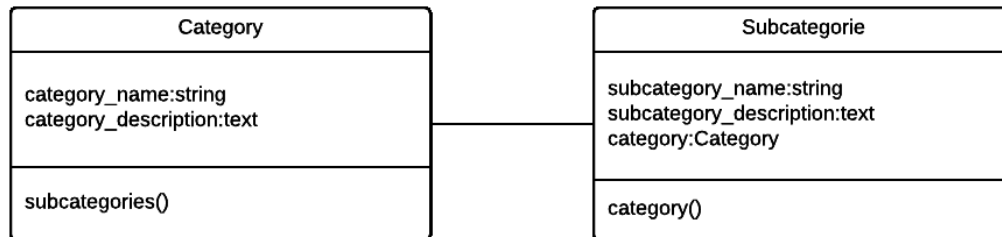
Figura 12: Casos de uso para subcategorías



Fuente: Elaboración propia

2.2.2.2. Diagrama de clases para categorías y subcategorías

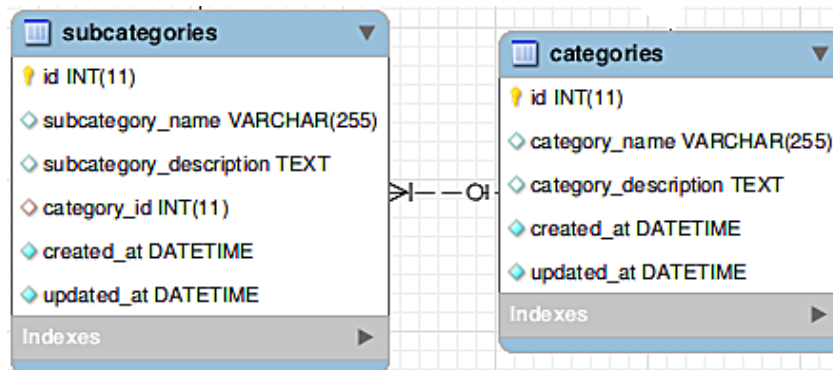
Figura 13: Diagrama de clases para categorías y subcategorías



Fuente: Elaboración propia

2.2.2.3. Diagrama de bases de datos para categorías y subcategorías

Figura 14: Diagrama de bases de datos de categorías y subcategorías



Fuente: Elaboración propia

2.2.2.4. Arquitectura de categorías y subcategorías

Como se puede observar en el diagrama de clases (Figura 13), la arquitectura utilizada para desarrollar este módulo es la de Modelo- Controlador.

La razón para usar esta arquitectura es que el desarrollo del módulo se da con el marco de trabajo *Ruby On Rails*, el cual por defecto utiliza MVC, pero al usar la gema *rails-api* se obvian las vistas, ya que una API no requiere implementar interfaz gráfica.

El *frontend* utiliza la arquitectura Modelo Vista Controlador, ya que esta es la que se maneja dentro del *framework EmberJS*.

2.2.2.5. Pruebas de categorías y subcategorías

El desarrollo de las pruebas necesarias para verificar este *sprint* se hizo mediante el componente integrado en *Ruby on Rails*, denominado como *Minitest*. Las pruebas que se realizaron fueron a los modelos *Category* y *Subcategory*, además a los controladores *CategoriesController* y *SubcategoriesController*.

La verificación de estos componentes se realizó mediante pruebas de unidad, donde se probaron las funciones para la creación de estos modelos y su persistencia en la base de datos. Un ejemplo de una prueba realizada para la creación de una categoría es:

```
test "should create category" do
  assert_difference('Category.count') do
    post: create, category: {category_description:
      @category.category_description, category_name:
      @category.category_name }
  end
  assert_response 201
end
```

Como puede verse en el anterior código, se realizan dos verificaciones, que el número de categorías haya cambiado y que además se obtenga del servidor una respuesta 201, la cual significa que el objeto ha sido guardado exitosamente.

Para este *sprint* se realizaron 8 pruebas, de las cuales todas pasaron de manera exitosa.

2.2.2.6. Retrospectiva

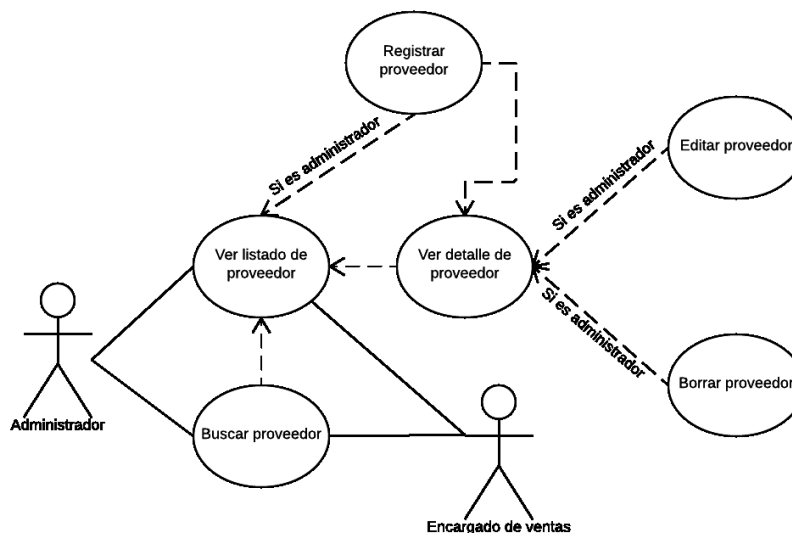
Este sprint fue llevado a cabo sin mayores dificultades y se pudo cumplir con los tiempos establecidos.

2.2.3. Sprint 2. Desarrollo de fabricante, proveedores y productos

En este *sprint* se llevó a cabo las funcionalidades referentes a los productos, sus fabricantes y proveedores. Las historias de usuario que se cumplieron en esta iteración fueron las de creación, edición y eliminación de los productos, los fabricantes y los proveedores.

