UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMÁTICA

TESIS DE GRADO

"APLICACIÓN MÓVIL PARA EL CONTROL DE INVENTARIOS BASADOS EN LA TECNOLOGÍA DE IDENTIFICACIÓN POR RADIOFRECUENCIA"

PARA OPTAR AL TÍTULO DE LICENCIATURA EN INFORMÁTICA
MENCIÓN INGENIERÍA DE SISTEMAS INFORMÁTICOS

POSTULANTE: PAOLO JHONATAN RAMOS MENDEZ
TUTORA METODOLÓGICA: M.SC. ROSA FLORES MORALES
ASESOR: PH.D. YOHONI CUENCA SARZURI

LA PAZ – BOLIVIA
2017

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMÁTICA

LA CARRERA DE INFORMÁTICA DE LA FACULTAD DE CIENCIAS PURAS Y NATURALES PERTENECIENTE A LA UNIVERSIDAD MAYOR DE SAN ANDRÉS AUTORIZA EL USO DE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO SI LOS PROPÓSITOS SON ESTRICTAMENTE ACADÉMICOS.

LICENCIA DE USO

El usuario está autorizado a:

a) visualizar el documento mediante el uso de un ordenador o dispositivo móvil.
b) copiar, almacenar o imprimir si ha de ser de uso exclusivamente personal y privado.
c) copiar textualmente parte(s) de su contenido mencionando la fuente y/o haciendo la referencia correspondiente respetando normas de redacción e investigación.

El usuario no puede publicar, distribuir o realizar emisión o exhibición alguna de este material, sin la autorización correspondiente.

DEDICATORIA

A Dios por todo, a mis padres Silver y Paulina, a mis hermanos Rudy, Lizeth y Rebeca por el apoyo en estos años de estudio.

AGRADECIMIENTO

Agradecer a Dios por darme fuerzas en los momentos más difíciles de mi vida y en el proceso de investigación que se realizó de la presente tesis como en los años de estudio.

A mi tutora M. Sc. Rosa Flores Morales que estuvo apoyándome en el proceso de realización del presente proyecto, muchas gracias por su tiempo, y sobre todo las meditaciones que nos daba para leer y seguir adelante.

A mi asesor PH. D. Yohoni Cuenca Sarzuri por aceptar el asesoramiento del presente trabajo, por su colaboración y brindarme el apoyo, para culminar la elaboración de la presente tesis.

A mi familia que estuvo presente en todo momento estos años de estudio, que a pesar de todos los obstáculos de la vida hay me enseñaron que se debe seguir adelante, agradezco a mis padres por su apoyo.

A mi hermano Rudy quien estuvo colaborándome con la guía y dirección del presente trabajo, la persona a la cual admiro y aprecio mucho y que me dedicó tiempo para ayudarme y estuvo ahí para levantarme cuando ya no podía más.

A mis amigos quienes me colaboraron con la presente investigación, por los momentos de apoyo y cariño que me ofrecieron.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

RESUMEN

En nuestro país, el uso de tecnologías a través de dispositivos móviles para el sector de medianas y pequeñas empresas es un campo recién explorado y además de innovador. Esta investigación propone el uso de dispositivos *RFID* mediante una aplicación móvil para facilitar las tareas de inventario de los productos del sector de importadoras. Si bien algunas empresas nacionales ya adoptan esta tecnología, la misma aún no ha cobrado la fuerza necesaria debido a la falta de mayor información para la implementación de otras tareas más complejas y necesarias del rubro.

Las medianas y pequeñas empresas dedicadas a la importación de productos llevan sus registros de inventario con riesgo a errores humanos involuntarios o hurto, porque no cuentan con un sistema automatizado que permita registrar el producto que ingresa y sale de sus sectores de venta o almacén, por lo tanto, el inventario de los productos que es revisado al finalizar el día, no llega a ser oportuna ni precisa por ser un registro manual e inexacto. Finalmente, este proceso apacible a controles dificulta el compartimiento de información del inventario en toda la empresa, ya que la falta de informatización hace que el inventario sea un proceso complicado.

Por lo tanto, esta investigación plantea demostrar las bondades tecnológicas que tiene una aplicación móvil en *iOS* capaz de registrar, contabilizar y ubicar los productos inventariados a través del uso de la tecnología *RFID* además de implementar el control y opción de búsqueda avanzada. Con los resultados obtenidos, se afirma que es posible el optimizar el control de inventarios basados en la tecnología *RFID,* ya que el usuario logra obtener información tanto de los ítems registrados en una determinada zona, como la búsqueda y el conteo de los ítems en un almacén, depósitos o sucursales. Además de la sincronización de dicha información para actualizar la información de un inventario.

Para el desarrollo de la aplicación móvil que permita optimizar el control de inventarios basados en la tecnología *RFID,* es necesario aplicar conceptos de *IoT* (Internet de las cosas) para los futuros proyectos el cual se invita a profundizar sobre el uso de esta tecnología no solo para el área de inventarios, sino que puedan ser aplicados en el área de la medicina, como también en el uso de marketing digital que puede ser de vital importancia de modo que el estudiante motive su aprendizaje.

Al concluir esta investigación, se sugiere buscar mayor innovación en el uso de aplicaciones móviles en nuestro país para el reemplazo de trabajos manuales o el uso de hojas de cálculos y dar al usuario mayor facilidad de herramientas con las cuales pueda mejorar su desempeño en el trabajo. El uso de nuevas tecnologías no debe ser extraño en el diario vivir.

Palabras clave: Aplicación Móvil, *RFID, Tag RFID,* Inventario.

ABSTRACT

In our country, the use of technologies through mobile devices for the medium and small enterprises sector is a newly explored field and also innovative. This research proposes the use of RFID devices through a mobile application to facilitate the products inventory tasks of the importers sector. Although some national companies already adopt this technology, it has not yet gained the necessary strength because of the lack of information for the implementation of other important tasks.

These medium and small companies which imports products do this process manually because they do not have an automated system that allows them to register the product that enters and leaves the store; therefore, the information that is reviewed at the end of the day does not become timely or accurate. Generally, this registration is manual and inaccurate which may cause losses. On the other hand, they do not have a record of sold out products, which leads to other problems since the importer usually runs out of a virtual resource at the wrong time. Finally, it makes difficult to share inventory information across the Company, since lack of computerization makes inventory access a complicated process.

Therefore, this research raises the technological benefits of a mobile application in iOS capable of registering, accounting and locating the inventoried products through the use of RFID technology, in addition to implementing the control and advanced search option. With the results obtained, it is stated that it is possible to optimize the inventory control based on RFID technology, since the user obtains information from both the items registered in a given area, and the searching and counting items in a facility. In addition to the synchronization of this information to update the information of an inventory.

For the mobile application development that allows to optimize the inventory control based on the RFID technology, it is necessary to apply concepts of IoT (Internet of Things) for the future projects which invites to deepen on the use of this technology not only for the inventory area, but also to be applied in the area of medicine, as well as in the use of digital marketing which can be of vital importance for the students to motivate their learning.

At the conclusion of this research, it is suggested to look for greater innovation in the use of mobile applications in our country in order to replace manual jobs or the use of spreadsheets and give users greater better tools to improve his work performance. The use of new technologies should not be strange in daily life.

Key Word: Mobile Application, *RFID*, *Tag RFID*, Inventory

CAPITULO I
MARCO INTRODUCTORIO

1.1. Introducción

En las últimas décadas, se ha producido un enorme desarrollo en los campos de la logística y la gestión de productos, debido a su creciente importancia para cada empresa de forma individual y del mercado en su conjunto. La mayoría de ellos están invirtiendo continuamente en la mejora y optimización de estrategias de logística y procesos. Por lo tanto, la introducción de diversas tecnologías que pretenden mejorar el control y gestión de productos ha sido el resultado natural de esta tendencia hacia la oferta en muchas compañías

Es por ello que la tecnología ha llegado a ser imprescindible en las tareas cotidianas. En cualquier ámbito se pueden encontrar avances tecnológicos que permiten simplificar y mejorar las actividades diarias lo cual proporciona mayores ganancias.

Una de las nuevas tecnologías introducidas dentro de este ámbito es la Tecnología de Identificación por Radiofrecuencia (*RFID*). El uso de esta tecnología se ha convertido en un asunto de compatibilidad de la industria. Además, la tecnología está llegando a ser más atractiva debido a su capacidad de apoyar a las empresas en optimizar procesos de negocio vía su visibilidad en tiempo real, control de su inventario y rastreo a través de la cadena de suministro, administración de activos, por ejemplo, compañías como *Inditex*[1], empresa española dedicada a la fabricación textil completó la incorporación de *RFID* en la red de sus sucursales de *Zara* e inició su extensión a sus sucursales de Massimo Dutti y Uterqüe.

[1] *Inditex S. A.*, acrónimo de Industria de Diseño Textil, Sociedad Anónima, es un grupo multinacional español de fabricación y distribución textil. Tiene su sede central en el Polígono Industrial de Sabón, en Arteijo, La Coruña, España.

En 2017, la compañía prevé continuar la implantación con su tienda Pull&Bear, para saltar en 2018 a Stradivarius, Bershka y Oysho. Todas las cadenas de *Inditex* completarán el proceso en agosto de 2020 (DIR&GE, 2017).

Otra de las empresas que se encuentra expandiendo el uso de la tecnología *RFID* y que está dedicada al rubro de ventas por menor es la de Marks & Spencer[2] (M&S), esta empresa, con su base en Reino Unido. Dicha empresa está dedicada a la venta al por menor líderes en el Reino Unido la cual ofrece ropa y productos para el hogar con una gran relación calidad-precio, también se encuentra expandiendo el uso de la tecnología *RFID* en todas sus sucursales (Journal, 2014).

El uso de *RFID* consiste en un dispositivo de radio que se comunica con un transpondedor[3] o etiqueta que contiene un procesador compuesto de un chip y una antena. El propio transpondedor es una ampliación de las etiquetas con códigos de barras que tienen todos los productos, solo que con mayor alcance en almacenamiento de información. Una de las mayores ventajas de estos sistemas es que, a diferencia de la captura de datos basada en códigos de barras, el sistema *RFID* puede leer la información de la etiqueta sin que exista ninguna línea de visión ni una orientación determinada, lo que trae como consecuencia que los sistemas *RFID* se pueden automatizar en gran medida, reduciendo la necesidad de lectura manual de aquellos productos que contienen la etiqueta (Carreon, 2014).

Sin embargo, en nuestro medio la tecnología *RFID* aún no ha acaparado el mercado, por el momento las empresas aún no han apostado por el uso de dicha tecnología para la simplificación y optimización de su proceso.

---

[2] *Marks and Spencer* es una multinacional británica dedicada al comercio minorista con sede en Westminster, Londres. Cotiza en la Bolsa de Londres y forma parte del índice FTSE 100.

[3] Un transpondedor o transponder es un tipo de dispositivo utilizado en telecomunicaciones cuyo nombre viene de la fusión de las palabras inglesas *Transmitter* (Transmisor) y *Responder* (Contestador/Respondedor).

Dentro de este contexto, el presente trabajo tiene como propósito principal el examinar los usos de esta tecnología *RFID* para la gestión y control de cualquier tipo de productos mediante una aplicación móvil que se ejecutará a través de un *Reader TSL-1128* que pueda almacenar y recuperar datos desde un *Tag RFID*, considerando las características de cada producto la cual se desea gestionar.

El equipo a usarse, *Reader TSL-1128*, puede ser utilizado mediante un dispositivo móvil en plataformas *Android*, *Windows Phone* y *iOS*[4], que es donde se desarrollará la aplicación para facilitar el control de inventarios.

Por lo tanto, el desarrollo de esta tesis comprenderá dos etapas: En la primera, que se encargará de realizar el registro de diferentes productos, los cuales serán almacenados en la aplicación móvil. Este registro se realizará utilizando etiquetas *RFID*. La segunda etapa, se encargará de realizar el escaneo de dichos productos, para así localizarlos de acuerdo a su código registrado.

1.2. Antecedentes

Hoy en día existe una gran necesidad de controlar objetos en gran cantidad como ser; automóviles, ropa, libros, etc. Sobre todo, en las grandes compañías como *M&S*, *Levi's* que manejan grandes cantidades de productos (Emb, 2017).

La tecnología *RFID* ha revolucionado la manera de transmitir información, brindando eficiencia en diversos sistemas de identificación. Las raíces de esta tecnología denotan desde la segunda guerra mundial, que identificaban los aviones enemigos, a diferencia de los radares que no estaban en capacidad de distinguirlos (Roberti, 2014).

[4] *Android, Windows Phone y iOS* hace referencia a sistemas operativos que se emplea en dispositivos móviles.

En el año 1970, Estados Unidos realizó una investigación para rastrear materia nuclear mediante *RFID*. Paralelamente estas tecnologías se aplicaron en la industria vial, agricultura y manufactura de igual forma aportando los beneficios de rastreo (Roberti, 2014).

En el caso de la cadena de valor, esta tecnología tiene grandes ventajas, ya que ayuda a aumentar los márgenes y beneficios, así como a la reducción de costos. Aquellas empresas que usan *RFID* pueden lograr:

- Fluidez en los procesos del negocio, ya que se puede conocer con exactitud el estatus del inventario.
- Incremento de la productividad.
- Precios más competitivos al haber una reducción en los costos de operación.
- Envío de productos con mayor velocidad.
- Una mejor administración del inventario.
- Mejor servicio al cliente.

Como se ha mencionado muchas veces, la tecnología *RFID* no solamente tiene ventajas en el piso de venta dirigidas a una mejor experiencia de compra para el consumidor, sino que los mayores beneficios se encuentran a lo largo de la cadena de suministro en los procesos de fabricación y distribución de los productos.

Existen cinco ventajas en el uso de esta tecnología para las empresas inmersas en la cadena de valor, estas son:

1.2.1. Recepción

Cuando el producto llega a los Centros de Distribución, el dispositivo de lectura lee la información de la etiqueta del pallet completo y así el sistema *RFID* puede verificar inmediatamente todo el contenido de la carga, ofreciendo visibilidad en tiempo real para los sistemas de administración de inventario del *CEDIS*[5]. Automáticamente se distribuyen las actividades de acuerdo a la información leída.

1.2.2. Clasificación

La etiqueta de *RFID* de cada producto incorpora la información del mismo, origen y destino. Una vez que se encuentra en la banda clasificadora o transportadora en el *CEDIS*, se lee el destino que tendrá dicho producto y entonces el sistema identifica el flujo que deberá tener para su clasificación y entrega. Los datos también se pueden almacenar en la etiqueta para ser leídos por el cliente que reciba el pedido. Una de las ventajas más contundentes en este proceso es el que se pueda garantizar la ubicación correcta de los artículos, así como la reducción en tiempo para una clasificación más precisa.

1.2.3. Almacenaje de Productos

Al contar con la información de cada uno de los productos en algunos casos del pallet[6] completo, la actividad de almacenaje se realiza con mayor rapidez y con un grado máximo de efectividad e incluso, a lo largo del proceso, se pueden emitir avisos de acciones incorrectas. De igual forma, a la hora de preparar los pedidos para las sucursales o diferentes clientes, la velocidad con la que esto puede ocurrir disminuye el tiempo de respuesta para el envío de los productos.

---

[5] *CEDIS*, Un centro de distribución es una infraestructura logística en la cual se almacenan productos y se dan órdenes de salida para su distribución al comercio minorista o mayorista.

[6] *Pallet* estructura de agrupación de carga, fabricada generalmente con madera.

1.2.4. Gestión de Inventarios

La etiqueta de *RFID* puede adherirse a un producto, ensamblaje o caja. La ubicación y el medio de transporte de los artículos se asocian en el punto de almacenaje. Durante el proceso, los lectores realizan el seguimiento en cuanto a la ubicación, contenido y traslado. Las ventajas más significativas en este proceso es que la tecnología permite realizar un inventario rápido y preciso, además de que permite la renovación automática de pedidos.

1.2.5. Envíos

La etiqueta de *RFID* incorpora los datos del contenido de cada pallet en cuanto a origen y destino. Toda la información se añade a la etiqueta en el momento de consolidación del pedido para que se indique a los sistemas de entrega en inventario el lugar a donde se hará el envío. Todo ello garantiza una secuencia correcta de envío eliminando errores de colocación de pedidos, reduciendo las demoras en la entrega de mercancía (Carreon, 2014). En la figura 1.1. se puede observar los campos que esta tecnología ha sido aplicada.

Figura 1.1. Aplicación de la tecnología RFID
Fuente: (Carreon, 2014)

Los investigadores de Sarma & Brock descubrieron una nueva dirección del *RFID* en 1999, centrado en productos, generando importantes cambios en el negocio de grandes compañías como Wal-Mart y Tesco que utilizan esta tecnología para rastrear sus productos y tomar decisiones en tiempo real (Violino, 2014).

En julio de 2004 la *Food and Drug Administration* (Administración de Alimentos y Medicamentos) hizo pública la decisión de comenzar un proceso de estudio que determinará si los hospitales pueden utilizar sistemas *RFID* para identificar a pacientes, permitir el acceso por parte del personal relevante del hospital a los expedientes médicos. Además, que la *FDA*[7] aprobó recientemente los primeros chips *RFID* de Estados Unidos (Portal Industrial, 2010).

En Bolivia, según el director de la Agencia Nacional de Hidrocarburos (ANH), el B-Sisa[8] permite monitorear en línea la carga de combustibles en el país, mediante el sistema *RFID*, que también mantiene interconexión con datos de la Aduana Nacional de Bolivia (ANB), Servicio de Impuestos Nacionales (SIN), Vías Bolivia, el Registro Único para la Administración Tributaria Municipal, y el Servicio de Identificación Personal (SEGIP). El programa B-Sisa "se está convirtiendo en un sistema muy poderoso de información" para combatir la venta ilegal de combustibles en las 700 estaciones de servicio que funcionan en Bolivia (ABI, 2015).

1.3. Planteamiento del Problema

El Estado Plurinacional de Bolivia es un país que aún está emergiendo hacia una cultura del uso de tecnologías. Es en ese contexto que se encuentra el uso de dispositivos *RFID* como herramientas de trabajo. Si bien empresas nacionales están adoptando esta tecnología, la misma aún no ha cobrado la fuerza necesaria debido a la falta de mayor información acerca de la implementación que requiere en los diferentes rubros de la empresa boliviana.

---

[7] *FDA* Administración de Alimentos y Medicamentos.

[8] B-*SISA* Empresa Boliviana de Sistemas de Auto-identificación.

En empresas bolivianas que realizan el inventario de productos. Este proceso lo realizan manualmente, por lo cual, no cuentan con un sistema automatizado que permita registrar el material que ingresa y sale de la tienda o almacén, por lo tanto, la información que es revisada al finalizar del día no llega a ser oportuna ni precisa.

Otra necesidad que presentan las empresas del sector de importadoras, es el seguimiento que se realiza cuando los productos son trasladados de uno a otro almacén, generalmente este registro es manual e inexacto por lo que, si algún producto se pierde, no queda el registro del mismo ocasionando pérdidas monetarias para la empresa.

Finalmente, tampoco cuentan con un registro de la cantidad de los productos vendidos y saber si cuentan con más de estos productos en el almacén. Para esto, ayudará la aplicación móvil en la ubicación del stock de un producto.

1.3.1. Problema central

¿Qué bondades tecnológicas tiene una aplicación móvil en *iOS* capaz de registrar, contabilizar y ubicar los productos inventariados a través del uso de la tecnología *RFID* además de implementar el control y opción de búsqueda avanzada?

1.3.2. Problemas secundarios

- ¿Qué beneficios se obtiene con el desarrollo de una aplicación móvil en dispositivos *iOS* para el sector empresarial?
- ¿Qué es la tecnología *RFID* y como se vincula con la aplicación móvil?
- ¿En qué consiste el registro, la contabilización y ubicación de productos usando la tecnología *RFID*?

- ¿Cómo se controla el traslado de productos que contengan un *Tag RFID*[9] de una zona a otra mediante la aplicación?

- ¿Cómo implementar la opción de búsqueda de un *Tag RFID* determinado a través de la aplicación móvil?

1.4. Planteamiento de Objetivos

1.4.1. Objetivo General

Desarrollar una aplicación móvil en *iOS* capaz de registrar, contabilizar y ubicar los productos inventariados además de implementar el control y opción de búsqueda avanzada a través del uso de la tecnología *RFID*.

1.4.2. Objetivos Específicos

- Elaborar la aplicación móvil denominado *TRACKER* en ambiente *iOS* capaz de registrar, contabilizar y ubicar los productos inventariados de una sucursal, almacén o depósito.

- Implementar una configuración para el *TSL-1128* a través del *SDK* de *iOS* permitiendo personalizar la optimización de búsquedas de *Tags RFID* mediante comandos del lector.

- Demostrar los beneficios del uso de la aplicación móvil para el registro, contabilización y ubicación de los productos de un almacén usando la tecnología *RFID*

- Implementar opción de búsqueda de un *Tag RFID* determinado a través de la aplicación móvil.

[9] *Tag RFID* Etiqueta *RFID*, sistema para identificar con tecnología *RFID*.

1.5. Justificación

1.5.1. Económica

La realidad de las compañías en el manejo de grandes cantidades de productos se transcurre en la contratación de personal y la manipulación de los productos que en efecto llegan a perderse o ser mal registrados por procesos manuales y de esta manera ocasionar pérdidas económicas. La tecnología *RFID* ayudará a reducir costos rastreando y administrando de manera eficaz los ítems/productos.

1.5.2. Social

La gestión de inventarios, el almacenaje de productos y el control de acceso de vehículos y personas a predios ahora es más fácil con ayuda de *RFID*. En nuestro país Bolivia, se pretende que este proyecto beneficie a la sociedad y empresas que puedan instalar sistemas personalizados de radiofrecuencia. Obteniendo de esta manera mejorar el control y acceder información en tiempo real, por ejemplo, un conteo de todos los activos fijos y obtener su ubicación, además de que puede adecuarse a otros usos como el registro del ingreso y salida de personal, el control antirrobo en empresas y bibliotecas y la apertura de garajes.

1.5.3. Técnica

El desarrollo de aplicaciones móviles va cobrando mayor importancia en la actualidad debido a los costos accesibles de smartphones de media y alta gama que pueden reemplazar en muchos aspectos a los ordenadores de escritorio o portables. El ingreso de aplicaciones móviles al sector empresarial es indudablemente un espacio que muchas compañías de desarrollo de software encuentran una gran demanda.

1.6 Alcances

La propuesta está determinada al sector de la pequeña y mediana empresa en el rubro de importaciones de productos como ser: prendas de vestir, calzados, mochilas, etc. que a través de la aplicación móvil *iOS* y el uso de un lector de etiquetas *RFID* será capaz de mejorar el control de inventario.

1.7 Límites

La aplicación desarrollada para la presente investigación estará limitada de la siguiente manera:

- El desarrollo de la aplicación móvil, se ejecutará en dispositivos móviles que contengan *iOS,* con el uso del lector de *Tag RFID TSL-1128.*
- El usuario podrá escanear un almacén para controlar el inventario de un respectivo lugar, como también la búsqueda de un respectivo ítem/producto.
- El usuario podrá ver información de los ítems/productos mediante reportes y también el detalle de cada uno de ellos.

CAPITULO II
MARCO TEÓRICO

2.1. Inventarios

El inventario representa la existencia de bienes almacenados destinados a realizar una operación, sea de compra, alquiler, venta, uso o transformación. Debe aparecer, contablemente, dentro del activo como un activo circulante.

Los inventarios de una compañía están constituidos por sus materias primas, sus productos en proceso, los suministros que utiliza en sus operaciones y los productos terminados. Un inventario puede ser algo tan elemental como una botella de limpiador de vidrios empleada como parte del programa de mantenimiento de un edificio, o algo más complejo, como una combinación de materias primas y sub-ensamblajes que forman parte de un proceso de manufactura (Muller, 2005).

Conjunto de bienes corpóreos, tangibles y en existencia, propios y de disponibilidad inmediata para su consumo (materia prima), transformación (productos en procesos) y venta (mercancías y productos terminados) (Perdomo, 2004).

Se define un inventario como la acumulación de materiales (materias primas, productos en proceso, productos terminados o artículos en mantenimiento) que posteriormente serán usados para satisfacer una demanda futura (Moya, 1999).

El *stock* es el conjunto de productos almacenados en espera de su ulterior empleo, más o menos próximo, que permite surtir regularmente a quienes los consumen, sin imponerles las discontinuidades que lleva consigo la fabricación o los posibles retrasos en las entregas por parte de los proveedores (Ferrín, 2007).

