UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

CARRERA DE SISTEMAS Y COMPUTACIÓN

Proyecto de Investigación previo a la obtención del título de Ingeniero en Sistemas y Computación

TRABAJO DE TITULACIÓN

METODOLOGÍA TEST DRIVEN DEVELOPMENT APLICADA EN EL DESARROLLO DEL SISTEMA DE CONTROL DE INVENTARIO PARA EL ALMACÉN ALEJANDRA

Autor(es):

Ronald David Macas Cando

Angel Mauricio Ramos Cepeda

Tutor:

Ing. Miryan Estela Narváez Vilema

Riobamba - Ecuador

2021

PÁGINA DE ACEPTACIÓN

Los miembros del tribunal de graduación del proyecto de investigación de título: **"METODOLOGÍA TEST DRIVEN DEVELOPMENT APLICADA EN EL DESARROLLO DEL SISTEMA DE CONTROL DE INVENTARIO PARA EL ALMACÉN ALEJANDRA"**, presentado por: Ronald David Macas Cando y Angel Mauricio Ramos Cepeda, dirigido por: **Ing. Miryan Estela Narváez Vilema, PhD.**

Una vez escuchado la defensa oral y revisado el informe final del proyecto de investigación con fines de graduación escrito en la cual se ha constatado el cumplimiento de las observaciones realizadas, remite la presente para uso y custodia en la biblioteca de la Facultad de Ingeniería de la UNACH.

Para constancia de lo expuesto firman.

Miryan Estela Narváez Vilema
Presidente del tribunal

Firmado electrónicamente por:
MIRYAN ESTELA NARVAEZ VILEMA

..... **Firma**

Wayner Xavier Bustamante Granda
Miembro del tribunal

Firmado electrónicamente por:
WAYNER XAVIER BUSTAMANTE GRANDA

..... **Firma**

Marlon Javier Silva Castañeda
Miembro del tribunal

Firmado electrónicamente por:
MARLON JAVIER SILVA CASTAÑEDA

..... **Firma**

DECLARACIÓN EXPRESA DE TUTORÍA

En calidad de tutor del tema de investigación "**METODOLOGÍA TEST DRIVEN DEVELOPMENT APLICADA EN EL DESARROLLO DEL SISTEMA DE CONTROL DE INVENTARIO PARA EL ALMACÉN ALEJANDRA**", realizado por los Sres. Ronald David Macas Cando y Angel Mauricio Ramos Cepeda, para optar por el título de Ingeniero en Sistemas y Computación, considero que reúne los requisitos y méritos suficientes para ser sustentada públicamente y evaluada por el jurado examinador que se designe.

Riobamba, noviembre de 2021

Firmado electrónicamente por:
MIRYAN ESTELA
NARVAEZ VILEMA

……

Ing. Miryan Estela Narváez Vilema, PhD.
C.C. 0603576778
TUTORA

AUTORÍA DE LA INVESTIGACIÓN

La responsabilidad del contenido de este proyecto de investigación titulado, "**METODOLOGÍA TEST DRIVEN DEVELOPMENT APLICADA EN EL DESARROLLO DEL SISTEMA DE CONTROL DE INVENTARIO PARA EL ALMACÉN ALEJANDRA**" corresponde exclusivamente a Ronald David Macas Cando, Angel Mauricio Ramos Cepeda, Miryan Estela Narváez Vilema y el patrimonio intelectual de la misma a la Universidad Nacional de Chimborazo.

…….

Ronald David Macas Cando

C.C. 0605316843

……..

Angel Mauricio Ramos Cepeda

C.C. 0650256050

ÍNDICE GENERAL

ÍNDICE FIGURAS

ÍNDICE TABLAS

RESUMEN

Actualmente los sistemas de información se encuentran inmersos en todos los ámbitos de nuestras vidas, las pequeñas y medianas empresas no se quedan al margen de esta tendencia, sin embargo, hay muchas empresas que no cuentan con un sistema que permita administrar de forma eficiente y confiable toda la información. El presente trabajo contempla el estudio de la metodología Test Driven Development (TDD) o Desarrollo Guiado por Pruebas, cuyo objetivo principal es encontrar las ventajas de trabajar con esta técnica dentro de la construcción de un sistema. TDD es una metodología de desarrollo de software, que se basa en test o pruebas para guiar el proceso. Los resultados obtenidos demuestran que la técnica TDD inmersa en el Framework JUnit no simplemente abarca el testing de la aplicación, sino que conduce el óptimo y eficiente diseño de esta, mejorando el modelo de codificación de software y obteniendo un código más tolerante al cambio, robusto y seguro. Finalmente, después de conocer las ventajas de trabajar con TDD y conocer los principios de la metodología Scrum, se desarrolló e implementó un sistema informático de entorno desktop, utilizando la plataforma Java SE y gestor de base de datos MySQL. El sistema permitirá optimizar los procesos de compra, venta y control de productos del Almacén Alejandra.