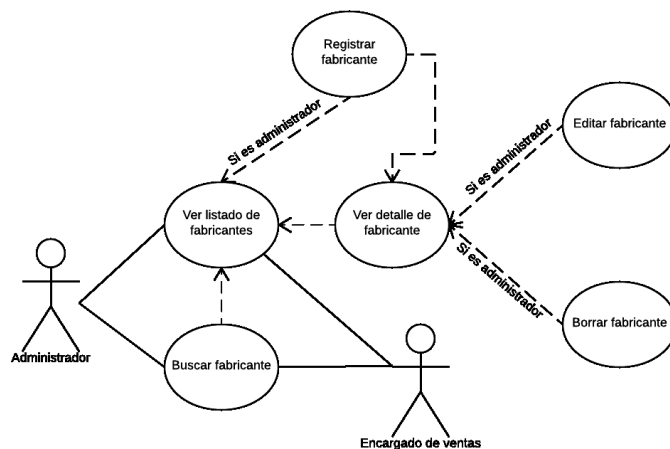
2.2.3.1. Casos de uso de fabricantes, proveedores y productos

Figura 15: Casos de uso para proveedores



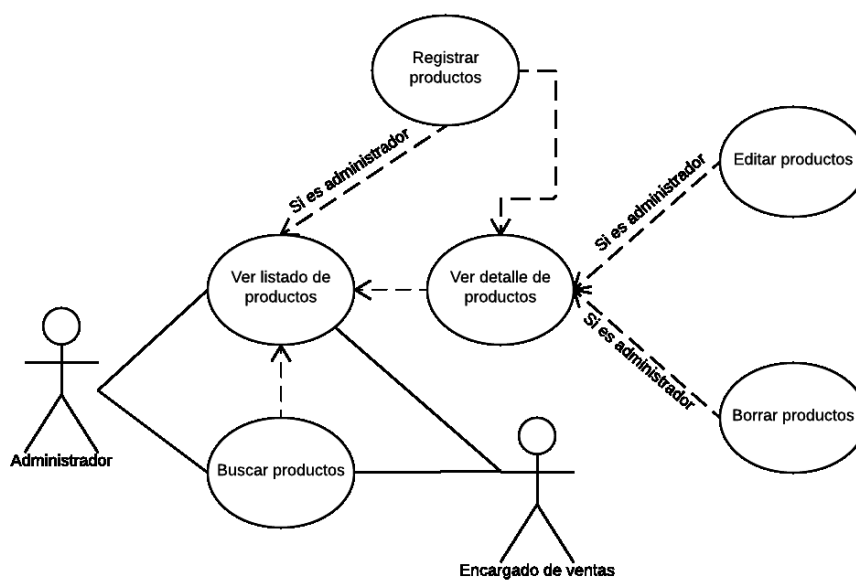
Fuente: Elaboración propia

Figura 16: Casos de uso para fabricantes



Fuente: Elaboración propia

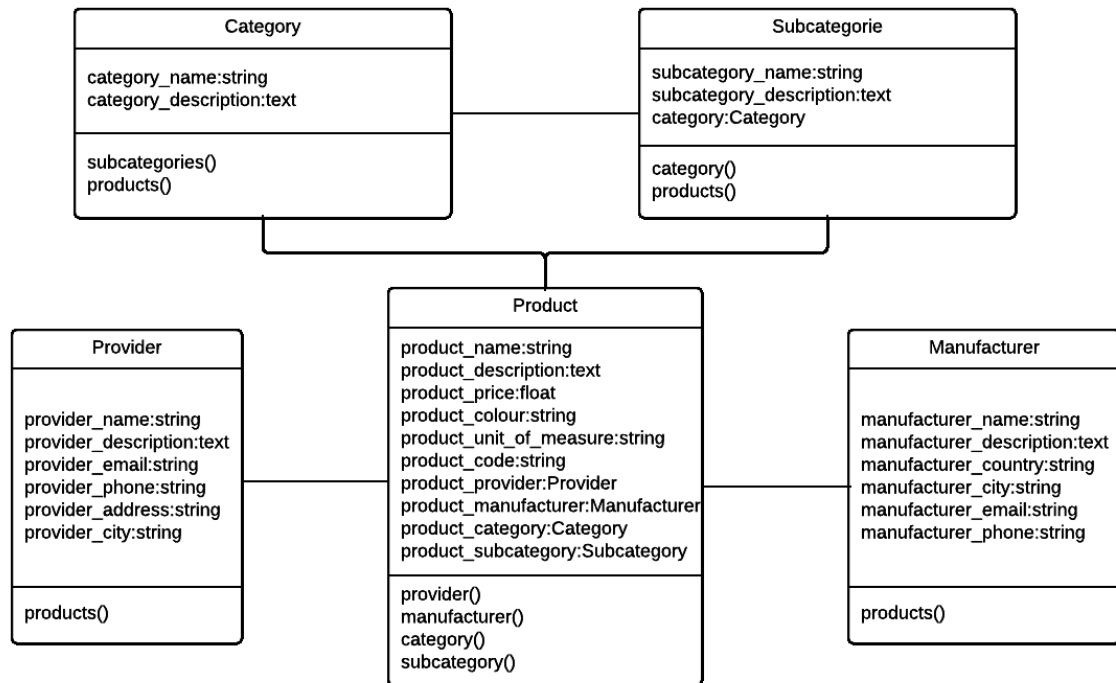
Figura 17: Casos de uso para productos



Fuente: Elaboración propia

2.2.3.2. Diagrama de clases de proveedores, fabricantes y productos

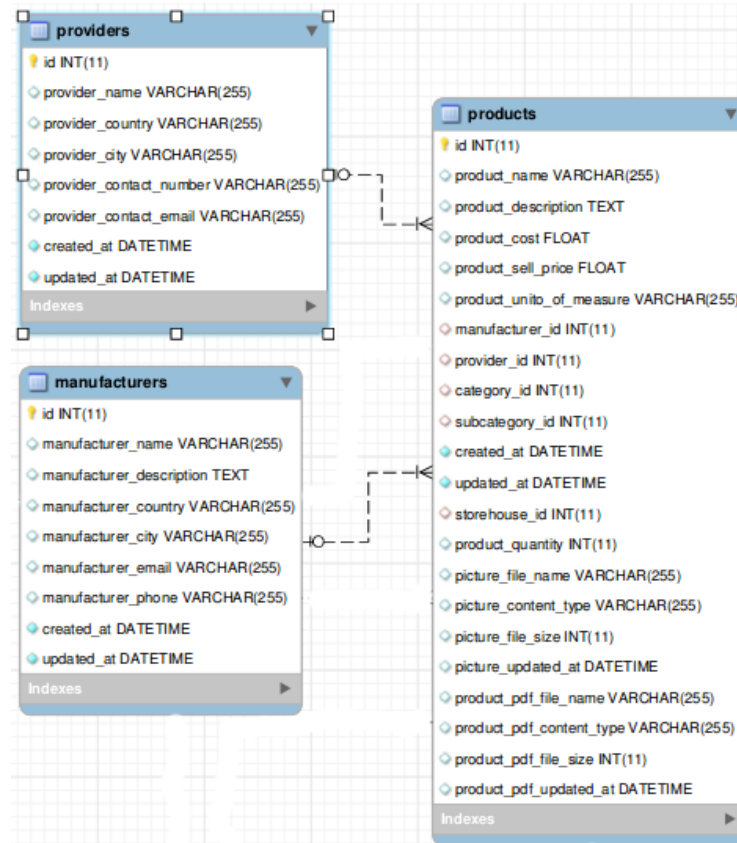
Figura 18: Diagrama de clase de proveedores, fabricante y productos



Fuente: Elaboración propia

2.2.3.3. Diagrama de bases de datos de proveedores, fabricantes y productos

Figura 19: Diagrama de bases de datos de proveedores, fabricante y productos



Fuente: Elaboración propia

2.2.3.4. Arquitectura de proveedores, fabricante y productos

De la misma forma que el anterior *sprint*, en este se utiliza la arquitectura Modelo Controlador, ya que solo se requiere generar archivos JSON, los cuales son accedidos por el *frontend*.

El *frontend* utiliza Modelo Vista Controlador por defecto, por lo cual tampoco hubo modificaciones en cuanto al *sprint* anterior.

2.2.3.5. Pruebas de proveedor, fabricante y productos

Tal como en el anterior *sprint*, en este se mantuvo el uso de *Minitest* para realizar las pruebas unitarias pertinentes, probando los modelos *Manufacturer*, *Provider* y *Product*, además de los controladores *ManufacturersController*, *ProvidersController* y *ProductsController*. Se realizaron 20 pruebas y todas pasaron de manera exitosa.