2.2. Mantenimiento de Inventarios

En un ambiente manufacturero justo a tiempo, el inventario se considera un desperdicio. Sin embargo, si la organización tiene dificultades en su flujo de caja o carece de control sólido sobre a) la transferencia de información electrónica entre los departamentos y los proveedores importantes, b) los plazos de entrega y c) la calidad de los materiales que recibe, llevar inventario desempeña papeles importantes (Muller, 2005).

Entre las razones más importantes para constituir y mantener un inventario se cuentan:

- **Capacidad de predicción:** Con el fin de planear la capacidad y establecer un cronograma de producción, es necesario controlar cuánta materia prima, cuántas piezas y cuántos subensamblajes se procesan en un momento dado. El inventario debe mantener el equilibrio entre lo que se necesita y lo que se procesa.

- **Fluctuaciones en la demanda:** Una reserva de inventario a la mano supone protección; no siempre se sabe cuánto va a necesitarse en un momento dado, pero aun así debe satisfacerse a tiempo la demanda de los clientes o de la producción. Si puede verse cómo actúan los clientes en la cadena de suministro, las sorpresas en las fluctuaciones de la demanda se mantienen al mínimo.

- **Inestabilidad del suministro:** El inventario protege de la falta de confiabilidad de los proveedores o cuando escasea un artículo y es difícil asegurar una provisión constante.

- **Protección de precios:** La compra acertada de inventario en los momentos adecuados ayuda a evitar el impacto de la inflación de costos.

- **Descuentos por cantidad:** Con frecuencia se ofrecen descuentos cuando se compra en cantidades grandes en lugar de pequeñas.

- **Menores costos de pedido:** Si se compra una cantidad mayor de un artículo, pero con menor frecuencia, los costos de pedido son menores que si se compra en pequeñas cantidades una y otra vez (sin embargo, los costos de mantener un artículo por un periodo de tiempo mayor serán más altos).Con el fin de controlar los costos de pedido y asegurar precios favorables, muchas organizaciones expiden órdenes de compra globales acopladas con fechas periódicas de salida y recepción de las unidades de existencias pedidas.

2.3. Sistemas de contabilización de inventarios

2.3.1. Sistema de inventario periódico

Con este método la empresa no lleva un registro continuo de su *stock*, en cambio, realiza el conteo de existencias al final del periodo o ejercicio y los resultados se plasman en los informes financieros (González, Morini, & Do Nascimento, 2005).

2.3.2. Sistema de inventario permanente o perpetuo

Con este método la empresa mantiene un registro continuo de sus existencias y los costos de los productos o mercancías que ha vendido. Además, señala las siguientes ventajas de este método sobre el periódico (González, Morini, & Do Nascimento, 2005):

- Permite un mejor control de los artículos y la aplicación de técnicas de productos al poseer una información en tiempo real de los niveles de inventarios, rotaciones, evolución de precios, etc. Por tanto, mejora la toma de decisiones.
- Facilita el recuento físico en el caso de que esto sea necesario para llevar a cabo una verificación del inventario.
- Permite reducir costes y ofrecer un mejor servicio a los clientes, etc.

2.4. Tecnología *RFID*

La tecnología *RFID*, más comúnmente llamado Radio Frecuencia, es la forma que tiene de comunicarse los objetos modernos. Las diferentes utilidades de la tecnología *RFID* dan respuesta a una amplia gama de procesos empresariales (Dipole, 2017). En la figura 2.1 se puede observar como esta tecnología ha estado innovando con las etiquetas *RFID* en varias áreas del sector industrial.

Figura 2.1. Tecnología RFID
Fuente: (Dipole, 2017)

La tecnología RFID, que responde a las iniciales de Radio Frecuencia Identificación, no es más que un sistema para comunicarse sin cables entre dos o más objetos, donde uno emite señales de radio y el otro responde en función de la señal recibida.

La identificación por radio frecuencia es una de las tecnologías nuevas más prometedoras que se han orientado al sector del almacenamiento y distribución en muchos años. Pertenece a una amplia gama de tecnologías para Adquisición de Datos e Identificación automática (AIDC) en la que también se incluyen los códigos de barras, la lectura de caracteres ópticos y los sistemas infrarrojos de identificación.

Es un sistema de almacenamiento y recuperación de datos remotos que usa dispositivos denominados etiquetas, tarjetas, transponedores o *Tags RFID*. El propósito fundamental de la tecnología *RFID* es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio.

Las tecnologías *RFID* se agrupan dentro de las denominadas Auto *ID* (*automatic identification*, o identificación automática) (Santos, 2013).

Se trata de una tecnología basada en la utilización de un pequeño chip adherido a un producto, y a través del cual es posible mantener un rastreo de su localización. La distancia de rastreo varía mucho, dependiendo del tamaño, tipo y antena del chip, pero puede ser desde 2cm. a 13 metros en los sencillos, hasta incluso varios kilómetros en los más complejos.

2.4.1. Funcionamiento básico

Para que la tecnología *RFID* funcione, son necesarios tres elementos básicos: una etiqueta electrónica o *Tag*, un lector de *Tags* y una base de datos. Las etiquetas electrónicas llevan un microchip incorporado que almacena el código único identificativo del producto al que están adheridas. El lector envía una serie de ondas de radiofrecuencia al *Tag*, que éste capta a través de una pequeña antena. Estas ondas activan el microchip, que, mediante la micro-antena y la radiofrecuencia, transmite al lector cual es el código único del artículo (Santos, 2013).

2.4.2. Ventajas de este sistema.

La tecnología *RFID* supera muchas de las limitaciones del código de barras. A continuación, se mencionan las ventajas de las etiquetas electrónicas según (Santos, 2013):

- A diferencia del código de barras, las etiquetas electrónicas no necesitan contacto visual con el módulo lector para que éste pueda leerlas. La lectura se puede hacer a una distancia de hasta 10 metros.

- Mientras el código de barras identifica un tipo de producto, las etiquetas electrónicas identifican cada producto individual.

- La tecnología *RFID* permite leer múltiples etiquetas electrónicas simultáneamente. Los códigos de barras, por lo contrario, tienen que ser leídos secuencialmente.

- Las etiquetas electrónicas pueden almacenar mucha más información sobre un producto que el código de barras, que solo puede contener un código y, en algunos casos, un precio o cantidad.

- Mientras que sobre el código de barras se puede escribir solo una vez, sobre las etiquetas electrónicas se puede escribir todas las veces que haga falta.

- La tecnología *RFID* evita falsificaciones. Con una simple fotocopia se puede reproducir un código de barras. Las etiquetas electrónicas, en cambio, no se pueden copiar. Un *Tag* sobre un artículo de marca garantiza su autenticidad.

- Un código de barras se estropea o se rompe fácilmente, mientras que una etiqueta electrónica es más resistente porque, normalmente, forma parte del producto o se coloca bajo una superficie protectora y soporta mejor la humedad y la temperatura.

2.4.3. Aplicación

Son muchos los sectores industriales que pueden beneficiarse de las ventajas de la tecnología *RFID*. Algunas de sus aplicaciones son las siguientes según (Santos, 2013):

- Control de calidad, producción y distribución.
- Localización y seguimiento de objetos.
- Control de accesos.
- Identificación de materiales.
- Control de fechas de caducidad.
- Detección de falsificaciones.

- Almacenaje de datos.
- Automatización de los procesos de fabricación.
- Información al consumidor.
- Reducción de tiempo y coste de fabricación.
- Reducción de colas a la hora de pasar por caja.
- Identificación y localización de animales perdidos.
- Elaboración de censos de animales.
- Identificación y control de equipajes en los aeropuertos.
- Inventario automático.
- Entre muchas otras aplicaciones más.

2.4.4. Tecnología inalámbrica

La comunicación inalámbrica o sin cables es aquella en la que la comunicación (emisor/receptor) no se encuentra unida por un medio de propagación físico, sino que se utiliza la modulación de ondas electromagnéticas a través del espacio. En este sentido, los dispositivos físicos sólo están presentes en los emisores y receptores de la señal, entre los cuales encontramos: antenas, computadoras portátiles, teléfonos móviles, etc. (Urbina, 2011).

2.4.5. Identificación por Radiofrecuencia

Es una evolución del código de barras, que funciona desde hace 40 años, por medio de identificación por Radiofrecuencia remota e inalámbrica. Esta tecnología funciona a través de un lector, el cual está conectado a un sistema computarizado, permitiendo así la comunicación por medio de una antena con un *Tag* por medio de ondas de radio.

La lectura de los *Tag* es en segundos y sin necesidad de contacto (identificación a largas distancias), por medio de marcos de lecturas. Con los ajustes necesarios, se puede obtener una lectura del 100% de los elementos de estudio. Por ello, en la actualidad es bastante utilizado para la agilización del inventario de productos (Urbina, 2011).

2.4.6. Etiqueta o *Tag*

Es un dispositivo de pequeñas dimensiones que se adhiere a un producto, prenda o pertenencia de una persona. Contiene un conjunto de antenas que emiten y reciben información por radiofrecuencia desde un dispositivo de ancho de banda que emite y recibe señales. Cada *Tag* tiene una característica distinta, en su centro existe un chip de silicio del tamaño de un alfiler que almacena toda la información única del elemento al que se adjunta, como las características del producto (color, tipo, registro, fechas etc.) (Urbina, 2011).Está compuesto por una antena, chip, transductor de radio y se clasifican en:

- **Etiquetas Activas***:* Las etiquetas activas funcionan por medio de baterías, con lo que a todo momento envía señales por medio de esa energía. La batería tiene la energía suficiente y es buena para grandes distancias de lectura y escritura entre los lectores y etiquetas. Por lo general, la batería tiene una vida de 5 a 7 años. Se caracterizan por ser etiquetas más seguras, ya que pueden estar bajo el agua o estar en contacto con metales.
- **Etiquetas Pasivas**: Las etiquetas pasivas funcionan a través de un campo magnético que es enviado por el lector. Por esta razón, es una etiqueta que tiene un menor rango de alcance de comunicación frente a los otros tipos de etiquetas, sin embargo, es una de las más usadas.

2.5. Lectores *RFID*

Los lectores *RFID* se utilizan para rastrear inventarios y activos rápidamente y en forma precisa. Se comunican con etiquetas *RFID*, que envían y guardan datos, para la gestión de datos de último minuto.

El *hardware* existente para aplicaciones de lectura/escritura varía desde dispositivos Móviles llamados *Hand Held* o *Scanner* con comunicación vía cable o *WiFi*, equipo de escritorio con comunicación a puertos *USB*, lectores con antenas integradas y lectores de 4 puertos para el ensamble de portales.

Adicionalmente, existe otro tipo de complementos que permitirán adecuar módulos de lectura a teléfonos celulares, computadoras portátiles y otros dispositivos convencionales (Urbina, 2011).

2.5.1. Hand Helds o Scanners

Dispositivos móviles con sistemas operativos de *Windows Mobile* y *Windows CE* como se puede observar en la figura 2.2, de características diversas donde en la mayoría de sus casos son configurables para integrar módulos de *RFID* y código de barras, con aplicaciones de comunicación vía cable, redes *WiFi*, *Bluetooth*, *GPRS* y *GPS*.

Módulos adicionales para expansión de memoria y tarjetas de memoria externas y las funcionalidades de equipos y características permiten tener elecciones para aplicaciones sencillas o sumamente robustas para ambientes industriales. Las aplicaciones en campo que requieran movilidad, almacenes, inventarios y procesos de *picking* son ideales para este tipo de equipos (Solís, 2012).

Figura 2.2. Hand Helds
Fuente: (HTK-ID, 2017)

2.5.2. Lectores de escritorio

Estos equipos tienen configuraciones de *HF* y *UHF*, constan de lectoras y antenas integradas que permiten lecturas a corta distancia (promedio 10 cm) de cualquier identificador o *Transponder* que se coloque en la parte superior del equipo de escritorio (véase figura 2.3). Los métodos de comunicación convencionales son por medio de *USB* y opcionales a *TCP/IP*, este tipo de equipos son usuales para aplicaciones de lecturas de archivos, puntos de venta, transportadores y aplicaciones similares (HTK-ID, 2017).

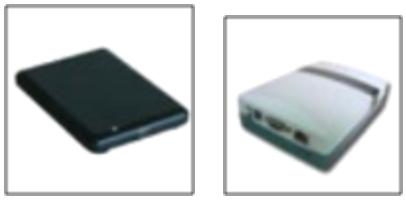


Figura 2.3. Lectores de escritorio
Fuente: (HTK-ID, 2017)

2.5.3. Lectores Integrados

Los lectores integrados son lectores que cuentan con una antena integrada en el mismo dispositivo, lo que facilita la instalación, conexiones, comunicación (véase figura 2.4).

Tienen capacidad de trabajar en interiores y exteriores, para aplicaciones convencionales o industriales. Ya que la antena está integrada al mismo lector, es ideal para aplicaciones de control de acceso, montaje sobre montacargas, aplicaciones móviles y similares. Para garantizar la lectura y funcionamiento, estos equipos cuentan con elección de polarización vertical o horizontal.

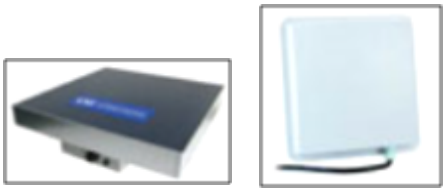


Figura 2.4. Lectores integrados
Fuente: (HTK-ID, 2017)

2.5.4. Lectores Fijos

Para el ensamble de portales, campos de lectura, identificadores continuos en bandas transportadoras, el dispositivo consta de una lectora con cuatro puertos de conexión para antenas, que pueden ser integradas con configuraciones de polarización vertical, horizontal o circular, métodos que garantizan la lectura masiva y a distancia en cada caso en especial.

Estos equipos cuentan con puertos *IO* que pueden interactuar con actuadores de entrada y salida como son sensores ópticos, contactos magnéticos, semáforos, contrachapas y otros similares como se puede observar en la figura 2.5.

Los métodos de comunicación de estos equipos son *RS232*, *RJ485*, serial, *USB* y *Ethernet TCP/IP*.

Figura 2.5. Lectores Fijos
Fuente: (HTK-ID, 2017)

2.6. Reader TSL - 1128

El lector Reader *TSL - 1128* es un dispositivo que tiene la característica de recuperación de datos remoto a través de etiquetas, tarjetas, transpondedores o *Tag RFID* (véase figura 2.6), este dispositivo además tiene la capacidad de utilizar un dispositivo móvil, ya sea en *Android, iOS y/o Windows Phone.*

Figura 2.6. Reader TSL 1128
Fuente: (TSL-UK, 2017)

2.6.1. Kit de desarrollo de software iOS (SDK) para lectores UHF-RFID

Mediante el SDK[10] de *iOS* para lectores de TSL se puede contener las siguientes características según (TSL-UK, 2017):

- TSLAsciiCommands.framework, un conjunto de clases *Objective-C* fáciles de usar que encapsulan el protocolo TSL UHF ASCII 2 proporcionado como un marco estático universal que se puede utilizar tanto con dispositivos iOS como con el simulador iOS.
- TSL ASCII 2 SDK Documentación - disponible como documento de configuración para la integración dentro de Xcode y en formato HTML.
- Proyectos de Xcode como demostración del uso del SDK.
- *Inventory*, demuestra la conexión con el 1128 y el uso del TSLInventoryCommand
- *ReadWrite*, demuestra el uso de los comandos TSLReadTransponder y TSLWriteSingleTransponder.
- *Trigger*, un ejemplo de cómo utilizar el disparador del dispositivo para iniciar operaciones en la aplicación mediante TSLSwitchActionCommand, TSLSwitchStateCommand y TSLSwitchResponder.
- *TSLTerm*, un programa de terminal simple que permite a los usuarios experimentar con el protocolo raw ASCII 2.

2.7. Patrones de diseño

Un patrón de diseño es la descripción de un problema que ocurre de manera reiterada en nuestro entorno, así como una solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin repetir lo mismo dos veces (Fresneda, 2016).

[10] *SDK*. Kit de desarrollo de software, conjunto de herramientas y/o programas de desarrollo que permite al programador crear aplicaciones para un determinado paquete de software.

Esta definición puede ser aplicada no solo al diseño de software si no a cualquier tipo de patrón en cualquier situación.

2.7.1. Componentes de un Patrón de Diseño

Un Patrón de Diseño (Fresneda, 2016) debe tener los siguientes elementos esenciales:

- **Nombre:** Es la parte que permite describir al patrón, normalmente se utilizan una o dos palabras para ello. Al utilizar nombres de este tipo, permite hablar de ellos, documentarlos. De esta manera, resulta más fácil pensar en estos diseños y poder transmitirlos a otros. Quizás esta sea la parte más difícil de la creación de un patrón de diseño.

- **Problema:** Debe describir cuando aplicar el patrón. Explica el problema y en el contexto en el que se haya. Puede ser la descripción de un problema en concreto de diseño, así como estructuras de clases u objetos que puedan ser síntoma de un diseño poco flexible. El problema puede incluir una serie de condiciones que deben darse para que el patrón tenga sentido.

- **Solución:** Describe los elementos que forman el diseño, las relaciones, responsabilidades y colaboraciones. No describe un diseño o una implementación concreta, sino que un patrón es como una plantilla que se puede aplicar en muchas diferentes situaciones. El patrón proporciona una descripción abstracta de problemas de diseño y como se resuelve una disposición general de elementos.

- **Consecuencias**: Son los resultados, tanto las ventajas como inconvenientes de aplicar el patrón. Las consecuencias son fundamentales para poder evaluar las posibles alternativas de diseño y comprender los costes y beneficios de aplicarlo. Ya que la reutilización suele ser uno de los factores del diseño orientado a objetos, las consecuencias que tiene un patrón incluyen su impacto sobre la flexibilidad, extensibilidad y portabilidad de este en un sistema.

2.7.2. Descripción de un patrón de diseño

Utilizar anotaciones gráficas, no son suficientes a la hora de describir un patrón, aunque sean importantes para su descripción. Para la reutilización de un diseño, es necesario hacer constar las decisiones, alternativas y ventajas e inconvenientes que dieron lugar al patrón. También es importante dar ejemplos concretos, que ayuden a ver el diseño aplicado a una situación real (Fresneda, 2016).

En la descripción de patrones existe un formato consistente. Cada patrón será dividido en secciones de acuerdo con la plantilla que explicaré a continuación. Esta estructura le dará uniformidad a la información, haciendo que el patrón sea más fácil de aprender y utilizar.

Plantilla:

- **Nombre del patrón y clasificación**: El nombre del patrón debe describir de manera clara el sentido del patrón. Su clasificación depende de su nivel de abstracción, relacionando estas clasificaciones asignadas como familias entre patrones.
- **Propósito**: Esta parte debe ser breve, y debe responder a qué función realiza el patrón, en qué se basa y cuál es el problema concreto que soluciona.
- **También conocido como…**: Informa sobre posibles variaciones en el nombre que permita el reconocimiento de este.
- **Motivación**: Es un escenario ilustrativo sobre un problema de diseño y como con el uso de estructuras de clases y objetos del patrón, se puede resolver el problema. Este escenario ayuda en la comprensión de la aplicabilidad.
- **Aplicabilidad**: Tiene que resolver las cuestiones de qué situaciones se puede utilizar el patrón, qué ejemplos de mal diseño puede resolver y como se pueden reconocer estas situaciones.
- **Estructura**: Es una representación gráfica de las clases del patrón utilizando una notación basada en el Modelado de Objetos (OMT). También mediante el uso de diagramas de interacción para mostrar posibles consecuencias de peticiones y colaboraciones entre los distintos objetos.

- **Participantes**: Clases y objetos participantes en el patrón, junto con sus responsabilidades en este.
- **Colaboraciones**: De qué manera colaboran los participantes para realizar sus responsabilidades.
- **Consecuencias**: Responde a de qué manera consigue completar sus objetivos el patrón, cuáles son las ventajas e inconvenientes y el resultado que ofrece el uso del patrón, también qué aspectos de la estructura del sistema está permitido modificar de manera independiente.
- **Implementación**: La dificultad, trucos o técnicas que se deben tener en cuenta cuando se aplique el patrón y si existen cuestiones específicas del lenguaje.
- **Código de ejemplo**: Fragmentos de código que muestre cómo se implementa el patrón.
- **Usos conocidos**: Posibles entornos reales en los que se puede utilizar el patrón.
- **Patrones relacionados**: La relación que tiene el patrón con otros patrones y cuáles son las diferencias con estos. También con qué otros patrones se debería de utilizar. También cabe la posibilidad de añadir un apéndice que de información adicional para ayudar a entender el patrón.

2.7.3. Patrones para el desarrollo

Los patrones que se han aplicado al diseño de software de la aplicación están basados en el libro Gang of Four (GoF) de Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Estos patrones que se describen en el libro son los más importantes y fundamentales en el desarrollo de software moderno. Además, también se aplica el conocido patrón arquitectónico MVC y el patrón *Object Template* ambos explicados en el libro *Pro Design Patterns in Swift* de Adam Freeman (Fresneda, 2016).

2.7.3.1. Patrón Objeto Plantilla (*Object Template*)

A continuación, en la tabla 2.1. según (Fresneda, 2016) se puede ver la siguiente información.

Tabla 2.1. Patrón Objeto Plantilla

Definición	Este patrón utiliza una clase o estructura como especificación del tipo de datos y la lógica que recibe este tipo de dato. Los objetos son creados utilizando una plantilla, los valores de estos datos son insertados durante su inicialización, ya sea mediante el uso de valores por defecto en la plantilla o proporcionados por el componente de la clase o inicializador de la estructura, también conocido como constructor.
Beneficios	Proporcionar la base para agrupar los valores de los datos y la lógica de quien los manipula en conjunto, este método también es conocido como encapsulación. La encapsulación permite a un objeto representar una *API*[11] para sus consumidores al tiempo que oculta la implementación privada de esa *API*. Esto ayuda a prevenir el acoplamiento de componentes.
Aplicación	En cualquier proyecto, aunque no es recomendable en los que sean muy simples, ya que añadirían una complejidad extra, que no es necesaria. Para este tipo de proyectos simples se pueden utilizar las tuplas de *Swift*, ya que son una característica interesante, pero cuando el proyecto tiene una envergadura mayor pueden presentar un problema de mantenimiento a largo plazo. Utilizar este patrón sólo se requiere un poco de trabajo extra para crear una clase muy simple o una estructura.

El concepto más básico de estructura que se puede utilizar para este caso es una tupla (véase figura 2.7), la cual permite almacenar datos de múltiples tipos en un solo lugar y poder ser accesibles en un solo bloque. Ante esto se puede encontrar con el problema que suele conllevar el desarrollo de software, se trata de ampliar funcionalidades (Fresneda, 2016).

```
9
10  var user = [
11      ("Sergio", "Fresneda", "05-11-1990", 41, "Hombre", "mail@mail.es",
            "profileImage.jpg", "headerImage.jpg")
12  ]
13
```

Figura 2.7. Tupla Usuario
Fuente: (Fresneda, 2016)

Se puede observar que la lectura del código ha mejorado, también mediante la encapsulación se ha reducido el número de modificaciones que tienen que realizarse en caso de ampliar la funcionalidad de este objeto en un futuro.

[11] *API.* La interfaz de programación de aplicaciones, abreviada como API del inglés: *Application Programming Interface*.

El objeto *User*, se puede observar en la figura 2.8 de manera sencilla en cuanto a definición, en todo momento se puede conocer el tipo de datos que utiliza y de qué manera se almacena, sin necesidad de ningún tipo de documentación.

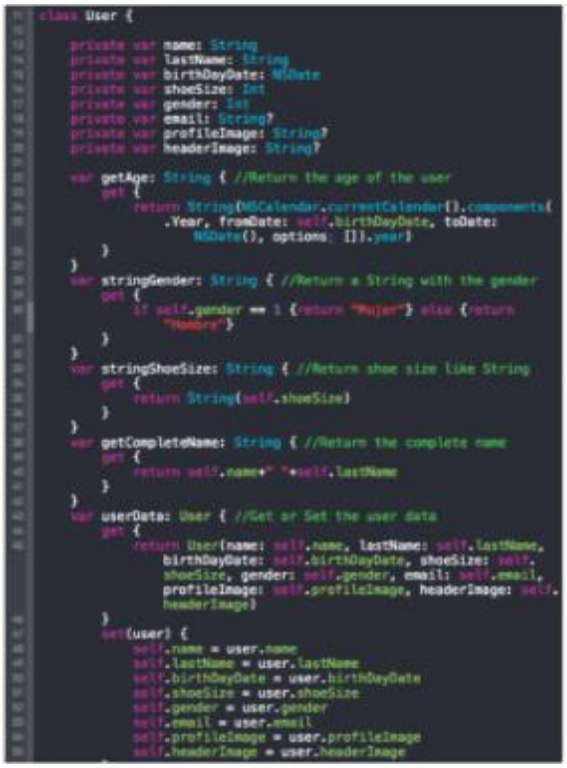


Figura 2.8. Patrón Objeto Plantilla
Fuente: (Fresneda, 2016)

2.7.3.2. Patrón Prototipo (*Prototype*)

En este patrón en cuanto a su estructura se puede observar en la figura 2.9, que la creación de nuevos objetos llega a ser parte de un objeto existente el cual es denominado prototipo.