2.2.3.6. Retrospectiva

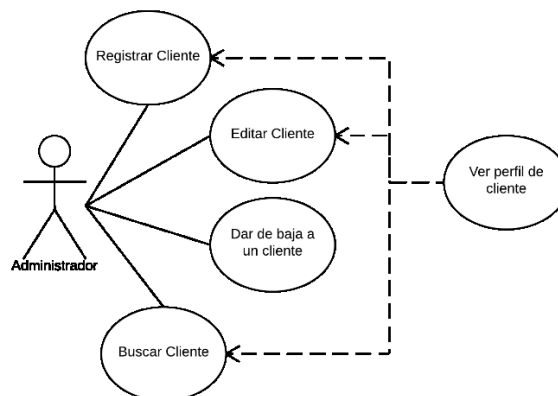
Este *sprint* al igual que el primero, fue culminado de manera satisfactoria, sin complicaciones y en el tiempo establecido.

2.2.4. Sprint 3. Desarrollo del módulo de clientes

En esta iteración se desarrollaron las funcionalidades referentes a los casos de uso para la creación, edición y eliminación de clientes. Esta iteración es muy importante para las iteraciones siguientes, por lo cual se le asignó más tiempo de entrega.

2.2.4.1. Casos de uso para clientes

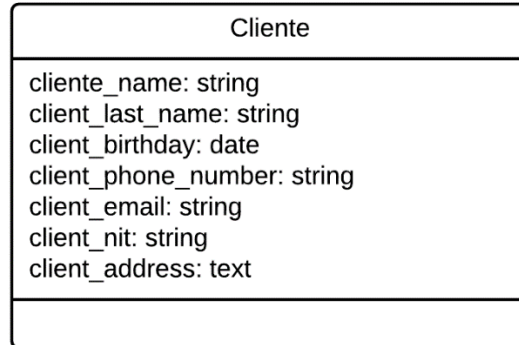
Figura 20: Diagrama de casos de uso para clientes



Fuente: Elaboración propia

2.2.4.2. Diagrama de clases

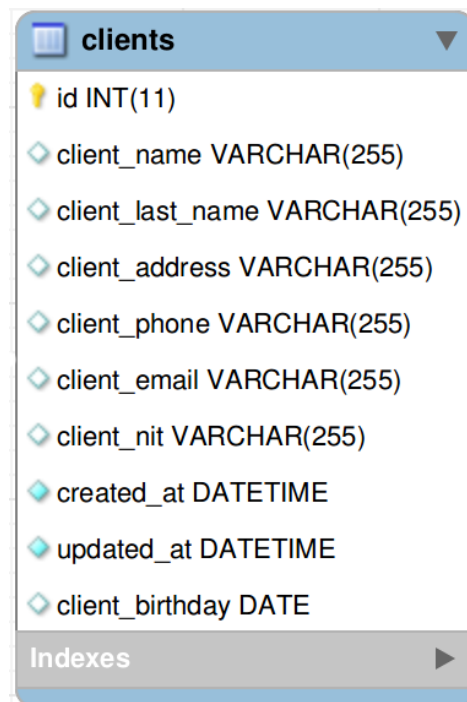
Figura 21: Diagrama de clases para clientes



Fuente: Elaboración propia

2.2.4.3. Diagrama de bases de datos

Figura 22: Diagrama de clase de clientes



Fuente: Elaboración propia

2.2.4.5. Pruebas de clientes

Como en *sprints* anteriores, se procedió a desarrollar los tests utilizando *Minitest*. Una vez que estos tests se desarrollaron se procedió a ejecutarlos; los resultados obtenidos fueron 10 pruebas ejecutadas y todas pasaron correctamente.

2.2.4.6. Retrospectiva

A pesar del cuidado especial que debía dársele a esta iteración, se pudo cumplir satisfactoriamente con el tiempo establecido y se desarrolló exitosamente las funcionalidades.

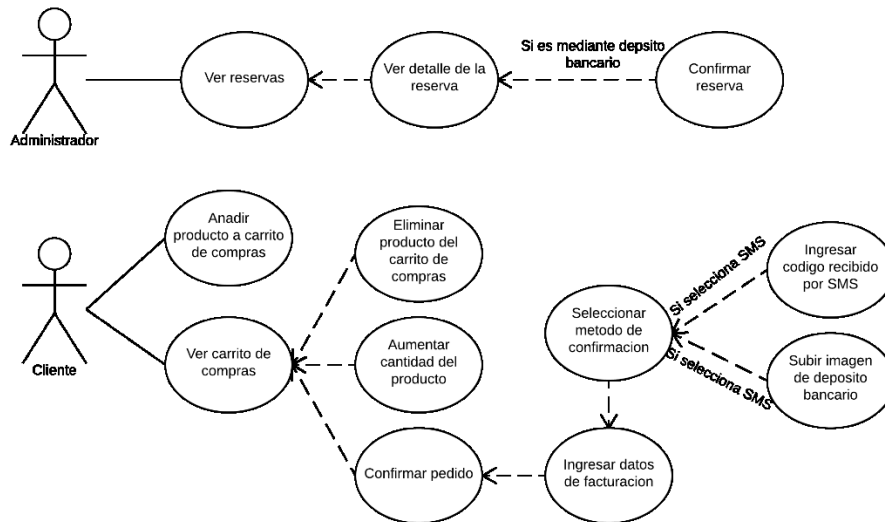
2.2.5. Sprint 4. Desarrollo del módulo de ventas

Para el desarrollo de este módulo, se trabajó tanto en el sistema desarrollado en *Ruby on Rails* como en el sistema que consume los servicios, es decir el desarrollado en *EmberJS*. Además, al ser la iteración más importante se le dio más tiempo que a iteraciones anteriores para su desarrollo. Las historias de usuario para este módulo son las de adición de productos a un carrito de compras para su posterior confirmación, mediante un SMS al teléfono móvil o una imagen del depósito bancario de la reserva.

En este módulo también se trabajaron los movimientos que se registran al realizar ventas, tanto de salidas, entradas y devoluciones.

2.2.5.1. Casos de uso para ventas

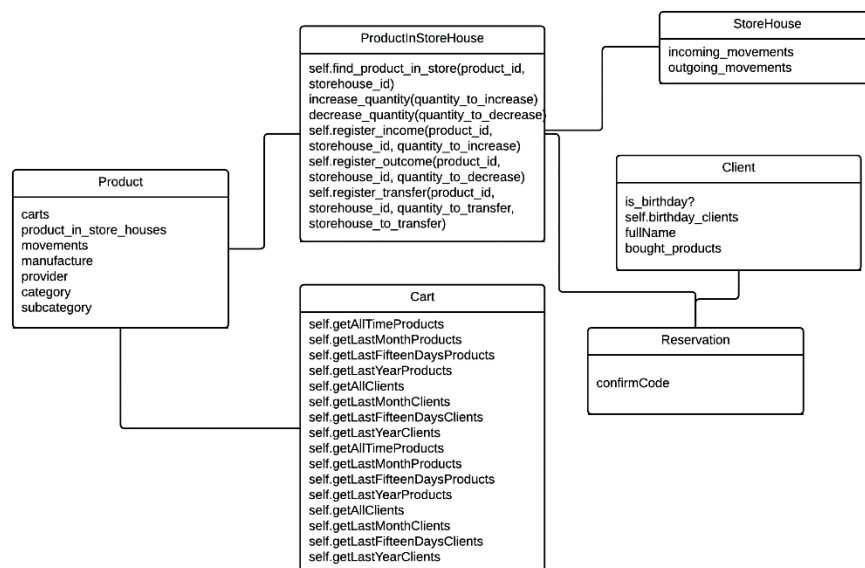
Figura 23: Diagrama de casos de uso para ventas