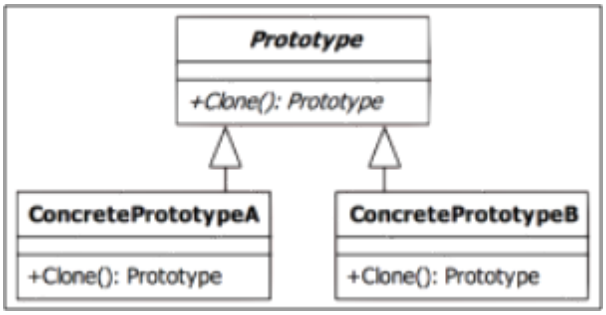


Figura 2.9. Diagrama patrón prototipo
Fuente: (Fresneda, 2016)

A continuación, en la tabla 2.2. se puede observar la siguiente información: Definición, beneficios y su respectiva aplicación.

Tabla 2.2. Patrón Prototipo

Definición	El patrón prototipo crea nuevos objetos copiando un objeto existente, conocido como prototipo.
Beneficios	El principal beneficio es poder ocultar el código que crea objetos a partir de los componentes que utilizan; esto significa que los componentes no necesitan saber qué clase o estructura se requiere para crear un nuevo objeto, no necesitan saber los detalles de inicializadores, y no tienen que cambiar cuando se crean y se instancian subclases. Este patrón también se puede utilizar para evitar la repetición de las inicializaciones cada vez que un nuevo objeto de un tipo específico que sea.
Aplicación	La aplicación de este patrón se la realiza cuando al cambiar el inicializador de la clase o estructura utilizada como objeto prototipo, no requiere un cambio correspondiente en el componente que crea clones. Otra forma de verificarlo, sería crear una subclase de la clase de prototipo y comprobar de nuevo de que el componente puede clonar sin requerir ningún cambio.

Para el ejemplo de este patrón se puede tomar como código previo el ejemplo de la figura 2.8 en el patrón Objeto plantilla con algunas modificaciones que se puede observar en la figura 2.10.

```
var anonimousUser = [(name:"", lastname:"", birthDayDate: NSDate(), shoeSize:0,
    gender: 0, email:"", profileImage: "", headerImage:"")]

print("Complete name: \(anonimousUser[0].name) \(anonimousUser[0].lastname)")
//"Complete name:  \n"

var registeredUser = [(name:"Sergio", lastname:"Fresneda", birthDayDate: NSDate(),
    shoeSize:41, gender: 0, email:"mail@mail.es", profileImage:
    "profileImage.jpg", headerImage:"headerImage.jpg")]

print("Complete name: \(registeredUser[0].name) \(registeredUser[0].lastname)")
//"Complete name: Sergio Fresneda\n"
```

Figura 2.10. Tupla usuario anónimo y registrado
Fuente: (Fresneda, 2016)

Como se puede observar, crear varios tipos de usuario se tiene que repetir la estructura base una y otra vez por cada usuario generado. De esta manera se ensucia mucho el código con repeticiones que pueden evitarse utilizando el patrón prototipo (Fresneda, 2016).

Con este patrón se puede continuar el código demostrativo del ejemplo anterior con el patrón objeto plantilla. A continuación, se puede observar su implementación para demostrar la utilidad de este patrón en la figura 2.11.

```
let anonimousUser : User = User(name: "", lastName: "", birthDayDate: NSDate(),
    shoeSize: 0, gender: 0, email: "", profileImage: "", headerImage: "")
print(anonimousUser.getCompleteName)
////"Complete name: \n"

let registeredUser : User = anonimousUser

registeredUser.name = "Sergio"
registeredUser.lastName = "Fresneda"
registeredUser.birthDayDate = NSDate()
registeredUser.shoeSize = 41
registeredUser.gender = 0
registeredUser.email = "mail@mail.es"
registeredUser.profileImage = "profileImage.jpg"
registeredUser.headerImage = "headerImage.jpg"
print("Complete name: \(registeredUser.getCompleteName)")
//"Complete name: Sergio Fresneda\n"
```

Figura 2.11. Patrón Prototipo
Fuente: (Fresneda, 2016)

Implementando un usuario anónimo o de cualquier otro tipo, se puede clonar este objeto y modificarlo para crear un usuario de otro tipo como se observa en la figura 2.11. No sólo permite el acceso a los atributos del objeto, sino también a sus funciones internas como se muestra con *getCompleteName*. Esto agiliza el desarrollo de objetos que tienen una base igual, pero pueden tener un rol distinto dependiendo de cómo estén formados.

2.7.3.3. Patrón Estrategia (*Strategy*)

Este patrón se utiliza para el desarrollo de clases que pueden ser extendidas a través de su aplicación de objetos que se ajustan a un protocolo definido, en la figura 2.12 se puede observar su respectivo diagrama a continuación.

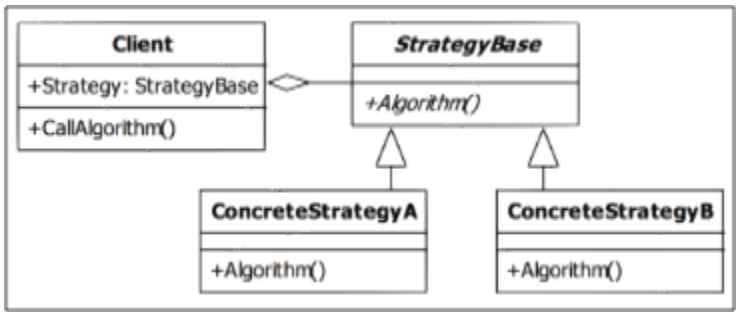


Figura 2.12. Diagrama patrón estrategia
Fuente: (Fresneda, 2016)

A continuación, en la tabla 2.3. se puede observar la siguiente información acerca de su definición, beneficios y su aplicación sobre el respectivo patrón según (Fresneda, 2016).

Tabla 2.3. Patrón Estrategia

Definición	Estrategia se utiliza para crear clases que se pueden extender sin modificación, mediante la aplicación de objetos que se ajustan a un protocolo bien definido.
Beneficios	Permite a los desarrolladores de terceros cambiar el comportamiento de las clases sin modificarlos directamente, y permiten cambios de bajo costo para hacerse en proyectos que tengan procedimientos de gran coste y longitud para clases específicas.
Aplicación	Se aplica cuando se pueda extender el comportamiento de una clase mediante su definición y aplicación de una nueva estrategia sin necesidad de realizar ningún cambio en la propia clase.

En la figura 2.13, se puede observar un ejemplo utilizando este patrón como en el ejemplo anterior.

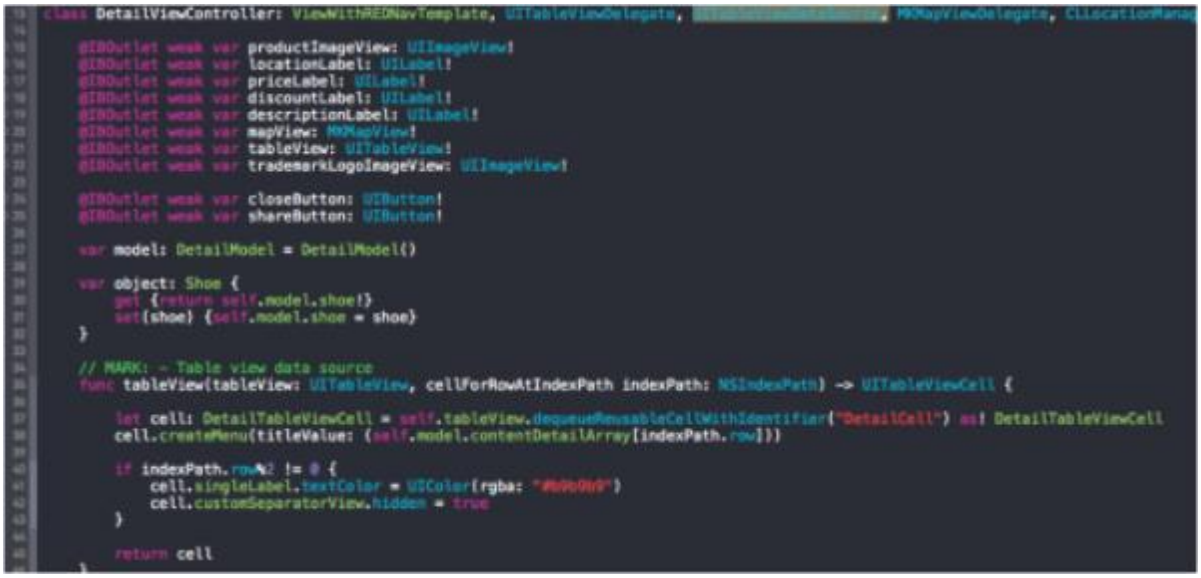


Figura 2.13. Patrón estrategia, en el controlador de la vista detalle
Fuente: (Fresneda, 2016)

2.7.3.4. Patrón Plantilla

Este patrón se puede complementar con el patrón estrategia ya mencionado anteriormente, se puede observar en la figura 2.14 que su estructura permite modificar un algoritmo para ser sustituido por las implementaciones proporcionadas por un tercero, se podrá utilizar la misma clase detalle de vista que en el patrón anterior para su demostración en cuanto a su aplicación.

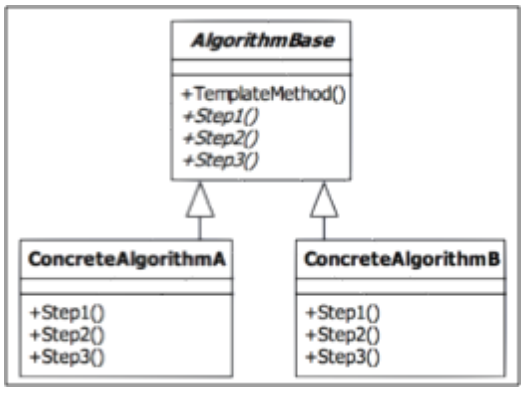


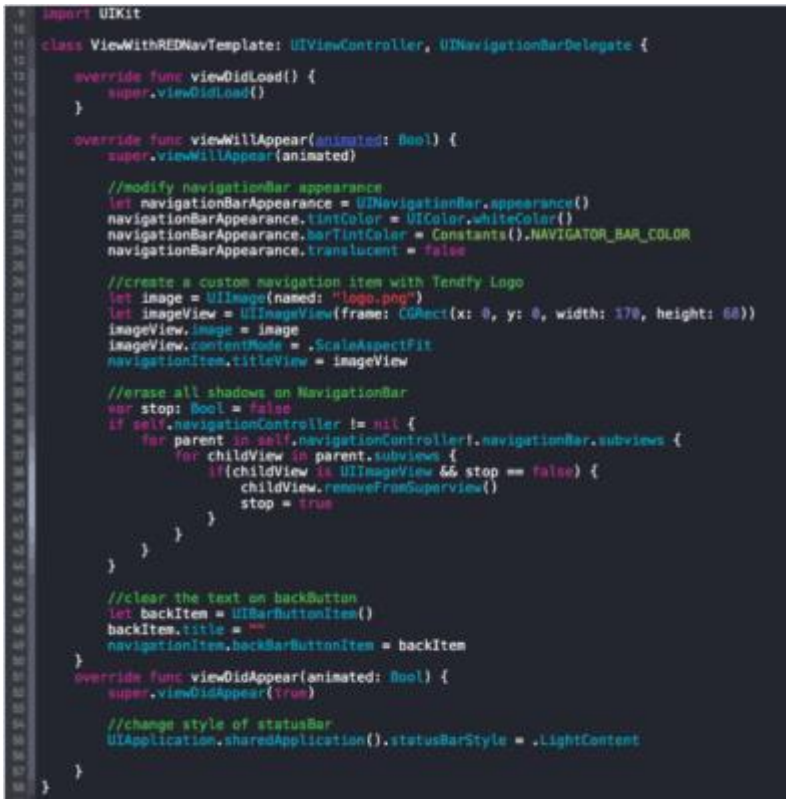
Figura 2.14. Diagrama del patrón plantilla
Fuente: (Fresneda, 2016)

En la tabla 2.4. se puede ver la siguiente información sobre su definición, beneficios y su aplicación sobre el patrón de desarrollo según (Fresneda, 2016).

Tabla 2.4. Patrón Plantilla

Definición	Este patrón permite modificar un algoritmo para ser sustituido por las implementaciones proporcionadas por un tercero, ya sea mediante la especificación de las funciones como clausuras o creando una subclase.
Beneficios	Permite que un *framework* se pueda extender y personalizar por otros desarrolladores que lo incorporen en su aplicación.
Aplicación	Se aplica en case de utilizar el contenido de un algoritmo sin modificar la clase original que lo contiene.

En la figura 2.15, se puede observar clase ViewWithREDNavTemplate de la cual se extenderán los dos controladores.



```
import UIKit

class ViewWithREDNavTemplate: UIViewController, UINavigationBarDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)

        //modify navigationBar appearance
        let navigationBarAppearance = UINavigationBar.appearance()
        navigationBarAppearance.tintColor = UIColor.whiteColor()
        navigationBarAppearance.barTintColor = Constants().NAVIGATOR_BAR_COLOR
        navigationBarAppearance.translucent = false

        //create a custom navigation item with Tendfy Logo
        let image = UIImage(named: "logo.png")
        let imageView = UIImageView(frame: CGRect(x: 0, y: 0, width: 170, height: 60))
        imageView.image = image
        imageView.contentMode = .ScaleAspectFit
        navigationItem.titleView = imageView

        //erase all shadows on NavigationBar
        var stop: Bool = false
        if self.navigationController != nil {
            for parent in self.navigationController!.navigationBar.subviews {
                for childView in parent.subviews {
                    if(childView is UIImageView && stop == false) {
                        childView.removeFromSuperview()
                        stop = true
                    }
                }
            }
        }

        //clear the text on backButton
        let backItem = UIBarButtonItem()
        backItem.title = ""
        navigationItem.backBarButtonItem = backItem
    }
    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(true)

        //change style of statusBar
        UIApplication.sharedApplication().statusBarStyle = .LightContent

    }
}
```

Figura 2.15. Patrón Plantilla
Fuente: (Fresneda, 2016)

Ahora se observa el resultado tras aplicar el patrón a las siguientes clases como se puede observar en las figuras 2.16. y 2.17.

```
class MainViewController: ViewWithREDNavTemplate {

    @IBOutlet var filterButton: UIBarButtonItem!
    @IBOutlet weak var kolodaView: CustomKolodaView!

    let sizeImagePulse: CGFloat = UIScreen.mainScreen().bounds.width/3
    let model: MainModel = MainModel()

    override func viewDidLoad() {
        super.viewDidLoad()

    }
```

Figura 2.16. Patrón Plantilla controlador vista principal
Fuente: (Fresneda, 2016)

```
class DetailViewController: ViewWithREDNavTemplate, UITableViewDelegate, MKMapViewDelegate, CLLocationManagerDelegate {

    @IBOutlet weak var productImageView: UIImageView!
    @IBOutlet weak var locationLabel: UILabel!
    @IBOutlet weak var priceLabel: UILabel!
    @IBOutlet weak var discountLabel: UILabel!
    @IBOutlet weak var descriptionLabel: UILabel!
    @IBOutlet weak var mapView: MKMapView!
    @IBOutlet weak var tableView: UITableView!
    @IBOutlet weak var trademarkLogoImageView: UIImageView!

    @IBOutlet weak var closeButton: UIButton!
    @IBOutlet weak var shareButton: UIButton!

    var model: DetailModel = DetailModel()

    var object: Shoe {
        get {return self.model.shoe!}
        set(shoe) {self.model.shoe = shoe}
    }

    override func viewDidLoad() {
        super.viewDidLoad()

    }
```

Figura 2.17. Patrón Plantilla controlador vista detalle
Fuente: (Fresneda, 2016)

Las diferencias que se puede observar a primera vista es la reducción de código en los controladores. Como segundo cambio aplicado, se puede observar que se cambió el tipo de controlador de UIViewController al tipo ViewWithRedNavTemplate. Con este cambio se permite a cada controlador reutilizar el mismo código, dejando de estar manera un código limpio y optimizado.

Si es necesario realizar cambios sobre el comportamiento de ambos controladores, se tiene que modificar la clase principal de la que derivan estas clases.

2.7.3.5. Patrón MVC

El patrón MVC es primordial en cuanto a la adición de una estructura a toda la aplicación en lugar de un componente individual, se puede observar en la figura 2.18 su respectiva estructura de este patrón.

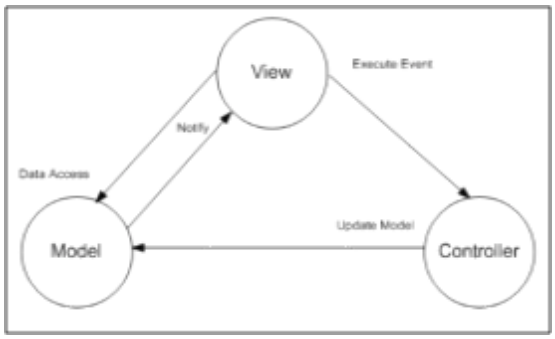


Figura 2.18. Diagrama Patrón MVC
Fuente: (Fresneda, 2016)

A continuación, bajo la siguiente tabla 2.5. se puede observar la siguiente información sobre su definición, beneficios y su aplicación según (Fresneda, 2016).

Tabla 2.5. Patrón MVC

Definición	El patrón MVC añade estructura a toda una aplicación en lugar de a un componente individual.
Beneficios	Cada parte de una aplicación puede ser desarrollada, testeada y mantenida con mayor facilidad.
Aplicación	Se aplica en caso de desarrollar un proyecto complejo. La implementación correcta de este patrón tiene que ver con las preferencias personales de cada desarrollador, no existe una explicación de cuando el patrón se implementa correctamente. En términos generales, las secciones individuales de la aplicación (modelo, vistas y controladores) deben estar muy poco acoplados con el fin de poder aislarlos fácilmente para realizar pruebas, también realizar cambios en una parte de la aplicación sin que sea necesario realizar los cambios en las otras secciones.

El uso del patrón permite que la lógica de negocio del controlador funcione de forma indistinta a los datos que se almacenan en el modelo. En caso de que el controlador necesite un dato para mostrarlo en la vista o realizar alguna tarea con los datos, se lo solicitará al modelo y recogerá el resultado. En las figuras 2.19 y 2.20 se puede observar el ejemplo de uso.

Figura 2.19. Patrón MVC clase modelo de ajustes
Fuente: (Fresneda, 2016)

Figura 2.20. Patrón MVC clase controlador de ajustes
Fuente: (Fresneda, 2016)

Como se puede observar, en el controlador se obtiene el modelo mediante el cual realiza la obtención y almacenamiento de datos, desacoplando así, parte del código.

Es el patrón de diseño más común en la aplicación, ya que la gran mayoría de las pantallas requieren de un modelo para almacenar información acerca de la vista.

La estructura de carpetas en este proyecto que sigue este patrón es la siguiente figura 2.21:

Figura 2.21. Estructura de Carpetas MVC
Fuente: (Fresneda, 2016)

2.8. Tecnologías para el desarrollo de la aplicación

En el proceso del desarrollo del prototipo se realiza variadas tecnologías para una aplicación en *iOS* la cual se dará la descripción a continuación:

2.8.1. iOS

iOS es un sistema operativo diseñado con el objetivo de abstraer el hardware y facilitar el desarrollo de aplicaciones para dispositivos de Apple, generalmente dispositivos móviles.

Inicialmente, *iOS* se denominaba *iPhoneOS*, presentado en 2007 junto al primer *Smartphone* más parecido a los que se tiene hoy en día (véase figura 2.22). El sistema no tenía soporte para aplicaciones externas al sistema operativo.

Disponía de una colección variada y útil de herramientas. Un año después tras el lanzamiento de *iPhoneOS* 2.0, se lanzó la primera versión de *XCode* con soporte para *iPhoneOS* y con ello la posibilidad de desarrollar aplicaciones para el sistema operativo y poder venderlas en la *AppStore* (Fresneda, 2016).

Figura 2.22. Evolución de *iOS* desde 2007 – 2016
Fuente: (Fresneda, 2016)

2.8.1.1. Capas de iOS

Como cualquier sistema operativo en la actualidad, el sistema se encuentra dividido en capas, cada una de ellas encargadas de una serie de tareas. Todas ellas son dependientes una de la otra, aunque estén definidamente separadas (véase figura 2.23). Según (Fresneda, 2016) se puede ver las siguientes características:

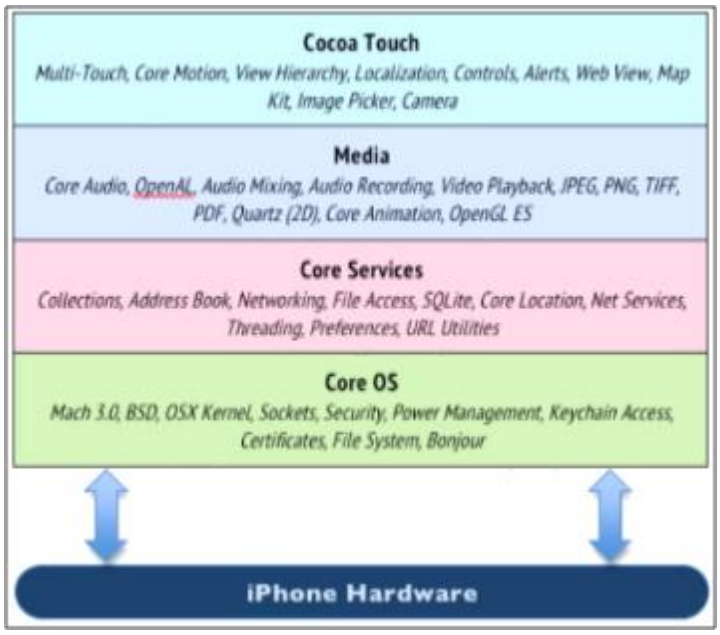


Figura 2.23. Capas del sistema operativo
Fuente: (Fresneda, 2016)

- ***Core OS***: Esta es la capa más a bajo nivel del sistema. Contiene las características más básicas como: manejo de ficheros del sistema, manejo de memoria, seguridad, drivers del dispositivo.

- ***Core Services***: Contiene servicios fundamentales del sistema que usan todas las aplicaciones para el almacenamiento de datos.

- ***Media***: Provee los servicios de gráficos y multimedia, como puede ser el renderizado de imágenes, reproducción de música.

- ***Cocoa Touch***: Esta es la capa más importante para el desarrollo de aplicaciones iOS. Contiene un conjunto de *Frameworks* que proporciona *Cocoa* para desarrollar aplicaciones. Esta capa está formada por dos *Frameworks* fundamentales:

    - ✓ ***UIKit***: contiene las clases que necesita el desarrollador para el diseño de la interfaz de usuario.

    - ✓ ***Foundation Framework***: define las clases básicas, acceso y manejo de objetos, servicios del sistema operativo.

2.8.1.2. Kernel iOS

iOS está basado en el *kernel* Unix, este adopta el nombre de Darwin. Funciona con el mismo *kernel* que el sistema operativo de los dispositivos de mesa con *macOS* dotando de gran fiabilidad. iOS aprovecha el *kernel* para la mejora su seguridad, la gestión de memoria con ARC, introducido junto a la presentación de la versión 5 del sistema, hasta entonces la gestión de la memoria era trabajo del programador (Fresneda, 2016).

iOS utiliza el modelo de controladores, proporcionando soporte a nuevos accesorios a través de estos controladores. Separando el hardware del software mediante una capa de abstracción.

2.8.1.3. Frameworks de iOS

Hay un conjunto de *frameworks* básicos desde las cuales se pueden implementar nuevos *frameworks* o simplemente utilizar los existentes para el desarrollo de aplicaciones. Pasamos a explicar algunos de ellos:

- ***UIKit***: Este *framework* se encarga de la generación de objetos de interfaz de la aplicación, desde vistas, hasta simples etiquetas o botones.
- ***Foundation***: Mediante *Foundation* se puede realizar el manejo de objetos o servicios del sistema operativo. El uso de este *framework* es completamente necesario para el desarrollo de cualquier tipo de aplicación, ya tenga esta una parte de interfaz o no.
- ***Core Data***: Con Core Data se puede manejar la persistencia de la aplicación mediante una base de datos del tipo *SQLite*.
- ***Core Location***: Gracias a este *framework* disponemos de acceso a los servicios de geolocalización del dispositivo, pudiendo manejarlos para las necesidades que tengamos en el desarrollo de una aplicación.

2.8.1.4. Entorno de las aplicaciones

Este entorno proporciona una plataforma de desarrollo permitiendo la reutilización de componentes, cada aplicación muestra las capacidades que tiene y se utilizan unas a otras en base a ellas mismas (véase figura 2.24).

Con el acceso a estos *frameworks*, el desarrollador puede tener acceso al hardware de los dispositivos, para poder gestionar el ciclo de vida de la aplicación (Fresneda, 2016).

Figura 2.24. Elementos gráficos de *UIKit*
Fuente: (Fresneda, 2016)

2.8.1.5. Nivel de aplicación

Mediante esta capa se encuentran las aplicaciones preinstaladas por Apple y las que instalará el usuario. Las aplicaciones se desarrollan en *Objective-C / C / C++ /Swift* utilizando el *SDK* (*Software Development Kit*) de Apple, llamado *XCode*.

En cuanto a aplicaciones por defecto se tiene *App Store*, que ofrece al resto de *Apps* desarrolladas por terceros.

2.8.1.6. Ciclo de vida

A la hora de desarrollar aplicaciones para cualquier sistema operativo, es necesario el ciclo de vida de las aplicaciones en el sistema para el que vayamos a desarrollarlas. El ciclo de vida se puede dividir en dos grupos principales:

- **Estado Activo**: Este estado es en el cual la aplicación aparece en pantalla de forma completa o parcialmente completa. Durante todo este proceso la aplicación puede acceder o realizar una serie de tareas dentro de las limitaciones del sistema.

- **Estado Background**: Siempre y cuando una aplicación pierda el foco, ya sea debido a que se cambia de aplicación, se cierra o se suspende debido al bloqueo del dispositivo, este pasa a modo *background*, el cual tiene unas limitaciones del sistema mayores y más restrictivas.

Debido a que el sistema es tan cerrado, las aplicaciones en estado *background* tienen dos opciones; finalizar el proceso que están ejecutando o piden un tiempo de prórroga para acabar lo que estaban haciendo y detenerse (Fresneda, 2016).

Es muy común en aplicaciones que descargan contenidos dentro de la aplicación, si se suspende la aplicación mientras se descarga una canción, la *App* debe pedir una prórroga al sistema para terminar de descargar la canción y luego detenerse. En caso de no ser suficiente la prórroga, el sistema la detendrá automáticamente (véase figura 2.25).

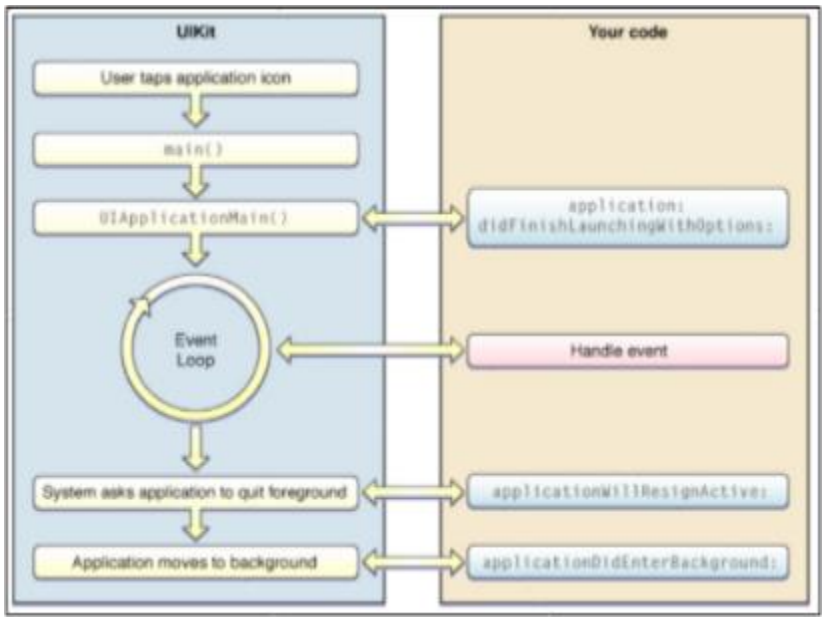


Figura 2.25. Ciclo de vida de una aplicación
Fuente: (Fresneda, 2016)

2.8.1.7. Procesos y su gestión

Es común pensar en la multitarea como un conjunto de procesos ejecutándose al mismo tiempo y al devolver el foco a uno de estos continuar con lo que estaba haciendo.

En *iOS* la multitarea es existente y a la vez no lo es. El proceso de gestión de procesos del sistema es distinto al que se puede encontrar en Android, en esa plataforma se pueden tener varias aplicaciones funcionando realmente, refrescando contenidos y realizando sus tareas sin un sistema que intenta que dejen de hacer su trabajo, lo que conocemos como una multitarea real. Sin embargo, en iOS funciona de una manera distinta (Fresneda, 2016).

Cuando un proceso pasa a *background*, como se ha explicado anteriormente, este tiene un tiempo determinado para acabar su trabajo y almacenarse en memoria, si transcurre ese tiempo y no se almacena, se desecha. El proceso al almacenarse en memoria se detiene completamente, no puede ser capaz ni de modificar la interfaz.

Cuando el proceso requiere volver a la actividad, este recoge de memoria y continúa por el mismo instante por donde se quedó suspendido. La ventaja que se obtiene es un mayor ahorro de recursos del sistema, ya no solo en procesador si no en el consumo de la batería. (Fresneda, 2016)

2.8.2. XCode

XCode es el *IDE* oficial para el desarrollo de aplicaciones en *iOS/macOS/tvOS/watchOS*. Este es el único entorno de desarrollo de aplicaciones en los lenguajes nativos del sistema. Aunque se puede desarrollar en otras plataformas como *Xamarin*, escogimos el IDE por excelencia de Apple. *Xcode* es un software propio de Apple, inicialmente orientado al desarrollo de aplicaciones para *macOS*, pero actualmente es el nexo del desarrollo para los distintos operativos de Apple.

2.8.2.1. Estructura de Datos

Por defecto, *Xcode* muestra los archivos del proyecto en la barra lateral izquierda. Esta vista proporciona un acceso rápido a los archivos fuente y muestra la estructura de nuestro proyecto (véase figura 2.26).

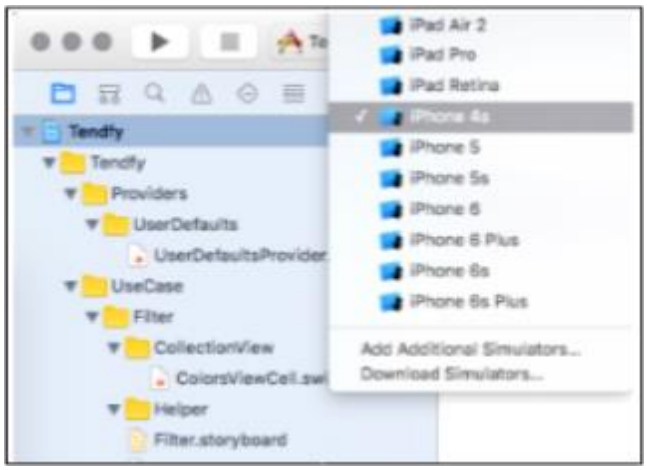


Figura 2.26. Vista del proyecto
Fuente: (Fresneda, 2016)

Esta vista muestra con un icono azul el fichero principal del proyecto, desde el que nace el resto del contenido de la aplicación, el cual se puede estructurar según lo desee el desarrollador. Se puede encontrar varios tipos de ficheros dentro de esta vista, se indica a los principales:

- ***.swift***: son ficheros de código fuente que utiliza el lenguaje *Swift*. En estos ficheros es donde se implementa el código que ejecutará la aplicación según se interactúe con él.

- ***.storyboard*****:** este tipo de fichero es con el que se puede realizar la interfaz de la aplicación mediante una serie de herramientas visuales que permiten generar la vista de la *App*, de manera muy sencilla y cómoda sin tener que recurrir a código.

- ***.xcassets*****:** se le puede considerar un directorio especial en lugar de un tipo de fichero. Con los *xcassets* almacenamos como su propio nombre indica los *Assets* de la aplicación, es decir todos los gráficos de tipo imagen que serán mostrados en la aplicación.

- ***.plist*****:** en cuanto a estos ficheros suelen almacenar opciones de configuración de la aplicación, como puede ser la versión del sistema que tiene como *target* la *App*.

2.8.2.2. *iOS Builder*

Como se puede apreciar en la parte derecha de la figura 2.27, se tiene una pequeña herramienta que permite la compilación, ejecución y test de la aplicación. El sistema también proporciona la opción de elegir sobre que dispositivo que se quiere ejecutar la aplicación (Fresneda, 2016).

2.8.2.3. Depuración y rendimiento

XCode proporciona un gran número de mejoras para la depuración y ayuda a mejorar el rendimiento de nuestro código, mediante un asistente para la emulación de dispositivos móviles, depuración línea a línea y herramientas de análisis del rendimiento. *XCode* también permite la activación de *BreakPoints* en el código en tiempo de ejecución (véase figura 2.27), es decir, se puede añadir puntos críticos en nuestro código mientras se está ejecutando la aplicación en el simulador, permitiéndonos tener mayor control sobre nuestra aplicación mientras se está desarrollando (Fresneda, 2016).

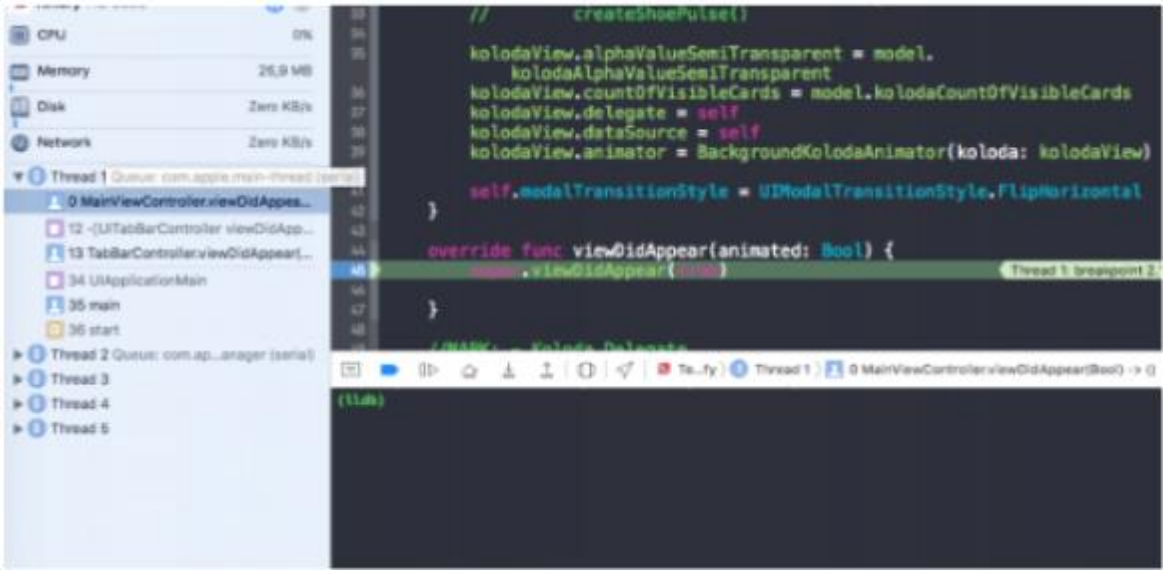


Figura 2.27. *BreakPoint* de *XCode*
Fuente: (Fresneda, 2016)

2.8.2.4. *iOS Simulator*

El simulador pone a nuestro servicio una máquina virtual que simula un dispositivo. Este simulador permite ejecutar cualquier dispositivo con *iOS* para el target al que va dirigido nuestra aplicación, incluyendo las *tablets*. El simulador te permite ejecutar cualquier función del sistema a excepción de notificaciones *push* y la cámara de fotos. En caso de necesitar estas dos funciones es necesario un dispositivo real.

También es posible la simulación de una geolocalización, incluso movimiento en vivo por un mapamundi, además existe la posibilidad de simular el movimiento del dispositivo, el uso de la huella dactilar o la tecnología *3D Touch* (Fresneda, 2016).

2.8.2.5. Monitorizar del rendimiento de una App

Xcode también provee de un apartado de monitorización de rendimiento de la aplicación, este se ejecuta siempre que lanzamos la aplicación desde el *IDE* (véase figura 2.28). Mientras la aplicación se ejecuta en el simulador o en el dispositivo real con el que se testee la aplicación, se puede ver las siguientes características mediante una serie de gráficos (Fresneda, 2016):

- ***CPU***: Indica el consumo de procesador que tiene la aplicación durante la ejecución. Esto puede dar pistas de qué partes de la aplicación tienen más carga y poder asegurarlas ante posibles problemas de calentamiento del dispositivo cuando se publique la aplicación.

- ***Memory***: Como su nombre indica muestra el consumo de memoria *RAM* del dispositivo, esto puede ser un factor muy a tener en cuenta cuando se desarrolla una *App*, ya que un consumo tanto de *CPU* como de memoria indica un mayor consumo de batería y posibles salidas inesperadas de la aplicación debido a que el sistema cierra una aplicación cuando considera que consume más memoria de la que debería.

- ***Disk***: Se puede ver el uso de escrituras que hacemos el disco del dispositivo. Esto suele ser muy útil cuando utilizamos *CoreData* y el acceso a los datos de la base de datos es continuo.

- ***Network***: Otro factor a tener muy en cuenta cuando se desarrolla una aplicación es el uso que hacemos de la red, ya que un uso excesivo puede acabar con la tarifa de datos del usuario. Por este motivo se puede optimizar los contenidos descargados con la aplicación que se está desarrollando.

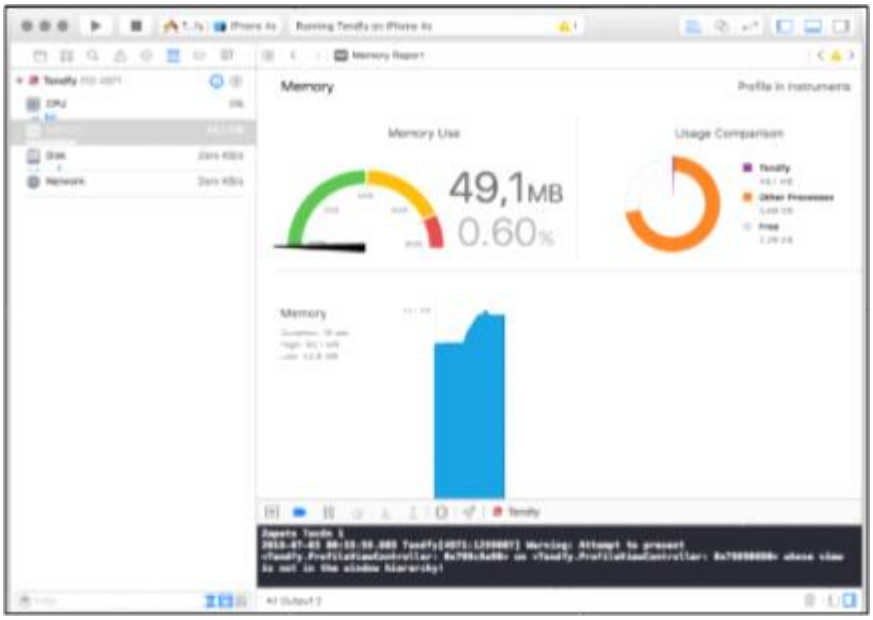


Figura 2.28. Monitor de Rendimiento
Fuente: (Fresneda, 2016)

2.8.3. *Bitbucket*

Bitbucket es una plataforma de alojamiento para trabajo colaborativo de software, utilizando el sistema de control de versiones, permite que varios desarrolladores trabajen sobre el mismo proyecto. El código se almacena de forma privada o pública, dependiendo del fin que le quiera dar el equipo de desarrollo. El uso de esta plataforma es gratuito para la mayoría de los proyectos (Fresneda, 2016).

Bitbucket aloja código y da herramientas como *sourceTree* para agilizar el uso del repositorio en equipo. También puede contribuir a mejorar los proyectos de otros usuarios que han publicado de manera pública. *Bitbucket* ofrece funcionalidades para hacer *fork* y solicitar *pulls*.

Para realizar un *fork* se tiene que clonar el repositorio con el que se quiere colaborar (genera una copia en la cuenta), se puede solucionar algún *bug* añadiendo funcionalidades al código. Una vez realizado los cambios se puede lanzar un *pull* al dueño del proyecto original. Éste valora los cambios que se han realizado, y puede considerar la contribución, adjuntándolo al repositorio original.

También se tiene la posibilidad de crear *branches* (ramas), con las que se puede alojar diferentes versiones de nuestro código sobre el mismo repositorio, facilitando así tanto el desarrollo como el mantenimiento y el *testing* de cada módulo por separado.

2.9. Aplicaciones Móviles

En los últimos años se incrementó el uso de aplicaciones móviles, las cuales fueron innovando el campo de la Tecnología Móvil ya que consiste en la utilización de medios informáticos, sin la necesidad de estar emplazados, es decir que estos pueden ser utilizados desde cualquier parte.

Una aplicación móvil es un programa que ha sido desarrollado para dispositivos móviles, cuya función es la de realizar alguna tarea específica. Su instalación y acceso se da a través de dispositivos móviles. A continuación, se puede citar algunas razones:

- Según un informe de KPCB (Kleiner, Perkins, Caufield & Byers) del año pasado en 2015 habrá nada menos de 1.600 millones de *smartphones*[12] en el mundo. En cuanto a dispositivos móviles se calcula una suma de 5.600 millones. Que es una gran audiencia. Además, según un reciente estudio de la GSMA (Asociación GSM) para 2020 la cifra de *smartphones* puede ser de 6.000 millones (KPCB, 2015).

[12] *Smartphone* Teléfono móvil inteligente.

- Según previsiones de *Statista*, los ingresos generados en las diferentes *AppStores*[13] están previstos en 45.400 millones de dólares para 2015. Y esa cifra sería de 76.520 millones para 2017. Esto va unido a las previsiones de descargas de apps, estimadas en 167.050 millones para 2015 y en 211.310 millones para 2016 (Statista, 2015).

- Según un estudio de BI Intelligence. También aumentará la inversión en publicidad móvil seguirá aumentado hasta alcanzan los 42.000 millones en 2018. Según Medialets los anuncios mostrados dentro de las aplicaciones son más eficaces que los mostrados en las webs móviles, un *click-through* de 0.56 de los *ads* in-apps frente a un CTR 0.23 de los de webs móviles (BI Intelligence, 2015).

2.9.1. Metodologías de desarrollo para aplicaciones móviles.

Hoy en día, el sector de los dispositivos móviles ha evolucionado de manera muy creciente en pocos años. Esto ha permitido crear un nuevo mercado, bastante amplio para todos los públicos y un abanico enorme de posibles soluciones eficaces que pueden hacer la vida más cómoda. Muy pocas personas son los que, a día de hoy, no tienen un dispositivo móvil, ya sea un *Smartphone* o una *Tablet*.

El desarrollo de aplicaciones móviles no se aleja mucho con respecto a los desarrollos de cualquier tipo de software normal, ya que se encuentra los mismos problemas a la hora de realizarlo. Algunos factores en los que sí varía mucho son en el hardware donde serán implantados dichos desarrollos, ya que están en constante evolución y el usuario suele cambiar de dispositivo en poco tiempo, a diferencia con los ordenadores personales que aguantan mucho más, a pesar de que el coste es menor.

[13] *AppStore Marketplace* de aplicaciones para usuarios de Apple, a través del cual, los desarrolladores ofrecen sus apps y juegos.

El desarrollo de aplicaciones móviles sufre prácticamente los mismos problemas que la gran mayoría de desarrollos de software. Aunque hay que tener en cuenta sus principales peculiaridades como la corta duración de sus desarrollos, la gran competencia del sector que obliga a una constante innovación, los cambios frecuentes en la plataforma de desarrollo y en el hardware o la simplicidad de algunas aplicaciones. Todo ello influye a la hora de elegir una metodología concreta de desarrollo (Rodríguez, 2015).

La mayor parte de los proyectos de desarrollo de software se lleva a cabo por equipos de desarrolladores que requieren de un método de desarrollo común para organizar sus tareas, ya sean de una forma ágil o de un modelo más estático y predictivo. A continuación, se puede ver una serie de metodologías especialmente indicada para el desarrollo de aplicaciones móviles:

2.9.2. Modelo *Waterfall* (en cascada)

Es clásico, sólo aplicable cuando están totalmente cerrados los requisitos y no van a cambiar. No hay retroalimentación entre las fases en que se divide el proyecto. Por lo que cada fase se va cerrando de forma secuencial. Todo el proceso está fijado por fechas límites y presupuestos. Este modelo sólo es aconsejable para proyectos móviles muy controlados y previsibles, no existe incertidumbre por lo que se quiere hacer ni influyen los cambios en la industria (Rodríguez, 2015).

2.9.3. Desarrollo rápido de aplicaciones

Se da énfasis a la obtención de un prototipo funcional de una aplicación para posteriormente ir mejorándolo incluyendo más funcionalidades y complejidad. Es recomendable el uso de patrones de diseño bien conocidos para adaptarse a los cambios de requisitos.

Se suele usar cuando los plazos de entrega son muy cortos y se precisa tener un entregable de forma inmediata. No se descarta utilizar otras metodologías de forma posterior, ya que este tipo de desarrollo puede ser usado para mostrar un esbozo de la aplicación a un cliente, generalmente en un par de días (Rodríguez, 2015).

2.9.4. Desarrollo ágil

Las metodologías ágiles son muy adecuadas para el desarrollo de aplicaciones móviles:

- En primer lugar, la alta volatilidad del entorno hace que constantemente el equipo de desarrollo se deba adaptar a nuevos terminales, cambios en la plataforma o en el entorno de desarrollo. Un ritmo cambiante que requiere una alta respuesta al cambio más que al seguimiento de un plan concreto.

- En la actualidad, los equipos de desarrollo móvil suelen se integrados por pocas personas. No más de ocho o diez desarrolladores entorno a una misma aplicación o, incluso, un único desarrollador. Las interacciones en el proceso y las herramientas son más controlables y es posible una fluida comunicación entre los miembros del equipo.

- Una aplicación se suele realizar en periodos de desarrollo cortos entorno a un mes a seis meses. Con el propósito de una realimentación rápida es posibles realizar varias actualizaciones de una aplicación según se van entregando funcionalidades. Un tema demandado por los usuarios en los distintos *markets* (sucursales) de aplicaciones que aprecian la frecuente mejora de la aplicación para mantenerla viva, e incluso ayuda a su propio marketing al tener más visibilidad cada vez que se realiza la actualización (Rodríguez, 2015).

2.9.5. *Mobile - D*

Esta metodología está basada en diversas metodologías de desarrollo como *Rational Unified Process*[14], *Extreme Programming*[15] y *Crystal Methodologies*[16], y su finalidad es intentar obtener pequeños ciclos de desarrollo de forma rápida en dispositivos pequeños. Un ciclo de proyecto con la metodología Mobile-D está compuesto por cinco fases:

2.9.5.1. Fase de Exploración

Esta fase es la encargada de la planificación y educción de requisitos del proyecto, donde tendremos la visión completa del alcance del proyecto y también todas las funcionalidades del producto.

2.9.5.2. Fase de inicialización

La fase de inicialización es la implicada en conseguir el éxito en las próximas fases del proyecto, donde se preparará y verificará todo el desarrollo y todos los recursos que se necesitarán. Esta fase se divide en cuatro etapas: la puesta en marcha del proyecto, la planificación inicial, el día de prueba y día de salida.

---

[14] *Rational Unified Process* Proceso *Rational* Unificado o *RUP* es un proceso de desarrollo de software desarrollado por la empresa *Rational Software*.

[15] *Extreme Programming* Programación Extrema XP como proceso de creación de software diferente al convencional nació de la mano de Kent Beck.