Fuente: Elaboración propia

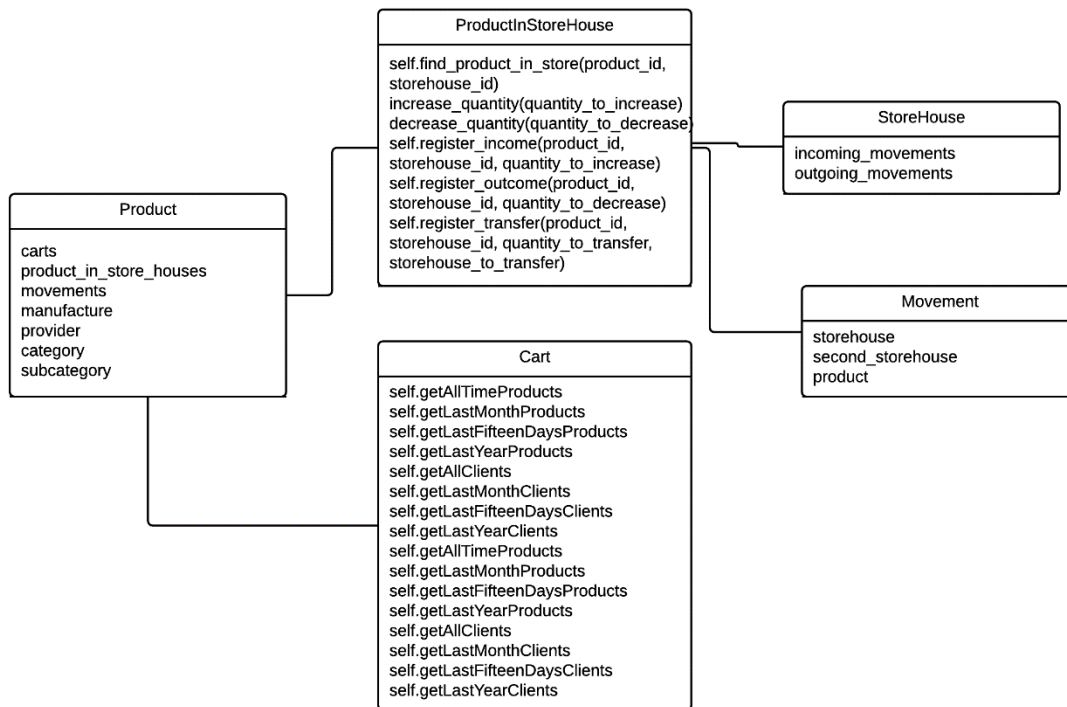
2.2.5.2. Diagrama de clases

Figura 24: Diagrama de clases para ventas



Fuente: Elaboración propia

Figura 25: Diagrama de clases para movimientos

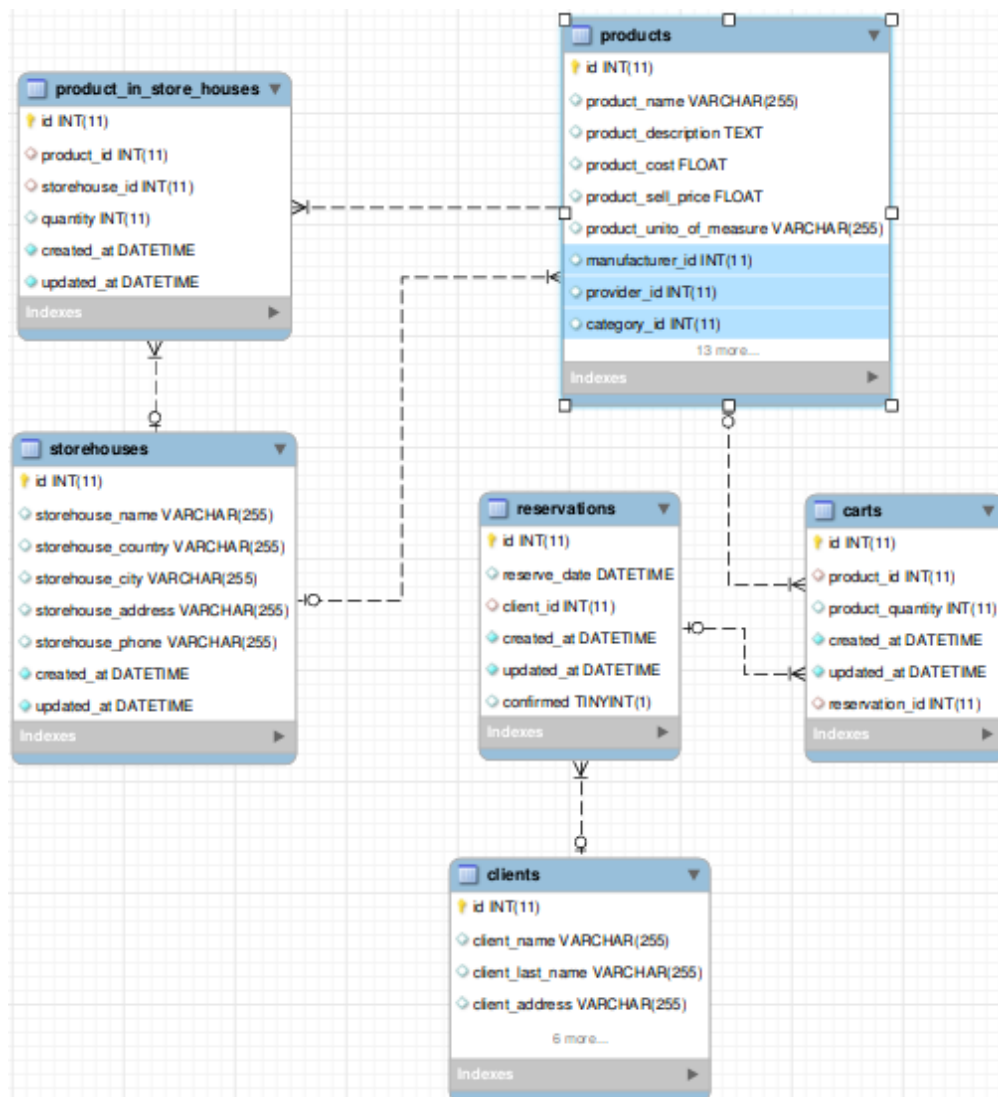


Fuente: Elaboración propia

Respecto al desarrollo de este módulo, para el manejo del carrito de compras para el cliente, la persistencia de estos datos se realiza en el navegador que el usuario está usando, es decir se usa el *LocalStorage* del marco de trabajo *EmberJS*, el cual accede a la memoria del navegador.