[16] *Crystal Methodologies* Metodologias Crystal, familia de metodologías ágiles para el desarrollo de software, fue desarrollado por *Alistair Cockburn.*

2.9.5.3. Fase de producción

En la fase de producción, se vuelve a repetir la programación de los tres días, iterativamente hasta montar (implementar) las funcionalidades que se desean. Aquí se usa el desarrollo dirigido por pruebas (*TDD*[17]), para verificar el correcto funcionamiento de los desarrollos.

2.9.5.4. Fase de estabilización

Se llevarán a cabo las últimas acciones de integración donde se verificará el completo funcionamiento del sistema en conjunto. De toda la metodología, esta es la fase más importante de todas ya que es la que asegura la estabilización del desarrollo. También se puede incluir en esta fase, toda la producción de documentación.

2.9.5.5. Fase de pruebas

Es la fase encargada del testeo de la aplicación una vez terminada. Se deben realizar todas las pruebas necesarias para tener una versión estable y final. En esta fase, si se encuentra con algún tipo de error, se debe proceder a su arreglo, pero nunca se han de realizar desarrollos nuevos de última hora, ya que haría romper todo el ciclo.

A continuación, en la figura 2.29, se puede ver las fases del ciclo del proyecto y sus etapas correspondientes:

[17] *TDD Test-driven development* Estilo de programación que incluye tres actividades: codificación, pruebas y diseño.

Figura 2.29. Fases de la metodología Mobile - D

2.9.6. Pruebas de calidad en Aplicaciones Móviles

En un entorno en el que el uso de dispositivos móviles toma posiciones frente a *PCs*[18] o televisiones, las empresas ven en *smartphones* y tabletas un canal clave en el que ganar presencia y a través del cual ofrecer sus servicios o productos. Son numerosos los estudios y referencias que reflejan un cambio en el comportamiento, en la forma de relacionarse e, incluso, en el proceso de toma de decisiones del consumidor que, cada vez con mayor frecuencia, consulta información a través de su dispositivo móvil.

Dependiendo de los presupuestos y estrategias particulares, y con el objetivo de acceder al nuevo consumidor, hay organizaciones que optan por optimizar sus sitios web para cada dispositivo, mientras que otras se deciden por el desarrollo de aplicaciones a medida. Ambas soluciones son aptas para conseguir una conversión en ventas a través de estos nuevos canales. Ahora bien, en este entorno, los usuarios son especialmente exigentes. Esperan obtener la información que necesitan de forma precisa, clara y, sobre todo, rápida. La paciencia no es precisamente una cualidad del nuevo consumidor, y en el momento en que un fallo o caída de la aplicación les

[18] *PC* Computadora de escritorio.

retrasa o impide alcanzar su objetivo, no dudan en acudir a otra de la competencia sin ninguna intención de ofrecer segundas oportunidades (MTP, 2015).

2.9.6.1. Garantizar la calidad de las aplicaciones móviles

Desde el inicio de su construcción o desarrollo es necesario probar las aplicaciones móviles teniendo en cuenta, precisamente, su propio contexto de movilidad y considerando el cumplimiento de las características de funcionalidad, usabilidad, seguridad y rendimiento de las mismas. La complejidad del aseguramiento de la calidad del software en aplicaciones móviles radica en la existencia de diferentes sistemas operativos y multitud de modelos de dispositivos. De esta forma, una aplicación debidamente construida y de calidad será aquella que funcione correctamente en todos los dispositivos móviles y bajo todos los sistemas operativos existentes en el mercado (MTP, 2015).

Las pruebas funcionales deberían llevarse a cabo considerando las características propias de estos dispositivos. Así, resulta diferente el uso que se hace y la presentación de la información en un *smartphone* respecto a una *tablet* o un ordenador. Además, la aplicación debe ser capaz de guardar, tanto de forma efectiva como selectiva, la información, dejando en el servidor aquellos datos que precisen de recursos adicionales. Finalmente, es necesario observar el comportamiento y fallos de la aplicación cuando pueda verse afectada por una situación de batería baja, entrada de llamadas o mensajes mientras realizamos una operación con la misma (MTP, 2015).

2.9.6.2. Usabilidad

Hablando de la usabilidad de una aplicación, es necesario poner especial atención en el desarrollo de pruebas que permitan asegurar que el diseño e interfaz resultan sencillos e intuitivos, así como que el flujo de comunicación entre la *App* y el usuario se establece de forma natural y lógica (MTP, 2015).

2.9.6.3. Seguridad

El comportamiento de una *App* móvil también preocupa al usuario desde el punto de vista de la seguridad, ya que están disponibles en dispositivos que pueden ser susceptibles de robo o de uso fraudulento. En este sentido, resulta imprescindible asegurar, entre otros factores, que la *App* mantenga la confidencialidad de los datos privados facilitados, la verificación de que el usuario es quien dice ser, los diferentes perfiles que puede tener un usuario dentro de la misma aplicación o el autoguardado de un histórico de operaciones o actividades realizadas (MTP, 2015).

2.9.6.4. Pruebas de Rendimiento

En cuanto a las pruebas de rendimiento, tienen como objetivo ayudar a la empresa a prevenir errores que eviten el abandono de la aplicación debido a caídas o a un tiempo excesivo en su ejecución. Un alto consumo de batería, la velocidad a la que se desarrolla dependiendo del navegador utilizado o las características de cada sistema operativo, pueden ser factores que influyen de forma decisiva en este apartado.

Finalmente, señalar que la conexión a Internet es también una consideración a tener en cuenta en lo que a pruebas de software se refiere. Esta conexión permite realizar búsquedas de información desde cada dispositivo, en cualquier momento y desde cualquier lugar. En este sentido, es necesario prestar atención a los caracteres especiales de escritura de los diferentes idiomas, las monedas de cada país, los formatos de códigos postales, fechas, direcciones, números de teléfono, entre otros, que deberían visualizarse dependiendo de la zona geográfica en la que se encontrara el usuario.

Dedicar especial atención a las pruebas de software desde las primeras fases de su desarrollo ayudará a las empresas a posicionar sus aplicaciones con una valoración de cinco estrellas, que otorga la máxima calidad en los *market/stores* de aplicaciones móviles. Y en el caso de las web optimizada y aplicaciones web, se garantizará su calidad, evitando el abandono de los usuarios (MTP, 2015).

CAPITULO III
MARCO APLICATIVO

El desarrollo de la aplicación se enmarcó en la metodología Mobile-D, una metodología orientada exclusivamente al desarrollo de aplicaciones móviles, en este capítulo se describe el desarrollo del prototipo en cada una de sus fases.

3.1. Exploración

En esta fase se describen los usuarios involucrados, los requerimientos, las historias de usuario, tareas y la planeación del proyecto.

3.1.1. Establecimiento de los usuarios

Los usuarios involucrados en este proyecto son descritos a continuación en la tabla 3.1.

Tabla 3.1. Descripción de los usuarios

Usuario	**Descripción**
Usuario de la aplicación móvil.	Persona que tiene a su cargo la aplicación con el lector de etiquetas *RFID*.
Usuario de la aplicación web	Persona que accede a la página web a través de un link para ver el control de los ítems, zonas y encargados.
Administrador	Persona administradora del sitio web de información del registro, validación de los ítems, zonas y encargados. Este usuario podrá ingresar datos de la configuración del lector y posteriormente ser consumido a través de la aplicación.

3.1.2. Requerimientos iniciales del producto

Los requerimientos para la construcción de la aplicación son los siguientes:

- Desarrollar una aplicación móvil para *iOS* con integración del lector *TSL-1128*, para el control de inventarios basados en la tecnología por radio frecuencia.

- Implementación del SDK TSL-1128 para la integración del lector con la aplicación móvil.
- Diseño de una base de datos para el registro de ítems, usuarios, zonas y configuración del lector.
- Construcción de una página web para el registro de ítems, zonas, encargados y configuración del lector.
- Desarrollo del módulo de control y conexión a través de *bluetooth* entre el dispositivo y el lector.
- Desarrollo de módulo *Scan*, para el escaneo de etiquetas a través de reportes.
- Desarrollo del módulo de asociación de nuevos ítems a través de la aplicación.
- Integración de mapas para la visualización de ítems.
- Implementación del módulo *Find It* para encontrar un determinado ítem.

3.2. Requerimientos del producto

En la tabla 3.2. se puede ver los requerimientos para la construcción de la aplicación que fueron los siguientes:

Tabla 3.2. Requerimientos del producto

Módulo	**Requerimiento**
Aplicación Móvil para el control de inventarios basados en la tecnología por Radiofrecuencia	Implementación y configuración del proyecto. Integración de las librerías para los módulos de la aplicación Implementación del *SDK TSL 1128* para el escaneo de *tags* y configuración del lector de *RFID*. Sincronización de nuevos *tags* y control de ítems a través de reportes.

3.3. Historias de usuarios y tareas

En las siguientes tablas se describen las historias de usuario con sus respectivas tareas de ingeniería que se efectuaron.

- ✓ En la siguiente tabla 3.3. se describe la historia de usuario 1 Creación del repositorio y configuración del proyecto.

Tabla 3.3. Historia de Usuario 1

Historia de Usuario	Creación y configuración del proyecto	
Número	**Tipos**	**Notas**
1	Desarrollo Mobile	
Descripción	Se debe crear el repositorio en *Bitbucket* y la configuración de *Cocopados* para las librerías que se usaran en el proyecto como la integración del SDK del lector.	
Prioridad	Alta.	
Esfuerzo	**Estimado**	10
	Real	8
Dificultad	Moderada.	

- ✓ En la siguiente tabla 3.4. se describe la tarea 1 Creación del repositorio en Bitbucket.

Tabla 3.4. Tarea 1.1 Creación del repositorio en *Bitbucket*.

Tarea	Creación del repositorio en *Bitbucket*.	
Número	**Tipo**	**Notas**
1.1	Desarrollo Mobile	
Dificultad	**Antes**	Medio
	Después	Medio
Descripción		
Realizar la creación del repositorio en *Bitbucket* para la configuración y control de versiones del proyecto.		

- ✓ En la siguiente tabla 3.5. se describe la tarea 2 Configuración del proyecto e integración con *Cocoapods* y *TSL SDK*.

Tabla 3.5. Tarea 1.2 Configuración del proyecto con *Cocoapods* y *SDK TSL-1128*

Tarea	Configuración del proyecto con *Cocoapods* y *SDK TSL-1128*.	
Número	**Tipo**	**Notas**
1.2	Desarrollo Mobile	
Dificultad	**Antes**	Medio
	Después	Medio
Descripción		
Realizar la configuración del proyecto habilitando el *podfile config* para la adición de librerías de *Cocoapods* e integración con el *SDK* del lector.		

- ✓ En la siguiente tabla 3.6. se describe la historia de usuario 2 Implementación del módulo de Inventario y Adición de Ítem.

Tabla 3.6. Historia de Usuario 2

Historia de Usuario	Implementación del módulo de Inventario y Adición de Ítem	
Número	**Tipos**	**Notas**
2	Desarrollo Mobile	
Descripción	Implementar el módulo de *Inventory* el cual se podrá obtener reportes para el control y escaneo de *tags*, como la adición de nuevos ítems los cuales se les añadirá su respectiva descripción.	
Prioridad	Alta.	
Esfuerzo	**Estimado**	30
	Real	35
Dificultad	Alta.	

- ✓ En la siguiente tabla 3.7. se describe la tarea 1 de la historia de usuario 2 Implementación del módulo de Inventario.

Tabla 3.7. Tarea 2.1 Implementación del módulo de Inventario

Tarea	Implementación del módulo de Inventario.	
Número	**Tipo**	**Notas**
2.1	Desarrollo Mobile	
Dificultad	**Antes**	Alta
	Después	Alta
Descripción		
Implementar la parte del módulo de Inventario para la obtención de los reportes y el escaneo de los ítems y configuración del modo de escaneo para el lector de RFID.		

- ✓ En la siguiente tabla 3.8 se describe la tarea 2 de la historia de usuario 2 Implementación del módulo de Adición de nuevos ítems.

Tabla 3.8. Tarea 2.2 Implementación del módulo de Adición de nuevos ítems

Tarea	Implementación del módulo de Adición de nuevos ítems.	
Número	**Tipo**	**Notas**
2.2	Desarrollo Mobile	
Dificultad	**Antes**	Alta
	Después	Alta
Descripción		
Implementar la parte del módulo de Adición de nuevos ítems el cual consiste en el registro de ítems los cuales se podrán agregar a los reportes agregando los respectivos detalles de la aplicación.		

- ✓ En la siguiente tabla 3.9. se describe la tarea 3 de la historia de usuario 2 Implementación del Escaneo de *Tags*.

Tabla 3.9. Tarea 2.3 Implementación del Controlador - Escaneo de *Tags*

Tarea	Implementación del Controlador - Escaneo de *Tags*	
Número	**Tipo**	**Notas**
2.3	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Alta
Descripción		
Implementar Controlador de Escaneo de *Tags* el cual se pueda ver a través de una vista y verificar su respectivo control, esta vista es de suma importancia ya que para que el usuario pueda verificar el control de escaneo de los ítems que pertenecen al reporte y aquellos que se encontraron.		

- ✓ En la siguiente tabla 3.10. se describe la tarea 4 de la historia de usuario 2 Configuración del Reader para el control de Inventarios.

Tabla 3.10. Tarea 2.4 Configuración del Reader para el control de Inventarios

Tarea	Configuración del Reader para el control de Inventarios.	
Número	**Tipo**	**Notas**
2.4	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Alta
Descripción		
En este módulo es importante realizar la configuración a través de los comandos para el Lector *TSL RFID-1128* con cual es podrá controlar el escaneo de *Tags* en modo *Single and Multiple Scan Mode*, agregando la posibilidad de integrar el modo *Barcode*		

- ✓ En la siguiente tabla 3.11. se describe la historia de usuario 3 Implementación del Módulo de Mapas.

Tabla 3.11. Historia de Usuario 3

Historia de Usuario	Implementación del Módulo de Mapas	
Número	**Tipos**	**Notas**
3	Desarrollo Mobile	
Descripción	Implementación de Mapas para mostrar la ubicación de los ítems registrados.	
Prioridad	Medio.	
Esfuerzo	**Estimado**	20
	Real	20
Dificultad	Moderada.	

- ✓ En la siguiente tabla 3.12. se describe la tarea 1 de la historia de usuario 3 Implementación de *MapKit* y *CoreLocation*.

Tabla 3.12. Tarea 3.1 Implementación de *MapKit* y *CoreLocation*

Tarea	Implementación de *MapKit* y *CoreLocation*	
Número	**Tipo**	**Notas**
3.1	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Media
Descripción		
Se debe agregar el módulo de *MapKit* para la implementación de mapas para que el usuario pueda ver la ubicación de los ítems y su referencia. Agregar *CoreLocation* para la obtener la ubicación del usuario.		

- ✓ En la siguiente tabla 3.13. se describe la tarea 2 de la historia de usuario 3 Configuración y creación de la vista de mapas para los ítems registrados.

Tabla 3.13. Tarea 3.2. Configuración y creación de la vista de mapas para los ítems registrados

Tarea	Configuración y creación de la vista de mapas para los ítems registrados	
Número	**Tipo**	**Notas**
3.2	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Media
Descripción		
En la vista y controlador de mapas se debe agregar los *markers* de los ítems para que el usuario pueda observar la geolocalización tanto de su ubicación como de los ítems registrados.		

- ✓ En la siguiente tabla 3.14. se describe la historia de usuario 4 Implementación del módulo *Find Ítem*.

Tabla 3.14. Historia de Usuario 4

Historia de Usuario	Implementación del módulo Find Ítem.	
Número	**Tipos**	**Notas**
4	Desarrollo Mobile	
Descripción	Find Ítem, es el módulo para buscar un determinado ítem de un reporte o a través de un lugar donde el usuario pueda estar ubicado.	
Prioridad	Alta.	
Esfuerzo	**Estimado**	20
	Real	30
Dificultad	Alta.	

- ✓ En la siguiente tabla 3.15. se describe la tarea 1 de la historia de usuario 4 Creación del controlador para los módulos *Scan Item Code* y *Find From List*.

Tabla 3.15. Tarea 4.1. Creación del controlador para los módulos *Scan Item Code* y *Find From List*

Tarea	Creación del controlador para los módulos *Scan Item Code* y *Find From List*	
Número	**Tipo**	**Notas**
4.1	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Media
Descripción		
Agregar un controlador para los módulos *Scan Item Code* y *Find From List* los cuales pertenecen a al módulo de *Find It*		

- ✓ En la siguiente tabla 3.16. se describe la tarea 2 de la historia de usuario 4 Implementar la navegación del módulo *Scan Item Code*

Tabla 3.16. Tarea 4.2. Implementar la navegación del módulo *Scan Item Code*

Tarea	Implementar la navegación del módulo *Scan Item Code*	
Número	**Tipo**	**Notas**
4.2	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Media
Descripción		
Scan Item Code es el módulo el cual permitirá la búsqueda de un *tag* en específico a partir de la ubicación del usuario previo a un escaneo del lugar usando el lector de RFID.		

- ✓ En la siguiente tabla se describe la tarea 3.17. de la historia de usuario 4 Implementar la navegación del módulo *Scan Item Code.*

Tabla 3.17. Tarea 4.3. Implementar la navegación del módulo *Find From List*

Tarea	Implementar la navegación del módulo *Find From List*	
Número	**Tipo**	**Notas**
4.3	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Media
Descripción		
Find From List es el módulo el cual permitirá la búsqueda de un *tag* en específico a partir de la ubicación del usuario previo a la selección de un reporte, y posteriormente el uso del lector de RFID.		

- ✓ En la siguiente tabla 3.18. se describe la tarea 4 de la historia de usuario 4 Creación del Controlador *Find It.*

Tabla 3.18. Tarea 4.4. Creación del Controlador *Find It*

Tarea	Creación del Controlador Find It	
Número	**Tipo**	**Notas**
4.4	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Alta
Descripción		
Find it es el controlador que se debe añadir el cual se podrá obtener la configuración respectiva del lector para obtener la búsqueda de un respectivo tag con el cual el usuario desee encontrar en una zona ubicada.		

- ✓ En la siguiente tabla 3.19. se describe la historia de usuario 5 Implementación del módulo de Configuración del Reader.

Tabla 3.19. Historia de Usuario 5

Historia de Usuario	Implementación del módulo de Configuración del Reader	
Número	**Tipos**	**Notas**
5	Desarrollo Mobile	
Descripción	El módulo de Configuración del Reader.	
Prioridad	Alta.	
Esfuerzo	**Estimado**	20
	Real	20
Dificultad	Alta.	

- ✓ En la siguiente tabla 3.20. se describe la tarea 1 de la historia de usuario 5 Creación del Controlador del módulo *Settings.*

Tabla 3.20. Tarea 5.1. Creación del Controlador del módulo *Settings*

Tarea	Creación del Controlador del módulo *Settings*	
Número	**Tipo**	**Notas**
5.1	Desarrollo Mobile	
Dificultad	**Antes**	Alta
	Después	Media
Descripción		
El módulo *Settings*, contendrá la conexión y control del lector como así configuración de sonido, vibración, poder de escaneo en modo *Single* y *Multiple*, y reinicio de controladores del lector.		

- ✓ En la siguiente tabla 3.21. se describe la tarea 2 de la historia de usuario 5 Implementar la navegación del módulo *Settings*.

Tabla 3.21. Tarea 5.2. Implementar la navegación del módulo *Settings*

Tarea	Implementar la navegación del módulo *Settings*	
Número	**Tipo**	**Notas**
5.2	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Media
Descripción		
Añadir la navegación de *Settings* dentro de la aplicación para tener las secciones de Información de usuario, Configuración del Lector y About.		

- ✓ En la siguiente tabla 3.22. se describe la historia de usuario 6 Ver información de ítems, encargados y configuración del lector de RFID.

Tabla 3.22. Historia de Usuario 6

Historia de Usuario	Ver información de ítems, encargados y configuración del lector de RFID.	
Número	**Tipos**	**Notas**
6	Desarrollo Back End	
Descripción	El usuario administrador podrá ver la información de ítems, zonas, encargados y la configuración del lector.	
Prioridad	Moderada.	
Esfuerzo	**Estimado**	20
	Real	25
Dificultad	Alta.	

- ✓ En la siguiente tabla 3.23. se describe la tarea 1 de la historia de usuario 6 Creación del servidor en *Node js* y base de datos *MongoDB.*

Tabla 3.23. Tarea 6.1. Creación del servidor en *Node js* y base de datos *MongoDB*

Tarea	Creación del servidor en *Node js* y base de datos *MongoDB*	
Número	**Tipo**	**Notas**
6.1	Desarrollo Back End	
Dificultad	**Antes**	Media
	Después	Media
Descripción		
Implementar la configuración del servidor en *Node JS* y *Mongo DB.* Habilitación de la base de datos y creación de tablas en *Mongo DB.*		

- ✓ En la siguiente tabla 3.24. se describe la tarea 2 de la historia de usuario 6 Configuración e implementación del *API* para consumir datos del servidor.

Tabla 3.24. Tarea 6.2. Configuración e implementación del *API* para consumir datos del servidor

Tarea	Configuración e implementación del *API* para consumir datos del servidor	
Número	**Tipo**	**Notas**
6.2	Desarrollo Mobile	
Dificultad	**Antes**	Media
	Después	Media
Descripción		
Configuración e implementación del *API* para consumir datos del servidor para las tablas *attendant, ítem, zone* y *readerConfiguration.*		

3.3.1. Planteamiento de las Iteraciones

En la tabla 3.25 se describen las fechas de las iteraciones y las tareas asignadas también se definen la siguiente relación con los requerimientos definidos en el punto.

Tabla 3.25. Tarea 6.2. Cronograma de las Iteraciones

Iteración	Nro.	Tarea	Inicio	Fin	Nro. Requerimiento o Asociado
Primera	1.1	Creación del repositorio en *Bitbucket*.	03/04/17	07/04/17	1
	1.2	Configuración del proyecto con *Cocoapods* y *SDK TSL-1128*.	03/04/17	07/04/17	2
Segunda	2.1	Implementación del módulo de Inventario.	10/04/17	12/04/17	3 y 4
	2.2	Implementación del módulo de Adición de nuevos ítems.	13/04/17	14/04/17	3
	2.3	Implementación del Controlador - Escaneo de *Tags*	17/04/17	18/04/17	3
	2.4	Configuración del Reader para el control de Inventarios.	19/04/17	21/04/17	2 y 3
Tercera	3.1	Implementación de *MapKit* y *CoreLocation*	24/04/17	28/04/17	1 y 2
	3.2	Configuración y creación de la vista de mapas para los ítems registrados	24/04/17	28/04/17	1 y 2
Cuarta	4.1	Creación del controlador para los módulos *Scan Item Code* y *Find From List*	01/05/17	03/05/17	2
	4.2	Implementar la navegación del módulo *Scan Item Code*	03/05/17	05/05/17	2
	4.3	Implementar la navegación del módulo *Find From List*	08/05/17	10/05/17	2 y 4
	4.4	Creación del Controlador *Find It*	10/05/17	12/05/17	2
Quinta	5.1	Creación del Controlador del módulo *Settings*	15/05/17	17/05/17	2 y 3
	5.2	Implementar la navegación del módulo *Settings*	17/05/17	19/05/17	2 y 3
Sexta	6.1	Creación del servidor en *Node js* y base de datos *MongoDB*	22/05/17	24/05/17	4
	6.2	Configuración e implementación del *API* para consumir datos del servidor	24/05/17	26/05/17	4

3.3.2. Iteración 1

En la primera iteración se completó la historia de usuario 1, se registró el proyecto en *Bitbucket* y la configuración del proyecto. Los recursos que se utilizaron fueron los siguientes:

- Xcode 8.3.3
- Xcode Command Line Tools
- Git 2.10.1
- Cocoapods 1.0.1
- SDK TSL-1128

3.3.2.1. Repositorio *Bitbucket*

Se creó el repositorio en *Bitbucket* en la cual se habilitaron dos ramas, la rama principal denominada *master* es la rama principal donde se obtiene el código de la aplicación que estará en producción o la versión estable de la aplicación como se puede observar en la figura 3.2. y la segunda rama es denominada *development* en la cual se obtiene el código de la aplicación el cual se lo utilizará para la implementación y desarrollo como se puede observar en la figura 3.2. A continuación, en la figura 3.1 se puede observar el repositorio del proyecto implementado en Bitbucket.

Figura 3.1. Repositorio Tracker Application

En la figura 3.2 se puede observar las ramas del repositorio en las cuales se ha desarrollado y almacenado el código de la aplicación móvil, de esta manera se logró manejar el control y versionamiento del código.

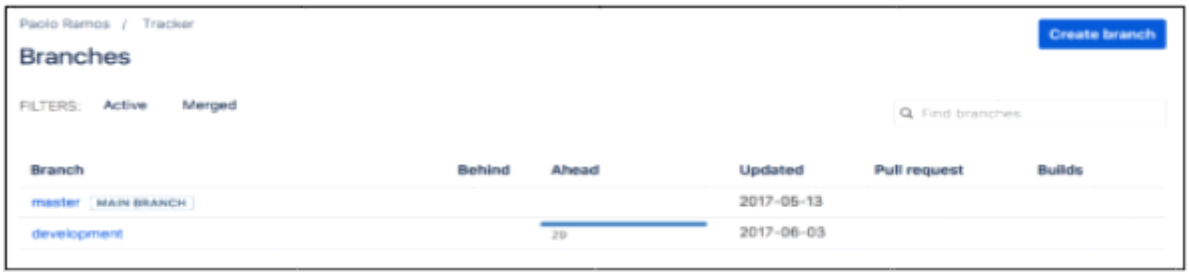


Figura 3.2. Implementación de los *Branches*

3.3.2.2. Configuración del Archivo *PodFile* para la integración de las librerías

Se realizó la integración dentro de la aplicación las siguientes librerías mediante el archivo de configuración *PodFile* (véase figura 3.3).

```
platform :ios, '9.0'

target 'Tracker' do
    use_frameworks!
    pod 'SwiftyJSON'
    pod 'SwiftLocation'
    pod 'Alamofire', '~> 4.4'
    pod 'IQKeyboardManagerSwift'
    pod 'Popover'
    pod 'MRProgress'
    pod 'KDCircularProgress'

    post_install do |installer|
        installer.pods_project.targets.each do |target|
            target.build_configurations.each do |config|
                config.build_settings['SWIFT_VERSION'] = '3.0'
            end
        end
    end

end
```

Figura 3.3. Configuración del archivo PodFile

3.3.3. Iteración 2

3.3.3.1. Módulo de Inventario.

Dentro del módulo de Inventario se implementó la navegabilidad para la selección de los reportes, el modo de escaneo con el lector y el escaneo de etiquetas *RFID* (véase figura 3.4).

Figura 3.4. Módulo de Inventarios.

3.3.3.2. Módulo de Adición de Ítems

Dentro del módulo de Adición de Ítems se implementó la navegabilidad para y el escaneo de etiquetas, esta navegabilidad es similar al Módulo de Inventarios en lo cual a través de este se realizó la adición de nuevos ítems, como se puede observar en la figura 3.4. ya mencionada.

3.3.3.3. *Counter View Controller*

CounterViewController, es una de las partes importantes de la aplicación en la cual se puede observar en tiempo real el escaneo de *tags* para que el usuario pueda obtener el control de ítems que fueron identificados, como también los nuevos (véase figura 3.5).

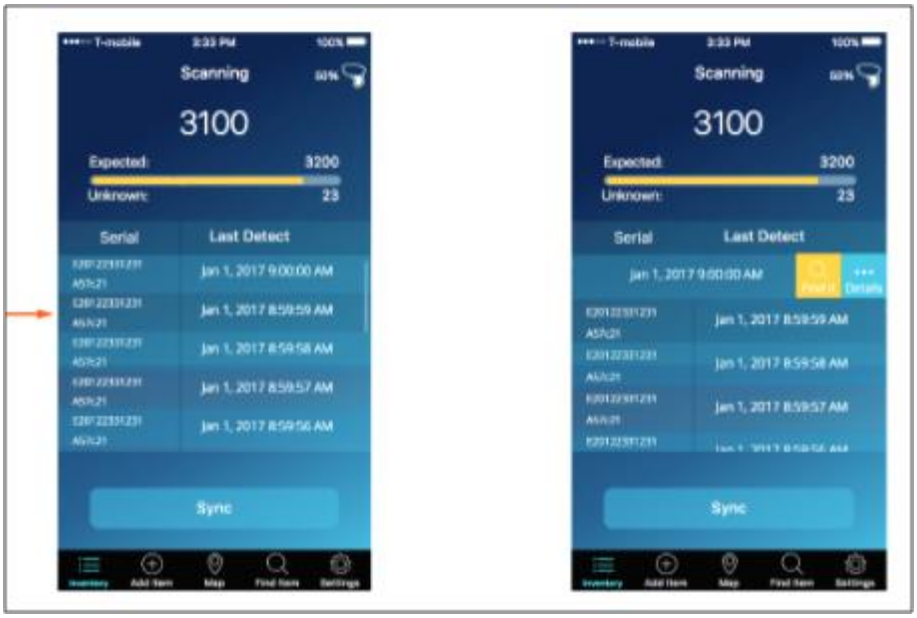


Figura 3.5. Counter View Controller

3.3.3.4. Configuración del Reader para el Control de Inventarios

La configuración del *reader* se implementó bajo la integración del *SDK* del lector *TSL-1128* dentro de la aplicación (véase figura 3.6). Posteriormente se adicionó una clase de tipo *NSObject* para la configuración del *reader* en modo inventario y la configuración del *barcode*.

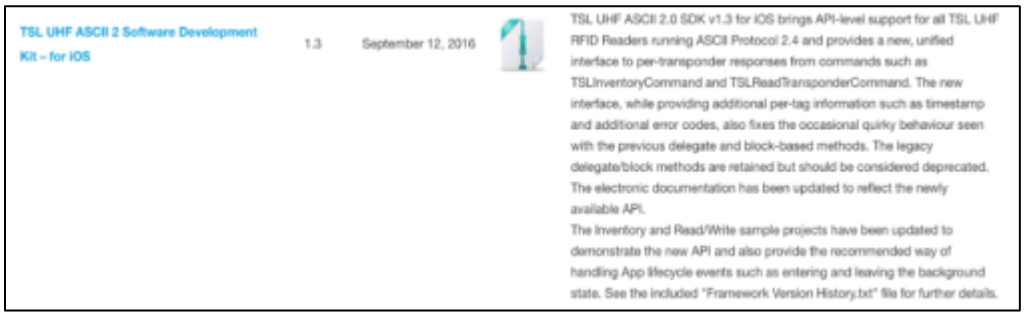


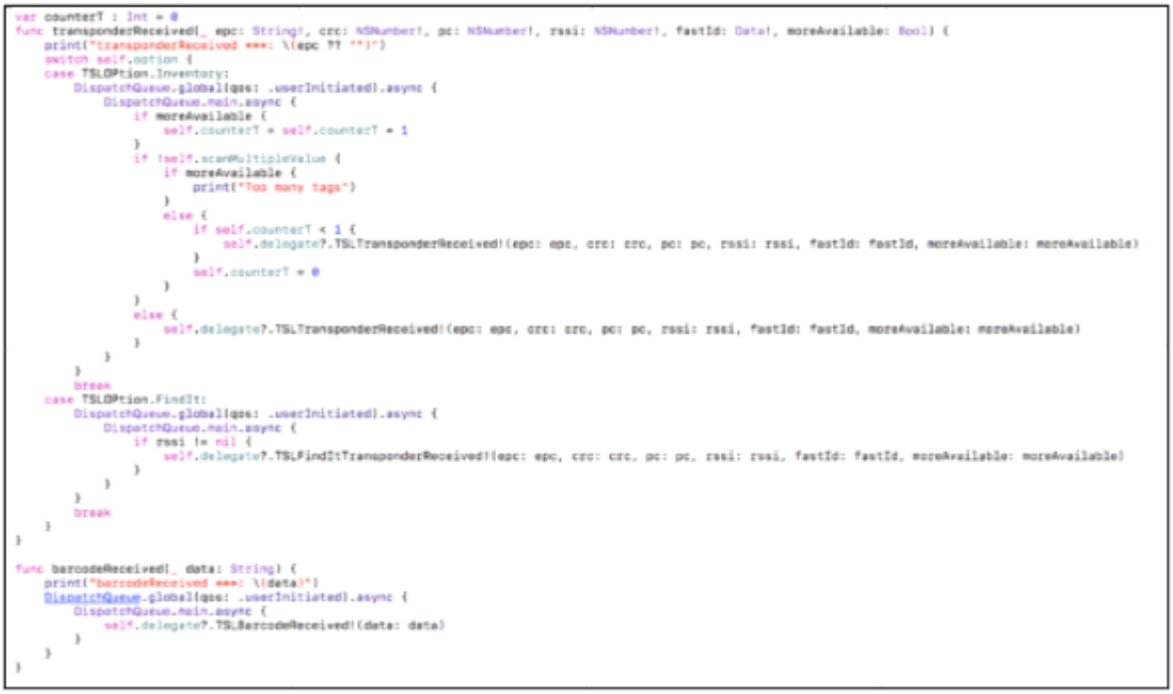
Figura 3.6. *SDK TSL-1128*

En la siguiente figura 3.7. se puede ver el código que se implementó para la configuración del *reader*:

```
class TSLCommandConfigure: NSObject, TSLInventoryCommandTransponderReceivedDelegate, TSLBarcodeCommandBarcodeReceivedDelegate {

    var scanMultipleValue = Bool()
    var option = TSLOPtion.Inventory
    weak var delegate: TSLCommandConfigureDelegate?

    override init() {
        super.init()
        self.scanMultipleValue = Bool()
    }

    init(isMultipleScan: Bool) {
        super.init()
        self.scanMultipleValue = isMultipleScan
        self.clearReaderResponder(isMultipleScan: self.scanMultipleValue)
    }

    init(epcTag: String) {
        super.init()
        self.clearResponderToFindTag(epcTag: epcTag)
    }

    func clearReaderResponder(isMultipleScan: Bool) {
        let appDelegate: AppDelegate = UIApplication.shared.delegate as! AppDelegate
        self.option = TSLOPtion.Inventory
        DispatchQueue.global(qos: .userInitiated).async {
            DispatchQueue.main.async {
                if let commander = appDelegate.commander {
                    if commander.isConnected {
                        commander.clearResponders()

                        let loggerResponder:TSLLoggerResponder = TSLLoggerResponder()
                        commander.add(loggerResponder)
                        commander.addSynchronousResponder()

                        let inventoryResponder = TSLInventoryCommand()
                        inventoryResponder.transponderReceivedDelegate = self
                        inventoryResponder.captureNonLibraryResponses = true
                        inventoryResponder.includeTransponderRSSI = TSL_TriState.YES
                        commander.add(inventoryResponder)

                        let value : Int32
                        if isMultipleScan {
                            TSLReaderConfiguration.setRFIDMultipleScanValues()
                            value = Int32(TSLReaderConfiguration.getTalValue(valueIn: Float(TSLLocalConfig.getMultiplePowerReader())))
                        }
                        else {
                            TSLReaderConfiguration.setRFIDSingleScanValues()
                            value = Int32(TSLReaderConfiguration.getTalValue(valueIn: Float(TSLLocalConfig.getSinglePowerReader())))
                        }

                        let barcodeResponder = TSLBarcodeCommand()
                        barcodeResponder.barcodeReceivedDelegate = self
                        barcodeResponder.captureNonLibraryResponses = true
                        commander.add(barcodeResponder)

                        if let command = TSLInventoryCommand.synchronousCommand() {
                            command.takeNoAction = TSL_TriState_YES
                            command.outputPower = value
                            commander.execute(command)
                        }

                        if let command = TSLSwitchActionCommand.synchronousCommand() {
                            command.asynchronousReportingEnabled = TSL_TriState_NO
                            command.resetParameters = TSL_TriState_YES
                            commander.execute(command)
                        }
                    }
                }
            }
        }
    }
```

Figura 3.7. Método para la configuración del lector RFID

En la siguiente figura 3.8. se puede ver el código que se implementó para los métodos delegados del *reader*:



```
var counterT : Int = 0
func transponderReceived(_ epc: String!, crc: NSNumber!, pc: NSNumber!, rssi: NSNumber!, fastId: Data!, moreAvailable: Bool) {
    print("transponderReceived ===: \(epc ?? "")")
    switch self.option {
    case TSLOption.Inventory:
        DispatchQueue.global(qos: .userInitiated).async {
            DispatchQueue.main.async {
                if moreAvailable {
                    self.counterT = self.counterT + 1
                }
                if !self.scanMultipleValue {
                    if moreAvailable {
                        print("Too many tags")
                    }
                    else {
                        if self.counterT < 1 {
                            self.delegate?.TSLTransponderReceived!(epc: epc, crc: crc, pc: pc, rssi: rssi, fastId: fastId, moreAvailable: moreAvailable)
                        }
                        self.counterT = 0
                    }
                }
                else {
                    self.delegate?.TSLTransponderReceived!(epc: epc, crc: crc, pc: pc, rssi: rssi, fastId: fastId, moreAvailable: moreAvailable)
                }
            }
        }
        break
    case TSLOption.FindIt:
        DispatchQueue.global(qos: .userInitiated).async {
            DispatchQueue.main.async {
                if rssi != nil {
                    self.delegate?.TSLFindItTransponderReceived!(epc: epc, crc: crc, pc: pc, rssi: rssi, fastId: fastId, moreAvailable: moreAvailable)
                }
            }
        }
        break
    }
}

func barcodeReceived(_ data: String) {
    print("barcodeReceived ===: \(data)")
    DispatchQueue.global(qos: .userInitiated).async {
        DispatchQueue.main.async {
            self.delegate?.TSLBarcodeReceived!(data: data)
        }
    }
}
```

Figura 3.8. Métodos delegados para la recepción de los *tags* en modo RFID y BARCODE

3.3.4. Iteración 3

Dentro de la iteración 3 se integró en la aplicación los mapas para lo cual se utilizaron los siguientes *frameworks*.

- *MapKit*
- *CoreLocation*
- *LocationManager*

Se implementó la integración de los mapas y aplicando el controlador de vista *MapViewController*. A continuación, en la figura 3.9. se puede ver la estructura del módulo de mapas:

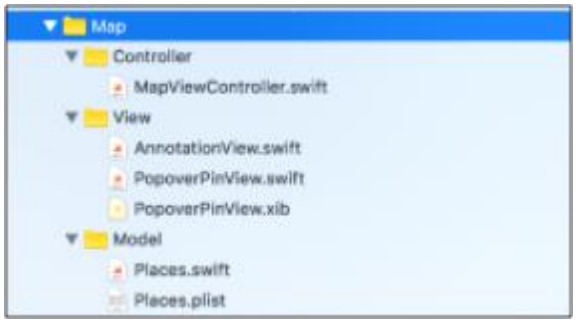


Figura 3.9. Módulo de Mapas

Se implementó el siguiente diseño para que el usuario pueda ver tanto su localización como la vista de los ítems mediante los reportes (véase figura 3.10).

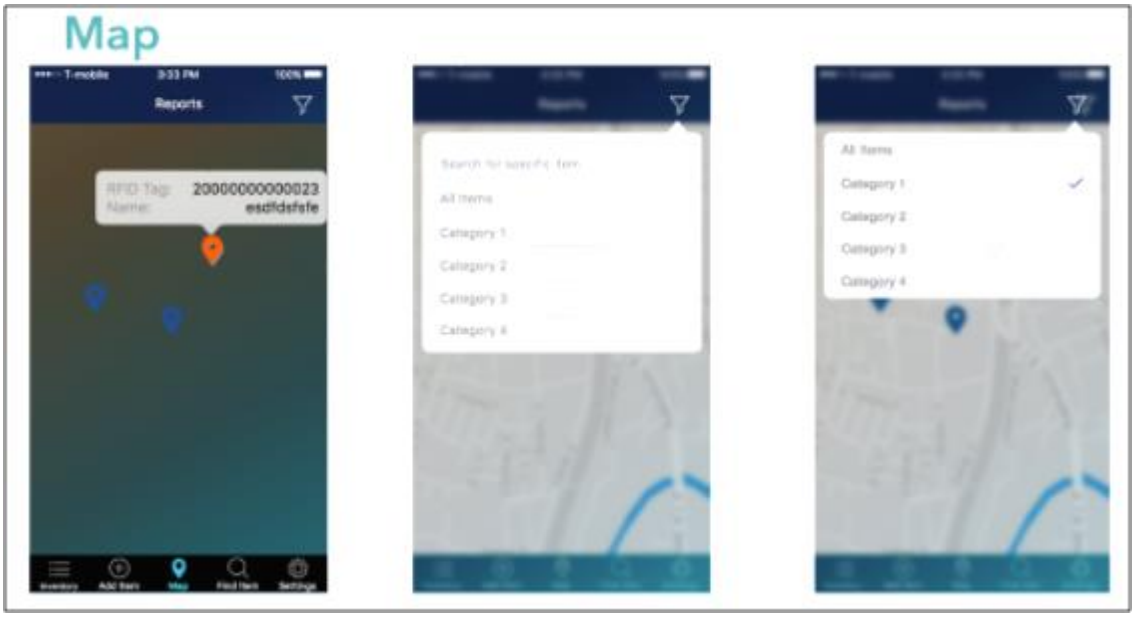


Figura 3.10. Diseño del Módulo de Mapas

Para ello se utilizó el *framework MapKit* el cual permite editar y colocar los pines en el mapa, bajo la información que se tiene de los mapas en el reporte. A continuación, en la figura 3.11. se puede observar el código de los siguientes métodos para el control de pines dentro del contenedor de mapas.

```swift
import UIKit
import MapKit
import Popover

extension MapViewController: MKMapViewDelegate {
    func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? {
        if annotation is MKUserLocation {
            return nil
        }

        var annotationView = mapView.dequeueReusableAnnotationView(withIdentifier: "annotationView")
        if annotationView == nil {
            annotationView = AnnotationView(annotation: annotation, reuseIdentifier: "annotationView")
            annotationView?.image = UIImage(named: Constant_Key.mapPinBlue)
            annotationView?.canShowCallout = false
        }
        else {
            annotationView?.annotation = annotation
        }
        return annotationView
    }

    func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer {
        let renderer = MKCircleRenderer(overlay: overlay)
        renderer.fillColor = UIColor.black.withAlphaComponent(0.5)
        renderer.strokeColor = UIColor.blue
        renderer.lineWidth = 2
        return renderer
    }

    func mapView(_ mapView: MKMapView, didSelect view: MKAnnotationView) {
        if view.annotation is MKUserLocation {
            return
        }
        let placeAnnotation = view.annotation as! Place
        let calloutView = PopoverPinView(frame: CGRect(x: 0.0, y: 0.0, width: 300.0, height: 60.0))
        calloutView.configureView(serial: placeAnnotation.title ?? "", name: placeAnnotation.subtitle ?? "")
        calloutView.center = CGPoint(x: view.bounds.size.width / 2, y: -calloutView.bounds.size.height*0.52)
        view.addSubview(calloutView)
        mapView.setCenter((view.annotation?.coordinate)!, animated: true)
    }

    func mapView(_ mapView: MKMapView, didDeselect view: MKAnnotationView) {
        if view.isKind(of: AnnotationView.self)
        {
            for subview in view.subviews
            {
                subview.removeFromSuperview()
            }
        }
    }

}
```

Figura 3.11. Métodos para la configuración de pines en el Mapa

3.3.5. Iteración 4

En esta iteración se integra el módulo *Find Item*, dentro de la aplicación móvil para lo cual se utilizarán los siguientes recursos:

- *JSONSwift*
- *SDK TSL-1128*

3.3.5.1. Scan Item Code

Dentro de esta sección se aplicó la configuración del lector de RFID para escanear *tags* cercanos a la ubicación del usuario para posteriormente encontrar el *tag* (véase figura 3.12).

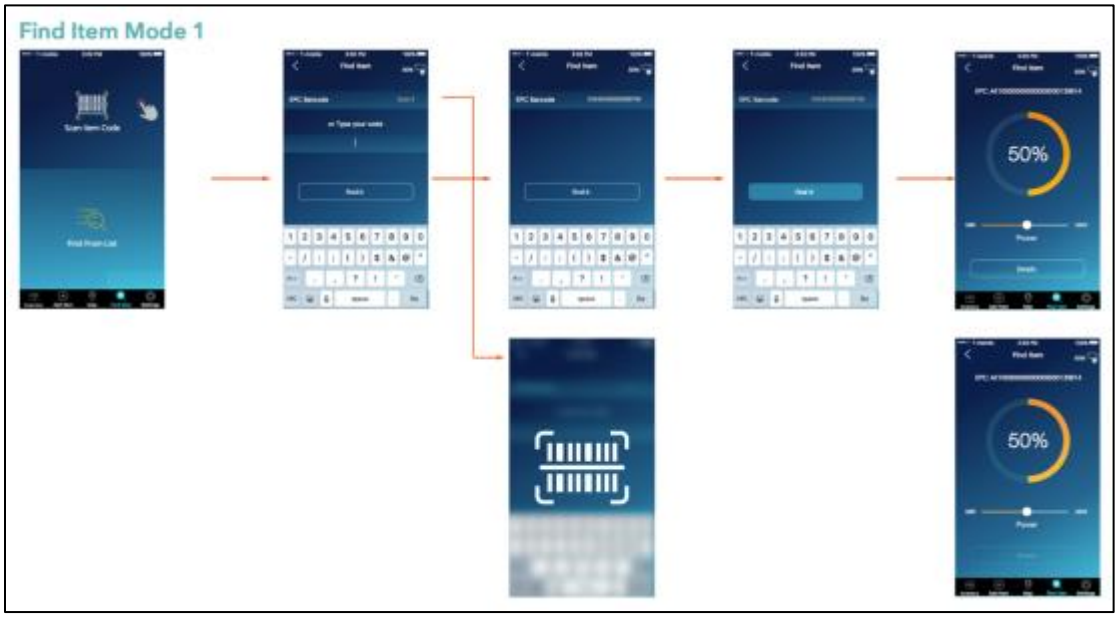


Figura 3.12. Sección *Scan Item Code*

3.3.5.2. Find From List

Dentro de esta sección se implementó el control de búsqueda de un *tag* a través de la selección de un reporte y posteriormente habilitar la búsqueda del *tag* seleccionado (véase figura 3.13).

Figura 3.13. Sección *Find from List*

3.3.5.3. Configuración y creación del Controlador *Find It*

Para la búsqueda de un determinado *tag* con el lector mediante la aplicación, se implementó la configuración del lector *RFID* utilizando el *framework TSLAsciiCommander* bajo el siguiente método de la clase *TSLCommandConfigure* (véase figura 3.14).

```
func clearResponderToFindTag(epcTag: String) {
    let appDelegate: AppDelegate = UIApplication.shared.delegate as! AppDelegate
    self.option = TSLOPtion.FindIt
    DispatchQueue.global(qos: .userInitiated).async {
        DispatchQueue.main.async {
            if let commander = appDelegate.commander {
                if commander.isConnected {
                    commander.clearResponders()
                    commander.addSynchronousResponder()
                    let inventoryResponder = TSLInventoryCommand()
                    inventoryResponder.transponderReceivedDelegate = self
                    inventoryResponder.captureNonLibraryResponses = true
                    inventoryResponder.includeTransponderRSSI = TSL_TriState_YES
                    commander.add(inventoryResponder)
                    TSLReaderConfiguration.setRFIDFindItValues(epcTag: epcTag)
                    if let saCommand = TSLSwitchActionCommand.synchronousCommand() {
                        saCommand.asynchronousReportingEnabled = TSL_TriState_NO
                        saCommand.resetParameters = TSL_TriState_YES
                        commander.execute(saCommand)
                    }
                }
            }
        }
    }
}
```

Figura 3.14. Método para la configuración *Find It*

3.3.6. Iteración 5

Dentro de este módulo llamado *Settings* se integró en la aplicación móvil el control y configuración del lector, en base a los comandos del lector *RFID* y configuraciones para Inventarios, modos de escaneo, sonido, vibración, poder del escaneo en modo *Single* y *Multiple*, selección de los lectores para la conexión con la aplicación mediante *Bluetooth* (véase figura 3.15).