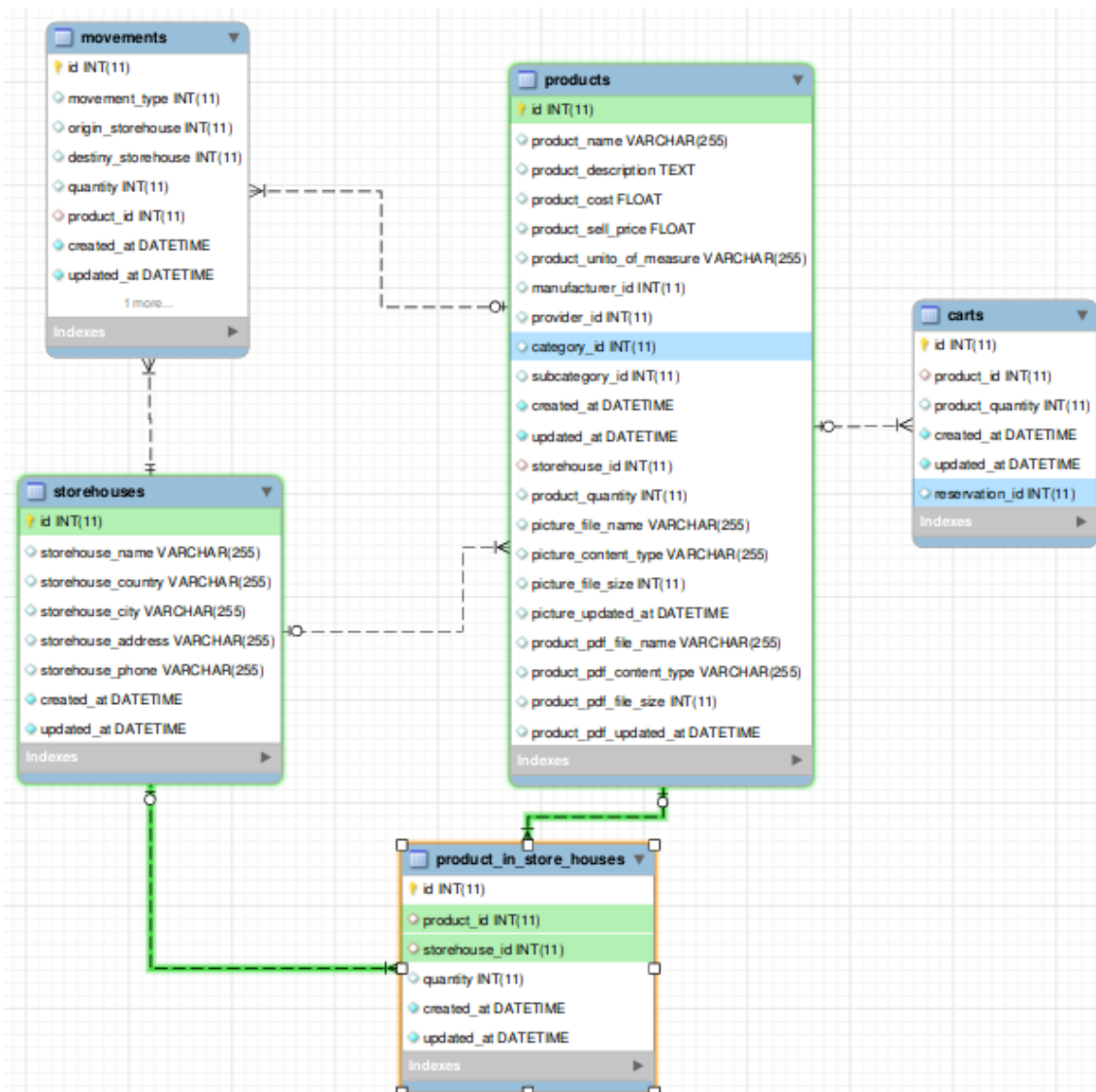
2.2.5.3. Diagrama de bases de datos

Figura 26: Diagrama de bases de datos para ventas



Fuente: Elaboración propia

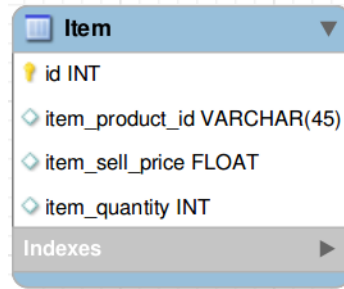
Figura 27: Diagrama de base de datos para movimientos



Fuente: Elaboración propia

Para manejar el carrito de compras desde el sistema desarrollado con EmberJS se procedió a usar la memoria del navegador, es decir el *LocalStorage*. Esta memoria nos permite almacenar información dentro del navegador, de manera que, si un usuario cerrara su navegador y no confirmara su pedido, los productos en el carrito se mantendrían en la memoria, para su posterior uso. *LocalStorage* permite almacenar desde 5 Mb de información hasta 10 Mb (Chromium Project, 2013).

Figura 28: Tabla utilizada para almacenar productos en *LocalStorage*



Item	
id	INT
item_product_id	VARCHAR(45)
item_sell_price	FLOAT
item_quantity	INT
Indexes	

Fuente: Elaboración propia

Debido al uso de la memoria del navegador se vio necesario realizar pruebas en distintos navegadores, además de tener claro que navegadores son compatibles con esta funcionalidad. Esta información puede observarse en la siguiente tabla:

Tabla 4: Navegadores y *LocalStorage*

NAVEGADOR	LOCALSTORAGE SOPORTADO DESDE LA VERSIÓN	VERSIÓN ACTUAL	PROBADO EN EL ESTE PROYECTO (ULTIMA VERSION)
Google Chrome	4	51	Sí
Firefox	3.5	46	Sí
Internet Explorer	8	11	Sí
Opera	10.50	37	Sí
Safari	4	9.1	Sí

Fuente: (Mozilla Developers, 2016)

Observando la información de la Tabla 4 y habiendo probado todos los navegadores en sus últimas versiones, además teniendo el conocimiento de las versiones desde las que se soporta la funcionalidad de *LocalStorage*, se puede proseguir con el uso de esta funcionalidad sin el riesgo de incompatibilidad con muchos navegadores.

2.2.5.4. Pruebas de ventas y movimientos

Las pruebas realizadas en este módulo fueron realizadas utilizando *Minitest* en la parte desarrollada con *Ruby on Rails*, y utilizando las opciones de pruebas unitarias

proporcionadas por el *framework EmberJS*. Se desarrollaron 12 pruebas con *Minitest* y 10 pruebas con *EmberJS*, habiendo pasado todas correctamente.

2.2.5.6. Retrospectiva

La culminación de este *sprint* tomo más tiempo del establecido inicialmente, debido especialmente a problemas con el envío de un SMS al teléfono móvil. Esto retrasó otras tareas referentes a esta iteración, por lo cual fueron necesarias dos semanas más de lo planeado inicialmente, pero finalmente se pudo cumplir con todas las funcionalidades.