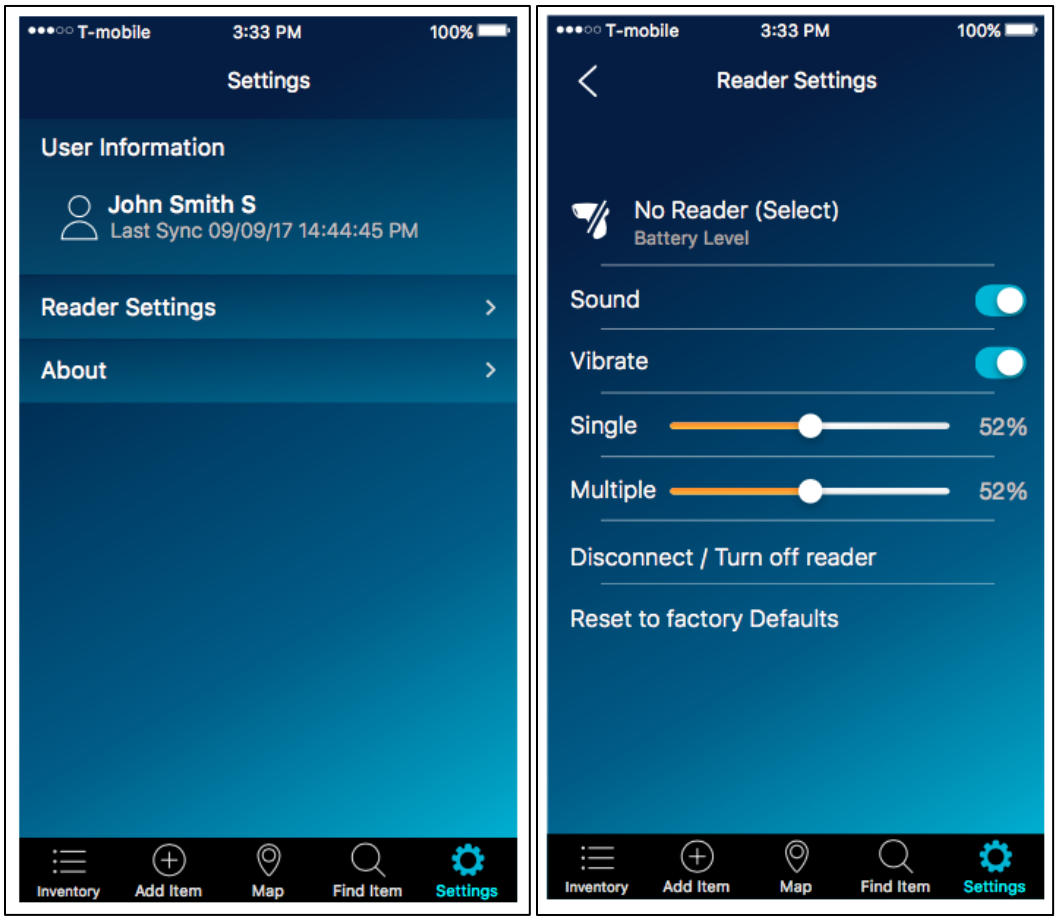


Figura 3.15. Módulo *Settings*

Para este módulo se ha implementado las siguientes clases para el control del lector *RFID*. A continuación, se puede observar en la figura 3.16. la estructura del módulo:

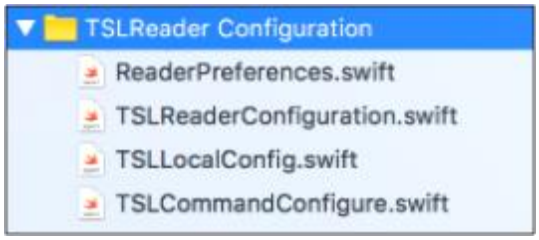


Figura 3.16. Configuración del Lector TSL-1128.

- *ReaderPreferences*, es la clase que se podrá obtener los valores del lector a partir del servidor para la configuración establecida por el usuario administrador.

- *TSLReaderConfiguration*, es la clase que controla todos los comandos del lector TSL-1128 para el uso de Inventarios, Sonido, Vibración, Selección del lector, valores del lector y otros.
- *TSLLocalConfig*, es la clase que obtendrá los valores por default del lector dentro de la aplicación.
- *TSLCommandConfigure*, es la clase que aplica el control y el uso de los métodos del lector a través de su *SDK* ya mencionado.

3.3.7. Iteración 6

En esta iteración se integra el servicio *BackEnd*, para la creación de nuevos ítems, la descarga de los reportes y la sincronización de estos, los recursos que se utilizaran son los siguientes:

- *Homebrew*, gestor de paquetes para MacOS.
- *NodeJS*, para el servicio de *BackEnd*.
- *MongoDB*, para la base de datos.
- *ExpressJS*, para la creación y configuración del servicio *BackEnd*.
- *RoboMongo*, herramienta para el gestor de base de datos en *MongoDB*.

3.3.7.1. Configuración del servidor BackEnd con MongoDB

Dentro de la configuración para el servidor se instaló con *Homebrew* gestor de paquetes para *MacOS*. a continuación, se puede ver en la figura 3.17 el archivo de configuración del servidor en *NodeJS*.

package.json

```
{
  "name": "tracker-service",
  "version": "1.0.0",
  "description": "tracker api service",
  "main": "tracker-server",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Paolo",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.17.2",
    "express": "^4.15.3",
    "method-override": "^2.3.9",
    "mongoose": "^4.10.4",
    "morgan": "^1.8.2",
    "node-restful": "^0.2.6"
  }
}
```

Figura 3.17. Archivo de Configuración del Servidor en *NodeJS*.

La configuración e instalación en *MongoDB*, se utilizó con el gestor de paquetes *Homebrew*, ya mencionado para su uso y con la ayuda de la herramienta *RoboMongo*.

3.3.7.2. Diseño de la base de datos en *MongoDB*

La base de datos está comprendida por 4 clases principales.

- Attendant, almacena los datos de los encargados que tienen ítems bajo su control.
- Item, almacena los datos de los productos que se tienen en para el inventario.
- Zone, almacena los datos de las zonas donde se encuentran los ítems.
- ReaderConfiguration, almacena los datos para la configuración personalizada de un lector de *RFID* con el uso de la aplicación.

3.3.7.3. Implementación del *API* para el acceso a los datos

Para el consumo y acceso a la base de datos se ha implementado un *API* con los siguientes *EndPoints* que se pueden ver en la figura 3.18.

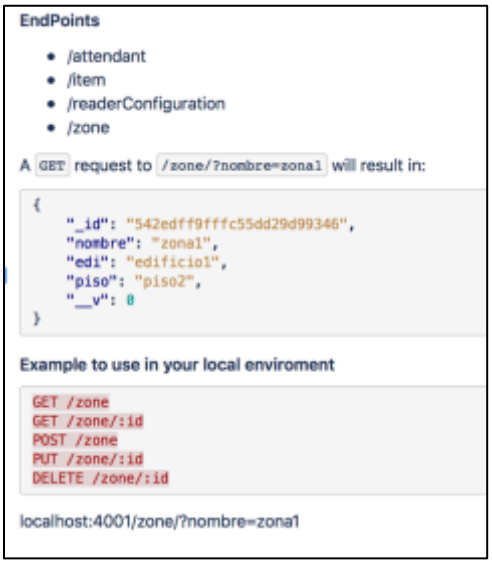


Figura 3.18. *EndPoints* para el consumo de datos.

En la siguiente figura 3.19 se puede observar la implementación de los métodos para el consumo de los reportes.

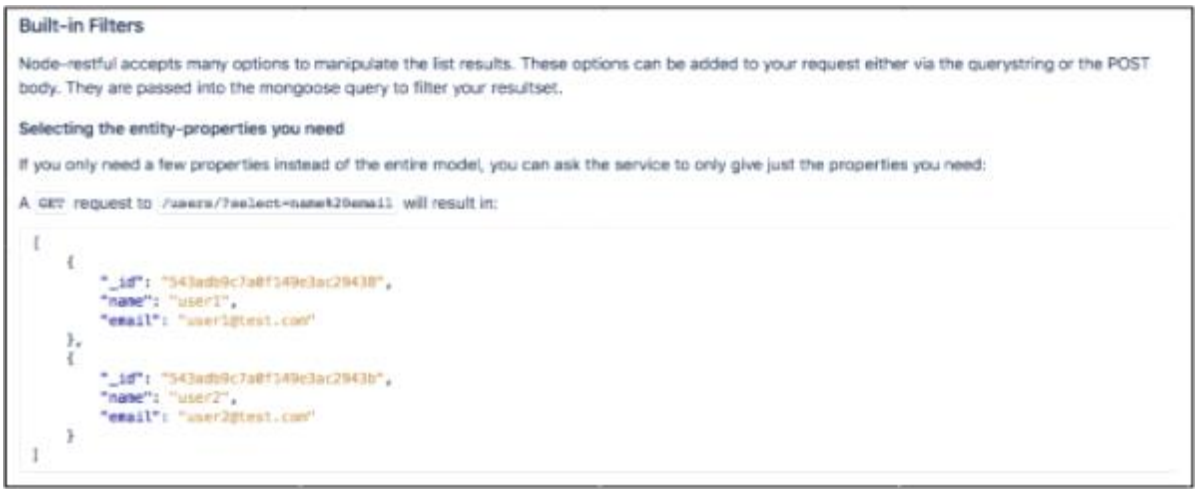


Figura 3.19. *Built in filters* para el consumo de datos.

3.4. Estabilización

En esta fase finaliza el desarrollo de la aplicación móvil para el control de inventarios basados en la tecnología de identificación por radiofrecuencia y se asegura la calidad del producto desarrollado. Para asegurar la calidad del producto en el desarrollo del mismo y según la metodología usada se aplicaron los siguientes mecanismos que aseguran la calidad de este:

3.4.1. Continuous Integration

Para esta aplicación se utilizó la herramienta de gestión de versiones *GIT,* en el cual se tiene una rama principal para la producción y la rama de desarrollo en la cual se realizaron las modificaciones de código, ya mencionado en la figura 3.2., la ruta del repositorio es el siguiente https://bitbucket.org/paolo-rm/tracker.

3.4.2. Estándares de Código

El código respeta los estándares determinados para el lenguaje de programación que se está utilizando, estos estándares se pueden observar en el Anexo C. Estos estándares están adaptados al lenguaje Swift considerando que en este lenguaje los nombres de los métodos de una clase empiezan con minúsculas como la declaración de variables.

3.4.3. Test-Driven Development

Se desarrolló varias pruebas de los métodos e integración de las librerías en el *IDE* propio de *XCode*, antes del mismo desarrollo antes del mismo desarrollo de estos, otorgando una idea previa a que estos métodos deberían hacer, este tipo de desarrollo se basa en realizar las pruebas unitarias previas al método de ejecutar un *build* en la aplicación para su posterior ejecución y al ser estos definidos en una interfaz, la refactorización se hace más rápida y constante, esta hace que sea cuestión de re ejecutar las pruebas para verificar si el *build* (construcción de la aplicación)

3.5. Pruebas del Sistema

En esta última fase del desarrollo de la aplicación según la metodología Mobile-D. El detalle de las pruebas realizadas sobre la aplicación se detalla en el capítulo siguiente “Pruebas y Resultados”.

CAPITULO IV
PRUEBAS Y RESULTADOS

4.1. Pruebas en la aplicación

4.1.1. Registro de etiquetas RFID

Para la implementación de las pruebas se aplicó el registro de los productos con las etiquetas *RFID* dentro del almacén de la Importadora de Ropa Ximena, la cual proporcionó permiso para el registro de los productos. El registro de los ítems es un paso importante hacia la prueba de la aplicación móvil en un caso real de uso del prototipo propuesto en la presente investigación. Para esto se tomó en cuenta el tipo de etiqueta *RFID* estándar motivo por el cual permite la identificación de cajas, *palets*, productos.

4.1.2. Etiqueta RFID pasiva

Los *tags* pasivos no poseen ningún tipo de alimentación. La señal que les llega de los lectores induce una corriente eléctrica mínima que basta para operar el circuito integrado CMOS de la etiqueta para generar y transmitir una respuesta. La mayoría de las etiquetas pasivas utiliza *backscatter* sobre la portadora recibida. Esto es, la antena ha de estar diseñada para obtener la energía necesaria para funcionar a la vez que para transmitir la respuesta por *backscatter*. (PANDAID, 2017).

Las etiquetas pasivas suelen tener distancias de uso práctico comprendidas entre los 10 cm (ISO 14443) y llegando hasta unos pocos metros (EPC e ISO 18000-6) según la frecuencia de funcionamiento, el diseño y tamaño de la antena (véase figura 4.1). Por su sencillez conceptual son obtenibles por medio de un proceso de impresión de las antenas. Como carecen de autonomía energética el dispositivo puede resultar muy pequeño: pueden incluirse en una pegatina o insertarse bajo la piel (etiquetas de baja frecuencia) (PANDAID, 2017).

Figura 4.1. Etiqueta RFID Pasiva
Fuente: (PANDAID, 2017)

El registro de la etiqueta en los productos fue colocado y juntamente registrado el producto en el sistema a continuación se puede ver en la figura 4.2.

Figura 4.2. Producto agregado con etiqueta RFID pasiva

A continuación, se puede ver en la figura 4.3. el registro de los productos en el sistema de los productos que fueron registrados.

Figura 4.3. Registro de los ítems

4.1.3. Funcionamiento de la Aplicación

En este apartado se describen los casos de estudio realizados para evaluar el desempeño de la aplicación. Estas pruebas fueron realizadas mediante un estudio de casos situaciones límites, ideales y normales a las que la aplicación pueda ser expuesta.

Primero se describen los dispositivos móviles en los cuales se realizaron las pruebas los cuales fueron llevados a cabo en la tabla 4.1. que se puede observar a continuación.

Tabla 4.1. Dispositivos utilizados para las pruebas.

Dispositivo	Versión de iOS	Resolución de pantalla
iPod Touch 4G	9.3.1	640 x 960
iPhone 6s	10.1.1	1080 x 1920
iPhone 7s	10.3.2	1080 x 1920

4.1.4. Caso de estudio 1

En este caso de estudio se tomó una situación ideal en la cual el usuario con el lector y el dispositivo móvil pudo escanear el almacén y la obtención de las etiquetas *RFID* sin ningún problema. Debido a la configuración del lector en la aplicación se pudo acceder al módulo de *Inventory* y *Find It* sin ningún problema.

Para ver la configuración del lector mencionado se puede ver en el anexo A la tabla 7.1.

a) Resultados

La aplicación cumple con los objetivos y requerimientos de funcionalidad.

b) Observaciones

Se debe considerar la potencia del lector para el escaneo dependiendo el lugar del almacén y considerando su distancia.

4.1.5. Caso de estudio 2

En este caso se ha considerado 50 etiquetas distribuidas en el almacén, para este caso se ha configurado las variables *Session* y *Target* del lector, se puede ver en el en la tabla 7.1. del Anexo A. Con el cual se pudo observar obtener el escaneo de las etiquetas a una distancia de aproximación de 1 metro.

a) Resultados

Para la búsqueda de las etiquetas con la configuración adecuada no se obtuvo problemas en cuanto al escaneo, pero si se tuvo el acceso de nuevas etiquetas que pertenecían a otro almacén.

b) Observaciones

Es necesario configurar la potencia del lector de acuerdo a la distancia que se tiene de un almacén a otro.

4.1.6. Caso de estudio 3

Se tomó en cuenta la distancia de 2 almacenes juntos en cuanto al escaneo de las etiquetas, para esto fue necesario configurar *Target, Session* y *QValue,* este tuvo el efecto en que el escaneo no sea repetitivo sobe la misma etiqueta sino al contrario pueda apagar la etiqueta y detectar las otras activas.

a) Resultados

La configuración del lector optimizó la búsqueda de las etiquetas que pertenecen a un almacén determinado y de esta manera evitar el conteo repetitivo de las etiquetas.

b) Observaciones

La aplicación cumple los objetivos y requerimientos de funcionalidad siempre y cuando se tome en consideración la configuración del lector, en los cuales se consideraron en el punto 3.3.3.4, además de los resultados obtenidos en el punto 3.3.5.3.

4.1.7. Caso de estudio 4

En este caso de estudio se sometió las pruebas en cuanto a la sincronización de la aplicación con los datos obtenidos del escaneo de un respectivo ambiente tanto cerrados como sótanos, o que están situados en la planta baja y en ambientes como sucursales y galerías.

a) Resultados

En sucursales y galerías la sincronización de los datos a través de la aplicación fue exitosa, pero se pudo observar que en algunos lugares cuando la señal de internet es demasiado baja y se intenta sincronizar dicha información no resulta ser exitosa. En especial esto sucede en ambientes completamente cerrados como sótanos.

b) Observaciones

Se implementó en la aplicación el método syncThingsToPatchData que se puede observar en el anexo B en la figura 7.1. para la sincronización de la información, con la cual este método consiste en sincronizar la información por intentos y por lotes, se tiene por defecto intentar sincronizar 4 veces, y los lotes de 50 etiquetas.

4.1.8. Caso de estudio 5

En este caso de estudio se tomó una situación en la que la etiqueta RFID se encuentre dañado o quemado.

a) Resultados

La aplicación cumple los objetivos y requerimientos de funcionalidad, siempre y cuando el código no tenga daños en más del 25% en cuanto a la antena de la etiqueta *RFID* pasiva.

b) Observaciones

En este caso de estudio se tuvieron distintos resultados, sin embargo, cuando la etiqueta se encontraba rota o partida a la mitad, el escaneo e identificación del código no fueron posibles.

4.2. Resultados

La evaluación realizada sobre la aplicación fue realizada mediante el uso de encuestas dirigidas al personal de la importadora y parte a la población, el cual determino la utilidad, la facilidad de uso y la actitud hacia el uso de la aplicación construida estuviera disponible en el mercado. También se pudo recolectar posibles cambios y sugerencias para mejorar el producto.

Respecto al número de encuestas, según el teorema del límite central, si una población tiene media μ y desviación típica σ, la distribución muestra de medias $L(x)$ se aproxima a una distribución $N\ (\mu, \sigma/n)$ cuando $n > 30$.

4.2.1. Aceptación de la aplicación

Se creó un cuestionario que fue realizado en base al "modelo tecnológico de aceptación" o TAM por sus siglas en inglés (Abu-Dalbouh, 2013). En este modelo se intenta determinar cómo los usuarios están dispuestos a aceptar o rechazar una nueva tecnología, la cual llega a ser en forma de aplicación para computadora, página web, sistema, aplicación móvil u otro.

El cuestionario cuenta con dos segmentos: utilidad y facilidad de uso. De acuerdo a estudios y encuestas realizadas por el autor al modelo TAM, la usabilidad es más importante que la facilidad de uso en un factor de 1.5. Las encuestas fueron realizadas a la población y se llevaron a cabo entre los días 19 y 21 de noviembre del año 2017 en zonas comerciales de la ciudad de La Paz

A continuación, en la Tabla 4.2. se puede ver el cuestionario de utilidad.

Tabla 4.2. Encuesta de Utilidad de la aplicación.

#	*Pregunta*	*Respuestas*
1	¿Logró encontrar rápidamente el ítem utilizando la aplicación móvil?	Si No
2	¿Logró conectar el dispositivo rápidamente?	Si No
3	¿Considera que la información del ítem desplegada en la aplicación es la necesaria?	Para nada En cierta manera Definitivamente
4	¿Le parece adecuada la distancia para encontrar los ítems?	Si No Posiblemente
5	¿Considera que el tiempo en el que se actualizo la información acerca de un ítem fue el necesario?	Para nada En cierta manera Definitivamente
6	¿Cree que la aplicación puede reducir el tiempo de búsqueda de los ítems?	Para nada En cierta manera Definitivamente
7	¿Le fue sencillo configurar la aplicación?	Si No Parte que sí y otro no
8	¿Logro obtener información de los registros de movimientos de productos dentro del almacén?	Si No
9	¿Logro obtener información oportuna acerca de los productos existentes en el almacén?	Si No Incompleta

El cuestionario de facilidad de uso se detalla en la Tabla 4.3. que se muestra a continuación:

Tabla 4.3. Encuesta de Facilidad de uso.

#	Pregunta	Respuestas
1	¿Necesitaría algún manual para poder utilizar la aplicación móvil?	Si No
2	¿Hizo mucho esfuerzo para aprender a utilizar la aplicación?	Mucho esfuerzo Poco esfuerzo Ningún esfuerzo
3	¿Cómo calificaría el desempeño de la aplicación?	Muy malo Malo Regular Bueno Muy bueno
4	¿Volvería a usar la aplicación?	Si No

4.2.2. Resultado Utilidad y Facilidad de uso

Los cuestionarios proporcionaron los siguientes resultados.

4.2.2.1. Utilidad

Los resultados de los cuestionarios se muestran desde la figura 4.3. hasta la figura 4.12.

En la siguiente figura 4.4. se muestra el resultado de cumplimiento al encontrar un ítem con la aplicación móvil.

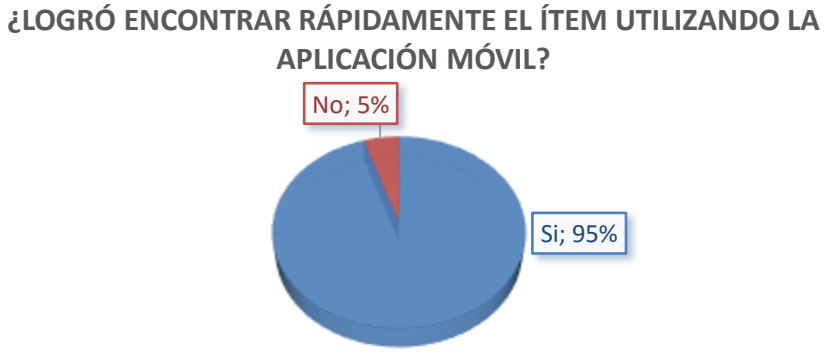


Figura 4.4. Uso de la aplicación móvil para encontrar un ítem.

Se puede observar en la figura 4.5. que el personal encuestado tuvo facilidad de conectar el lector con el dispositivo y se puede ver que un 90% no tuvo problemas al realizar dicha conexión.

Figura 4.5. Conexión con el lector RFID mediante la aplicación.

En la figura 4.6. los usuarios mostraron la factibilidad de uso de la aplicación para ver un determinado ítem con sus respectivos detalles en la aplicación.

Figura 4.6. Ver información de un determinado ítem.

En la figura 4.7. se puede observar que le pareció adecuado a un 65% a los usuarios en el escaneo a distancia.

Figura 4.7. Distancia del lector para el escaneo de *tags*.

La actualización de los ítems en tiempo real, fue un factor muy importante para los usuarios que un 75% considera importante este módulo que se puede observar en la figura 4.8.

Figura 4.8. Actualización de la Información.

El 80% de los usuarios confirmo que la aplicación efectivamente reduce el tiempo de búsqueda de los ítems, a continuación, el resultado en la figura 4.9.

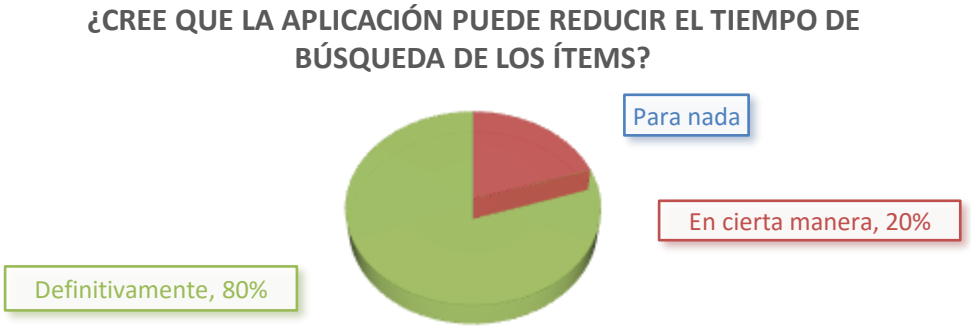


Figura 4.9. Reducción del tiempo de búsqueda de los ítems.

Sobre la configuración de la aplicación los usuarios no tuvieron muchos problemas al respecto como se puede observar en la figura 4.10.

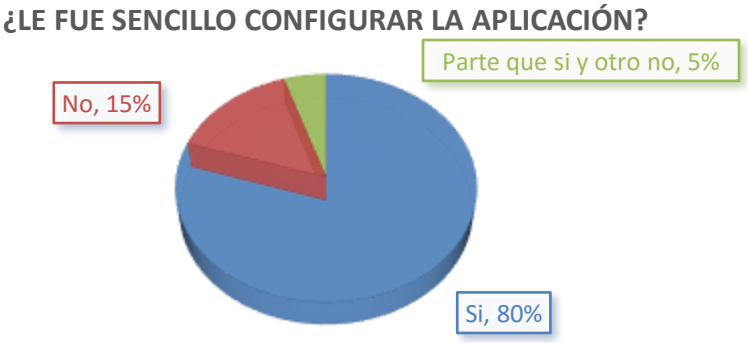


Figura 4.10. Configuración de la aplicación

En cuanto a la información sobre los registros de movimientos de los ítems un 100% de los usuarios logro obtener la satisfacción de la obtención de los datos como se puede ver en la figura 4.11.

Figura 4.11. Información de los registros de movimientos de productos dentro del almacén.

Sobre el control del almacén y la verificación de los productos el 95% de los usuarios pudo obtener ese detalle cómo se puede ver en la figura 4.12.

Figura 4.12. Información de los ítems en el almacén.

4.2.2.2. Facilidad de uso

El cuestionario de facilidad de uso sobre la aplicación móvil para el control de inventarios basados en la tecnología por radiofrecuencia, brinda los datos que se pueden observar desde la figura 4.13 hasta la figura 4.16.

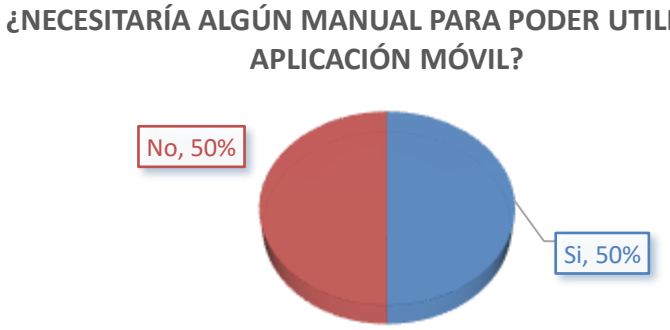


Figura 4.13. Necesidad de manual para la aplicación.

Los resultados obtenidos brindan a los usuarios encuestados creen que el uso de la aplicación es bastante intuitivo y simple, un 10% difiere de esta ya que había que explicar la interacción de lector con la aplicación (véase figura 4.14).

Figura 4.14. Esfuerzo para la utilización de la aplicación.

Para utilizar la funcionalidad no hubo problemas, pero si al momento del uso del lector ya que algunos desconocían del uso (véase figura 4.15).

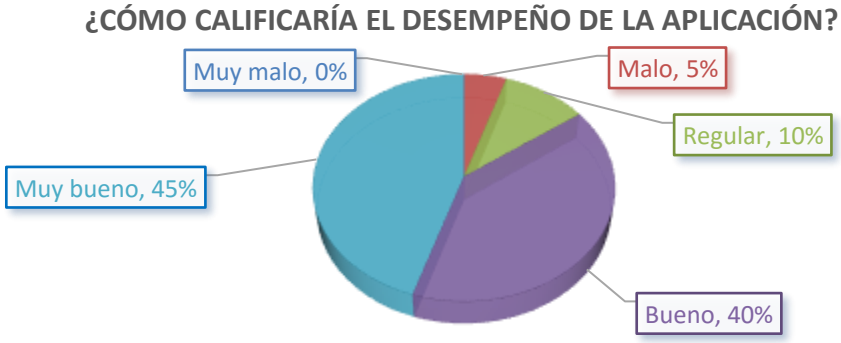


Figura 4.15. Desempeño de la aplicación

La aplicación en general tiene un desempeño entre bueno y muy bueno para los usuarios encuestados (véase figura 4.16).

Figura 4.16. Fidelidad del usuario.

4.3. Pruebas sobre la aplicación

Las pruebas de la aplicación fueron realizadas el día 23 y 24 de junio de 2017 en la ciudad de La Paz por el personal de la Importadora Ximena y personas de centros comerciales ubicados en la calle Eloy Salmon en la zona central. Los resultados fueron satisfactorios ya que probaron la confiabilidad de la aplicación en situaciones en las que se cuenta con acceso a internet y en situaciones en las que es limitado o nulo. Sin embargo, en la situación en la que se tenga problemas de acceso a internet el usuario no podrá descargar los reportes como la información de un ítem registrado. A continuación, se puede observar en las figuras 4.17 a la figura 4.24 las respectivas pruebas de la aplicación en la importadora Ximena.

Figura 4.17. Sucursal vista de frente

Figura 4.18. Sucursal vista de costado lado izquierdo

Figura 4.19. Espacio para el etiquetado de las Tag RFID

Figura 4.20. Productos para ser etiquetados con el Tag RFID

Figura 4.21. Ubicación del depósito ya descargado con los productos

Figura 4.22. Ubicación del depósito no inventariado

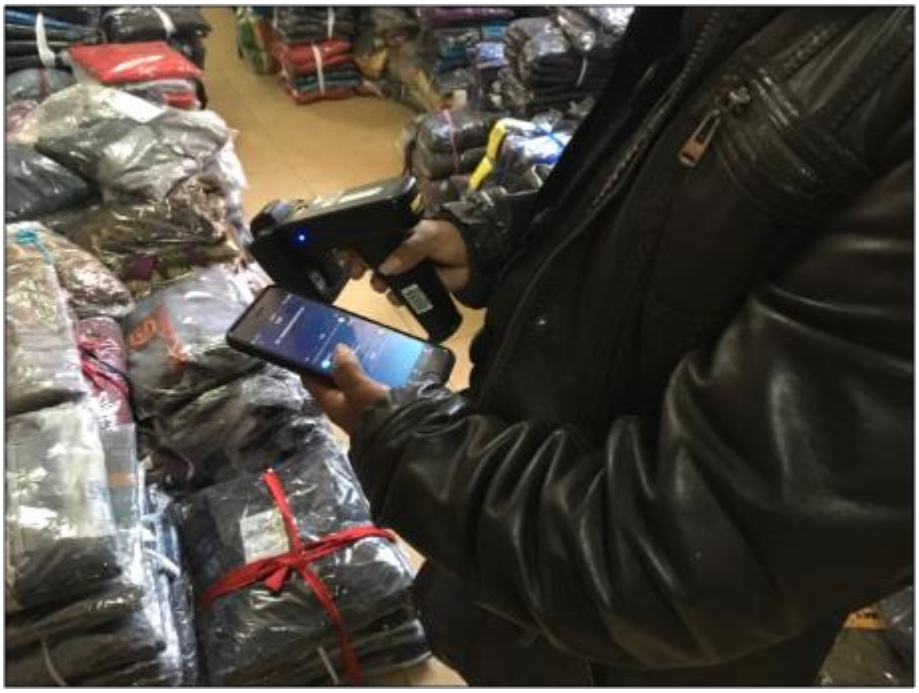


Figura 4.23. Pruebas realizadas con el personal de la Importadora

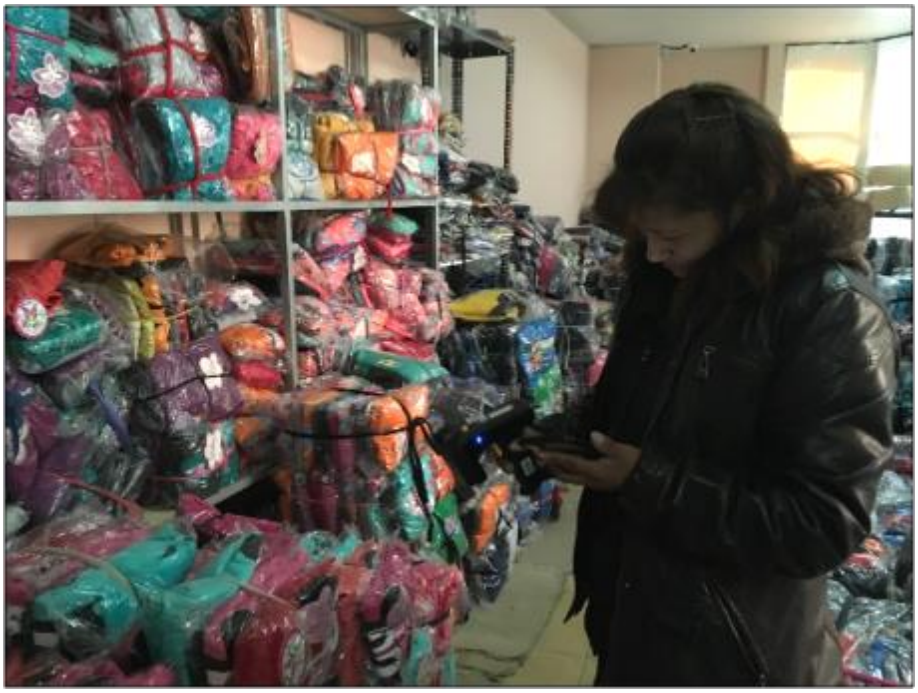


Figura 4.24. Pruebas realizadas desde otro ángulo

CAPITULO V
CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

En primera instancia, se ha logrado el objetivo propuesto al inicio de la investigación:

- Desarrollar una aplicación móvil en *iOS* capaz de registrar, contabilizar y ubicar los productos inventariados además de implementar el control y opción de búsqueda avanzada a través del uso de la tecnología *RFID*.

Con los resultados obtenidos del desarrollo de una aplicación móvil, se afirma que es posible el optimizar el control de inventarios basados en la tecnología *RFID,* ya que el usuario logra obtener información tanto de los ítems registrados en una determinada zona, como la búsqueda y el conteo de los ítems en un almacén. Además de la sincronización de dicha información para actualizar la información de un inventario.

Por otro lado, los objetivos secundarios se obtuvieron de acuerdo a los resultados esperados:

- Elaborar la aplicación móvil denominado TRACKER en ambiente iOS capaz de registrar, contabilizar y ubicar los productos inventariados de un almacén.

Se realizó el desarrollo de la aplicación en *iOS* con los requerimientos tecnológicos para el control de un almacén y la búsqueda de un producto puedan ser considerados como una herramienta y la automatización de procedimientos manuales que se realizan.

- Implementar una configuración para el TSL-1128 a través del SDK de iOS permitiendo personalizar la optimización de búsquedas de *Tags RFID* mediante comandos del lector.

Durante la implementación de la configuración del lector *TSL-1128*, se ha observado que el lector tiene diferentes configuraciones tanto para controlar la búsqueda de una determinado *Tag RFID* utilizado en el módulo *Find It*, como el conteo de múltiples *Tags* utilizado dentro del módulo de *Inventory*. Además de la configuración respecto a la cantidad de *Tags* y la potencia del lector que este puede realizar.

- Demostrar los beneficios del uso de la aplicación móvil para el registro, contabilización y ubicación de los productos de un almacén usando la tecnología *RFID*.

En las pruebas que se realizaron, fue necesario el registro de los productos con las etiquetas, por lo tanto, los reportes de los ítems registrados en la base de datos fueron de gran utilidad para visualizarlos a través de la aplicación y así obtener información sobre estos que pertenecen en un determinado almacén.

Después de identificar el control de traslado de productos que contengan un *TAG RFID* fue posible obtener dicha información usando el módulo de *Inventory* implementada en la aplicación por lo tanto esto fue considerablemente aceptable para el personal que realiza el conteo y control periódico de los almacenes y sucursales que tienen bajo su vigilancia.

- Implementar opción de búsqueda de un *Tag RFID* determinado a través de la aplicación móvil.

Finalmente, durante la implementación de la opción de búsqueda, fue necesario investigar y configurar el modo de escaneo del lector, ya que en el módulo *Find It* se implementó de manera distinta debido a que este trabaja con el objetivo de buscar una etiqueta, en cambio el módulo de *Inventory* trabaja con el modo de *Single* y *Multiple* para el escaneo de *Tags*.

5.2. Recomendaciones

Para el desarrollo de la aplicación móvil que permita optimizar el control de inventarios basados en la tecnología *RFID*, es necesario aplicar conceptos de *IoT* (Internet de las cosas) para los futuros proyectos el cual se invita a profundizar sobre el uso de esta tecnología no solo para el área de inventarios, sino que puedan ser aplicados en el área de la medicina, como también en el uso de marketing digital que puede ser de vital importancia de modo que el estudiante motive su aprendizaje.

Se recomienda incluir en la aplicación para seguir mejorando el desarrollo, el uso de la aplicación sin necesidad de acceso a internet aplicando base de datos *SQLite* o *CoreData*.

La recomendación principal es que se busque innovar el uso de aplicaciones portables en nuestro país, ya no debería solo basarse en cosas manuales o uso de hojas de cálculos, sino es dar al usuario esa facilidad otorgándole herramientas con las cuales pueda explotar en su desempeño de su trabajo y que el uso de nuevas tecnologías no sea ajeno en su diario vivir.

BIBLIOGRAFÍA

ABI. (24 de Marzo de 2015). *boliviaentusmanos.com*. Obtenido de boliviaentusmanos.com: http://www.boliviaentusmanos.com/noticias/economia/148253/anh-monitorea-el-100-de-surtidores-del-pais-con-tecnologia-rfid.html

BI Intelligence. (25 de Enero de 2015). *Have you considered BI Intelligence for your professional needs?* Obtenido de Intelligence.businessinsider.com: https://intelligence.businessinsider.com/

Carreon, A. (24 de Febrero de 2014). *Revista Merca2.0*. Obtenido de Revista Merca2.0: https://www.merca20.com/5-usos-de-rfid-en-retail/

Dipole. (10 de Abril de 2017). *Dipolerfid.es*. Obtenido de Dipolerfid.es: http://www.dipolerfid.es/es/tecnologia-RFID

DIR&GE. (16 de Marzo de 2017). *DIR&GE*. Obtenido de DIR&GE: http://directivosygerentes.es/ecommerce/noticias-ecommerce/inditex-invertira-1-500-millones-tecnologia-retail-2017

Emb. (25 de Junio de 2017). *Emb.cl*. Obtenido de Emb.cl: http://www.emb.cl/negociosglobales/articulo.mvc?xid=1391&edi=69&xit=rfid-para-identificacion-de-productos-individuales

Ferrín, A. (2007). *Gestión de stocks en la logística de almacenes.* FC Editorial.

Fresneda, S. (2016). *Aplicación de patrones de diseño para la resolución de problemas de software en el desarrollo de una aplicación móvil iOS.* Valencia, España: Universitat Politècnica de València.

González, J., Morini, S., & Do Nascimento, E. (2005). *Control y gestión del área comercial y de producción de la PYME.* Netbiblo.

HTK-ID. (10 de Abril de 2017). *Htk-id.com*. Obtenido de Htk-id.com: http://htk-id.com/tecnologia-rfid/

Journal, R. (16 de Enero de 2014). *Rfidjournal.com*. Obtenido de Rfidjournal.com: http://www.rfidjournal.com/articles/view?1338/

KPCB. (25 de Enero de 2015). *Kpcb.com*. Obtenido de Kpcb.com: http://www.kpcb.com/internet-trends

Moya, M. J. (1999). *Control de inventarios y teoría de colas.* EUNED.

MTP. (13 de Diciembre de 2015). *Pruebas de calidad en Aplicaciones Móviles*. Obtenido de Mtp.es: http://www.mtp.es/noticias/48-pruebas-de-calidad-en-aplicaciones-moviles

Muller, M. (2005). *Fundamentos de administración de inventarios.* Editorial Norma.

PANDAID. (8 de Abril de 2017). *¿Qué es una etiqueta RFID?* Obtenido de Panda ID Soluciones: http://www.pandaid.com/que-es-una-etiqueta-rfid/

Perdomo, A. (2004). *Fundamentos de control interno.* Cengage Learning Editores.

Portal Industrial. (28 de Diciembre de 2010). *Portal-industrial.com.ar*. Obtenido de Portal-industrial.com.ar: http://www.portal-industrial.com.ar/articulos/rfid-radio-frequency-identification----identificacion-por-radiofrecuencia/205/

Roberti, M. (16 de Enero de 2014). *Rfidjournal.com*. Obtenido de Rfidjournal.com: http://www.rfidjournal.com/articles/view?1338/

Rodríguez, T. (29 de Septiembre de 2015). *Métodos aplicables para el desarrollo de aplicaciones móviles*. Obtenido de Genbetadev.com: https://www.genbetadev.com/desarrollo-aplicaciones-moviles/metodos-aplicables-para-el-desarrollo-de-aplicaciones-moviles

Santos, R. (06 de Marzo de 2013). *Eoi.es*. Obtenido de Eoi.es: http://www.eoi.es/blogs/scm/2013/03/06/tecnologia-rfid/

Solís, I. (26 de Septiembre de 2012). *Htk-rfid.com*. Obtenido de Htk-rfid.com: http://www.logisticamx.enfasis.com/notas/65155-explica-htk-bondades-tecnologia-rfid

Statista. (25 de Enero de 2015). *Mobile app revenues 2015-2020 | Statistic*. Obtenido de Statista: http://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/

TSL-UK. (10 de Abril de 2017). *Technology Solutions (UK) Ltd*. Obtenido de Technology Solutions (UK) Ltd: https://www.tsl.com/products/1128-bluetooth-handheld-uhf-rfid-reader/

Urbina, R. d. (2011). *Tutorial sobre circuitos RFID.* Puebla, México: Universidad de las Américas.

Violino, B. (16 de Enero de 2014). *Rfidjournal.com*. Obtenido de Rfidjournal.com: http://www.rfidjournal.com/articles/view?1338/2

ANEXOS

ANEXO A:

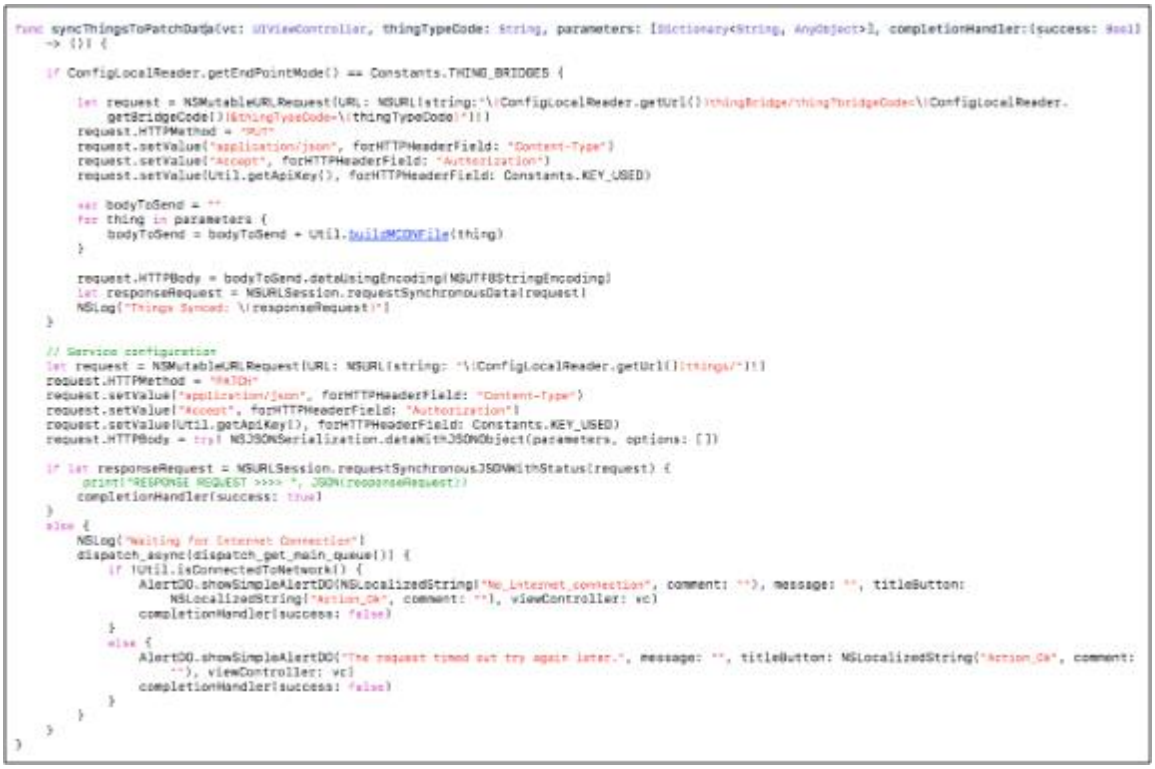
Configuración del lector para el uso del escaneo implementado en la aplicación:

Tabla 7.1. Encuesta de Facilidad de uso.

NAME	VALUE
QGeigerScan	1
QMultiScan	5
QSingleScan	0
ReaderAutoConnect	FALSE
ReaderMultiplePower	100
ReaderSinglePower	50
ReaderSoundEnabled	FALSE
ReaderVibrateEnabled	FALSE
selectGeigerScan	NO
selectMultiScan	NO
selectSingleScan	YES
sessionGeigerScan	S0
sessionMultiScan	S2
sessionSingleScan	S0
targetGeigerScan	B
targetMultiScan	A
targetSingleScan	A

ANEXO B:

En la figura 7.1. se puede observar el método que ha permitido mejorar la sincronización de los datos del reporte al servidor.



```
func syncThingsToPatchData(vc: UIViewController, thingTypeCode: String, parameters: [Dictionary<String, AnyObject>], completionHandler:(success: Bool)
    -> ()) {

    if ConfigLocalReader.getEndPointMode() == Constants.THING_BRIDGES {

        let request = NSMutableURLRequest(URL: NSURL(string:"\(ConfigLocalReader.getUrl())thingBridge/thingTbridgeCode=\(ConfigLocalReader.
            getBridgeCode())&thingTypeCode=\(thingTypeCode)"))
        request.HTTPMethod = "PUT"
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")
        request.setValue("Accept", forHTTPHeaderField: "Authorization")
        request.setValue(Util.getApiKey(), forHTTPHeaderField: Constants.KEY_USED)

        var bodyToSend = ""
        for thing in parameters {
            bodyToSend = bodyToSend + Util.buildJSONFile(thing)
        }

        request.HTTPBody = bodyToSend.dataUsingEncoding(NSUTF8StringEncoding)
        let responseRequest = NSURLSession.requestSynchronousData(request)
        NSLog("Things Synced: \(responseRequest)")
    }

    // Service configuration
    let request = NSMutableURLRequest(URL: NSURL(string: "\(ConfigLocalReader.getUrl())things/"))
    request.HTTPMethod = "PATCH"
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")
    request.setValue("Accept", forHTTPHeaderField: "Authorization")
    request.setValue(Util.getApiKey(), forHTTPHeaderField: Constants.KEY_USED)
    request.HTTPBody = try! NSJSONSerialization.dataWithJSONObject(parameters, options: [])

    if let responseRequest = NSURLSession.requestSynchronousJSONWithStatus(request) {
        print("RESPONSE REQUEST >>>> ", JSON(responseRequest))
        completionHandler(success: true)
    }
    else {
        NSLog("Waiting for Internet Connection")
        dispatch_async(dispatch_get_main_queue()) {
            if !Util.isConnectedToNetwork() {
                AlertDO.showSimpleAlertDO(NSLocalizedString("No_Internet_connection", comment: ""), message: "", titleButton:
                    NSLocalizedString("Action_Ok", comment: ""), viewController: vc)
                completionHandler(success: false)
            }
            else {
                AlertDO.showSimpleAlertDO("The request timed out try again later.", message: "", titleButton: NSLocalizedString("Action_Ok", comment:
                    ""), viewController: vc)
                completionHandler(success: false)
            }
        }
    }
}
```

Figura 7.1. Método SyncThingsToPatchData

ANEXO C:

Coding Standards

Test Naming:
Test classes names should start with "Test" word for example: TestUserList
Test methods names should start with "test" word for example: testAddOneUser()

Class comments:

```
/**
 * @author AHa, KLe              ORIGINAL AUTHORS
 * @date   8.10.2003            DATE OF CREATION
 * @task   2.1                  FOR WHICH TASK THE CLASS WAS CREATED
 *
 * This class implements functionality needed to
 * retrieve stocks information from the database
 */                              CLASS DESCRIPTION, GOOD COMMENTS
```

Method comments:

```
/**
 * @author Pky, AHa, KLe        ALL AUTHORS
 * @date   8.10.2003            DATE OF CREATION
 * @task   2.15, 3.1            TASKS FOR BACKTRACKING, ALL TASKS
 * changes 1.2.3003 AHa, KLe: Changed to implement new BACK-functionality
 *                              CHANGES MADE AFTER CREATION
 *
 * This method returns a single stock by given ID
 *                              METHOD DESCRIPTION, GOOD COMMENTS
 * @param AID                   PARAMETERS, DECENT COMMENTS
 * @return Stock stock          RETURN VALUE, DECENT COMMENTS
 */
```

Variable notation:

```
Static String GFooBar;          CLASS STATIC VARIABLE WITH G PREFIX

int FBranchID;                  CLASS VARIABLES WITH F PREFIX

public Stock get(int AID)       METHOD PARAMETER WITH A-PREFIX
{
        .                       ALL PARENTHESES ON THEIR OWN ROWS
        .
}
public class MainMenuScreen     CLASS NAMES START WITH UPPER CASE LETTER

public void addNew()            METHOD NAMES START WITH LOWER CASE LETTER
```

A TASK IS NOT DONE BEFORE THESE STANDARDS HAVE BEEN FOLLOVED!

Figura 7.2. Estándares de codificación de la metodología Mobile-D