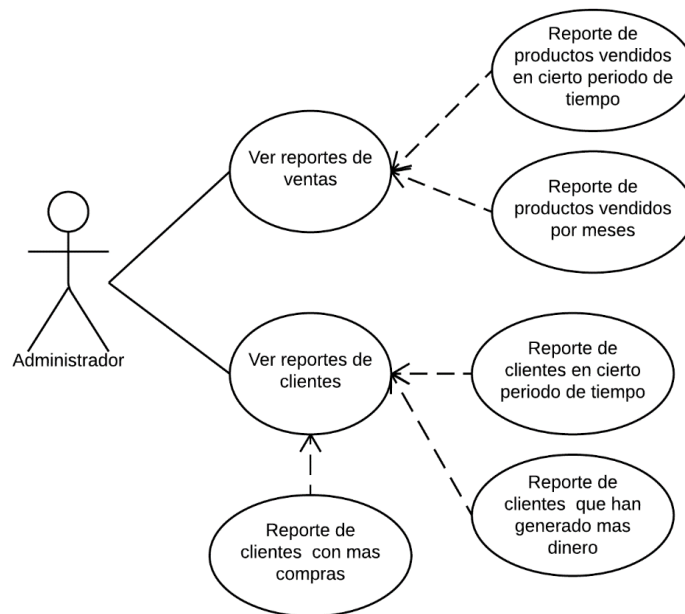
2.2.6. Sprint 5. Desarrollo del módulo de reportes

Para el desarrollo de este módulo, no resultó necesario crear tablas en la base de datos, debido a que solo se procesa la información ya existente en el sistema, por lo cual para esta sección se obviaron los puntos de diagramas de clases y de bases de datos. Donde se procesa los datos para la generación de reportes es en el controlador denominado *MainController*, que hasta este módulo solo permitía mostrar vistas estáticas en el sistema.

Esta iteración contempla las historias de usuario referentes a los reportes que el sistema genera para el administrador, pudiendo ver reportes sobre ventas, clientes y productos.

2.2.6.1. Casos de uso para reportes

Figura 29: Diagrama de casos de uso para reportes



Fuente: Elaboración propia

2.2.6.2. Pruebas

Para las pruebas en este módulo, se creó un archivo nuevo denominado *ReportsTest* de manera que en este se pueda desarrollar las pruebas necesarias para la funcionalidad de reportes.

2.2.6.3. Retrospectiva

Debido al retraso en la iteración anterior, este *sprint* tuvo que modificarse para poder adecuarse a los plazos establecidos, pero esto no generó ningún problema, ya que las tareas pudieron realizarse de manera exitosa y antes de lo planeado.

CONCLUSIONES

Luego de haber concluido con el desarrollo del sistema, se dio cuenta de que los objetivos específicos han sido cumplidos, llevando así a la culminación satisfactoria del objetivo general.

Durante el desarrollo se pudo plantear un modelo de oferta y adquisición de productos que se adapte al mercado boliviano, para permitir a los clientes adquirir productos, haciendo uso de mensajes a celular e imágenes de depósitos bancarios (Figura 8 y Figura 9).

Respecto al desarrollo de los objetivos específicos relacionados con los módulos del sistema, cada uno fue culminado de manera satisfactoria, cumpliendo con los casos de uso y los elementos del *product backlog*.

El control de stock para los productos permite la gestión correcta de los productos y todos sus movimientos (Figura 18 y Figura 19).

Las ventas son registradas correctamente, haciendo uso de ambos métodos planteados para la confirmación y este módulo se comunica correctamente con el resto del sistema.

Respecto a la metodología, se pudo evidenciar que la elección fue adecuada, ya que, debido a las bases de la misma, se pudo adaptar el sistema a cambios que surgieron durante el desarrollo.

Finalmente, el modelo SOA permitió desarrollar el sistema sin mayores dificultades, ya que mientras se sigan sus fundamentos, el sistema que se va construyendo es siempre apto a modificaciones y a ser escalado, debido a que los cambios más grandes son realizados en el *backend*, en cambio la interfaz y aplicaciones que consumen los servicios prácticamente no deben ser modificadas.

RECOMENDACIONES

Las recomendaciones de este proyecto pasan por su ampliación, ya que debido al uso de modelo SOA, es posible ampliar el sistema e integrarlo con otros sistemas.

Se recomienda a futuro implementar un módulo contable, para realizar el control de la contabilidad de la empresa. Además, podría implementarse un sistema móvil, es decir una aplicación móvil para la venta de productos.

Respecto al sistema actual, es posible expandir los mecanismos de comunicación, ya que, al momento de la finalización de este proyecto, se utiliza el tipo de archivos JSON, pero esto podría ser ampliado para utilizar XML.

BIBLIOGRAFÍA

Agile For All. 2016. Agile For All. [En línea] 2016.

Aldrich, Michael. 2011. The Michael Aldrich Archive. [En línea] Noviembre de 2011. http://www.aldricharchive.com/inventors_story.html.

Api pie. 2016. Api pie Github. [En línea] 2016. <https://github.com/Api pie/apipie-rails>.

Association of Modern Technologies Professionals. 2015. IT Knowledge Portal. [En línea] 2015. <http://www.itinfo.am/eng/software-development-methodologies/>.

Chromium Project. 2013. Chromium Code Reviews. [En línea] 6 de 2013. <https://chromiumcodereview.appspot.com/21680002>.

EmberJS. 2014. EmberJS. [En línea] 2014. <https://guides.emberjs.com/v2.2.0/getting-started/core-concepts/>.

Fielding, Roy Thomas. 2000. Architectural Styles and The Design of Network-based Software Architectures. [En línea] 2000.

Fowler, Martin y Lewis, James. 2014. Martin Fowler. *Microservices*. [En línea] 25 de Marzo de 2014.

Hot Frameworks. 2016. Hot Frameworks. [En línea] 2016.

IBM. 2016. Introducción a SOA y servicios web. [En línea] 2016. <http://www.ibm.com/developerworks/ssa/webservices/newto/service.html>.

—. **2015.** RESTful Web services: The basics. *RESTful Web services: The basics*. [En línea] 9 de Febrero de 2015. <http://www.ibm.com/developerworks/library/ws-restful/>.

Javatpoint. 2015. SOAP vs REST Web Services. [En línea] 2015. <http://www.javatpoint.com/soap-vs-rest-web-services>.

Mozilla Developers. 2016. Mozilla Developers Network. [En línea] 10 de 6 de 2016. [Citado el: 15 de 6 de 2016.] <https://developer.mozilla.org/es/docs/Web/API/Window/localStorage>.

Novosiolov, Andrzej. 2015. SoftElegance's Blog. [En línea] 17 de Febrero de 2015. <http://blog.softelegance.com/angularjs/angularjs-advantages-and-limitations/>.

Oasis Group. 2006. *Reference Model for Service Oriented Architecture 1.0*. 2006.

Oracle MySQL. 2016. MySQL Developers. [En línea] 2016. <http://dev.mysql.com/doc/refman/5.7/en/features.html>.

Rodriguez, Alex. 2015. IBM. *IBM*. [En línea] 9 de 2 de 2015.
<http://www.ibm.com/developerworks/library/ws-restful/>.

Schwaber, Ken y Sutherland, Jeff. 2013. Scrum Guides. *La guía de Scrum*. [En línea] Julio de 2013.

Search IT Channel. 2007. What are the top MySQL features? What is MySQL? [En línea] Enero de 2007. <http://searchitchannel.techtarget.com/feature/What-are-the-top-MySQL-features-What-is-MySQL>.

Shivers, Tom. 2011. Capture Commerce. [En línea] 2011.
<http://www.capturecommerce.com/blog/general/forecast-for-global-ecommerce-growth/>.

Teresa Ferreiro, Mario Tanco. 2008. Magrama. [En línea] 2008.

The Open Group. 2015. SOA Reference Architecture Technical Standard : Description of Layers. [En línea] 2015. <https://www.opengroup.org/soa/source-book/soa/soa.htm>.

—. **2015.** What is SOA? [En línea] 2015. <https://www.opengroup.org/soa/source-book/soa/soa.htm>.

Twilio. 2016. Twilio. [En línea] 2016. <https://www.twilio.com/api>.

—. **2016.** Twilio-ruby. [En línea] 2016. <https://github.com/twilio/twilio-ruby>.

Wikipedia. 2015. ASP.NET. [En línea] 10 de Diciembre de 2015.
<https://es.wikipedia.org/wiki/ASP.NET>.

—. **2016.** Convention Over Configuration. [En línea] 8 de Enero de 2016.
https://en.wikipedia.org/wiki/Convention_over_configuration.

—. **2016.** E-commerce. [En línea] 2016. <https://en.wikipedia.org/wiki/E-commerce>.

—. **2016.** Emberjs. [En línea] 2016. <https://en.wikipedia.org/wiki/Ember.js>.

—. **2015.** Intercambio electrónico de datos. [En línea] 5 de Noviembre de 2015. [Citado el: 9 de Enero de 2016.]
https://es.wikipedia.org/wiki/Intercambio_electr%C3%B3nico_de_datos.

—. **2016.** MySQL. [En línea] 14 de 5 de 2016. <https://es.wikipedia.org/wiki/MySQL>.

—. **2015.** Ruby. [En línea] 31 de Diciembre de 2015. <https://es.wikipedia.org/wiki/Ruby>.

—. **2015.** Service-oriented Architecture. [En línea] 29 de Diciembre de 2015.
https://en.wikipedia.org/wiki/Service-oriented_architecture.

ANEXOS

Anexo 1. Estudio sobre el comercio electrónico en Latinoamérica

Fuente: Instituto Latinoamericano de comercio electrónico

PAIS	TARJ. CREDITO	TARJ. DÉBITO	CAJ. AUTOM. (UNIDADES)	ÍNDICE SIST. BANCARIO	BANCA M- CHA MÓVIL	E-COMPR- DORES	COMPRAS POR INTERNET	ÍNDICE ADOPCION TECNOLÓGICA	PAGADORES DE IMPUES- TOS ONLINE	GRANDES RETAILERS ONLINE	ÍNDICE POTENCIA OFERTA	ÍNDICE E-READINESS
ARGENTINA	48.8%	48.7%	15.800	0,24	3,32%	3,88%	875	0,32	0	2	0,09	0,46
BOLIVIA	2,2%	12,7%	988	0,04	0,32%	1,50%	44	0,28	0	0	0,08	0,17
BRASIL	71,2%	123,0%	174.255	0,97	4,27%	9,73%	13.230	0,61	0	10	0,33	0,95
CHILE	53,2%	48,0%	7.562	0,70	3,41%	7,02%	1.828	0,55	131.974	6	1,06	0,63
COLOMBIA	16,6%	33,0%	9.274	0,28	1,85%	4,02%	435	0,28	0	1	0,07	0,36
REP. DOMINICANA	16,7%	30,8%	2.800	0,17	1,18%	4,16%	289	0,34	0	0	0,08	0,31
ECUADOR	14,2%	19,7%	1.340	0,13	0,68%	2,50%	71	0,15	0	0	0,08	0,26
GUATEMALA	8,6%	12,5%	1.254	0,09	0,76%	2,02%	77	0,21	0	1	0,06	0,25
HONDURAS	8,1%	11,6%	735	0,18	0,71%	1,99%	28	0,28	0	1	0,06	0,25
MÉXICO	10,8%	53,0%	39.856	0,28	0,99%	4,30%	2.625	0,31	418.972	11	0,22	0,53
NICARAGUA	10,8%	12,3%	568	0,18	0,15%	1,79%	12	0,15	0	1	0,06	0,20
PANAMÁ	15,7%	56,5%	944	0,62	1,52%	2,94%	102	0,26	0	0	0,00	0,43
PERU	16,1%	46,9%	3.763	0,22	0,53%	3,14%	276	0,28	34.049	1	0,13	0,34
PUERTO RICO	33,8%	42,5%	1.478	0,42	4,91%	3,30%	588	0,59	0	0	0,00	0,50
PARAGUAY	12,2%	11,7%	526	0,14	0,61%	2,84%	38	0,24	0	0	0,00	0,27
EL SALVADOR	7,2%	14,9%	694	0,18	0,88%	2,84%	46	0,33	0	1	0,06	0,30
URUGUAY	51,5%	43,9%	1.985	0,58	4,85%	5,12%	82	0,36	0	0	0,00	0,48
VENEZUELA	24,5%	51,0%	15.124	0,34	0,86%	5,16%	908	0,34	0	0	0,00	0,45
AMERICA LATINA	37,6%	69,6%	277.266	0,51	2,59%	5,99%	21.775	0,41			0,22	0,62
ESPAÑA	95,4%	67,0%	61.374	1,08	11,19%	16,01%	8.400	1,00	5.630.896	-	1,00	1,00
EE.UU.	187,8%	165,1%	500.000	2,14	14,59%	74,00%	144.420	2,25	98.000.000	-	2,25	2,12