Palabras clave: Metodología TDD, Framework JUnit, Inventario, Desarrollo guiado por pruebas, Metodología SCRUM

ABSTRACT

Currently with the rise of the Internet, information systems are immersed in all areas of our lives, small and medium-sized companies are not left out of this trend, however, the user is not involved as part of the work team nor is it gives importance to the automation of the source code which brings problems at the time of delivery of the software to the user. The present work contemplates an in-depth study on the Test-Guided Design Methodology whose main objective is to find the advantages of working with this technique within the construction of systems. After the analysis of the different testing paradigms, unit and integration tests were selected as a frame of reference for the elaboration of a comparative table with respect to TDD. The results obtained show that by using the TDD technique immersed in the JUnit Framework, it does not simply cover the testing of the application, but also leads to the optimal and efficient design of the application, improving the software coding model. In addition, during the execution of the tests, the failed tests and where the error is found for the respective refactoring are observed in the console. Finally, after learning about the advantages of working with TDD and knowing the principles of the Scrum methodology, a desktop environment computer system was developed and implemented using the Java SE platform, the MySQL DBMS, through which purchasing processes are optimized. sale and inventory control of the Alejandra Warehouse.

Keywords: TDD methodology, JUnit framework, inventory control, test-driven development, SCRUM methodology

Reviewed by:

Lcdo. Jhon Inca Guerrero.

ENGLISH PROFESSOR

C.C. 0604136572

INTRODUCCIÓN

Uno de los desafíos más grandes es garantizar la calidad de los productos de software y mejorar la productividad a lo largo del ciclo de vida, desde el comienzo del desarrollo hasta las etapas de mantenimiento. Este es un reto enorme porque actualmente los requerimientos son más volátiles que en el pasado, potencialmente se introducen muchos errores, ocasionando que funcionalidades ya implementadas empiecen a fallar, aparecen muchas fallas o errores, llamados “Bugs” en la etapa de mantenimiento. (Vaca et al., 2014)

El equipo de desarrollo de un proyecto de software por lo general le da más importancia a la arquitectura del software y a las nuevas tecnologías; sin embargo, muy pocos toman en cuenta la importancia de utilizar una adecuada metodología de trabajo. Tanto las metodologías tradicionales como las ágiles tienen la labor de guiar al desarrollador desde la fase de inicio hasta la fase de mantenimiento.

El desarrollo guiado por pruebas (TDD por sus siglas en inglés) es una técnica fundamental del ciclo de desarrollo que permite a los desarrolladores analizar, modificar y refactorizar el código fuente sin el temor a destruir las funcionalidades, debido a que cada una de ellas tiene asociado un test que debe cumplirse.

En el presente trabajo de investigación, se realizó una aplicación de escritorio de control de inventario para el Almacén Alejandra utilizando la metodología TDD, cuya mayor fortaleza es escribir la mínima cantidad de código posible para obtener el resultado deseado, el mismo que fue implementado a través del Framework Junit.

El presente documento está comprendido por V capítulos, que se detallan a continuación:

Capítulo I: Planteamiento del problema, en este capítulo se plantea el problema de la investigación, la justificación y los objetivos del proyecto.

Capítulo II: Marco Teórico, se realiza un estudio profundo del desarrollo guiado por pruebas (TDD) y temas relacionados con la temática.

Capítulo III: Metodología, se describe rápidamente el tipo de investigación, las técnicas de investigación y la metodología de desarrollo.

Capítulo IV: Desarrollo, aborda las actividades realizadas en el desarrollo del software de gestión de Inventarios para el Almacén Alejandra. Además, se detalla las pruebas unitarias utilizando TDD.

Capítulo V: Resultados y discusión, se describen los resultados obtenidos del trabajo de investigación.

Conclusiones y Recomendaciones, esta sección se describe las conclusiones obtenidas al finalizar el trabajo y las recomendaciones sugeridas para cada conclusión.

Para finalizar en la **Bibliografía** se detallan las fuentes bibliográficas consultadas en la elaboración de la tesis.

CAPITULO I

1. PLANTEAMIENTO DEL PROBLEMA

1.1. Problema de la investigación

El almacén Alejandra es una mediana empresa perteneciente al Cantón Guano, provincia de Chimborazo, en los últimos años ha crecido de forma significada posicionándose como una de las mejores del sector.

Actualmente debido a la cantidad de productos que tiene el almacén surge la necesidad de adaptarse hacia un entorno tecnológico, específicamente la implementación de un software que automatice las diferentes actividades diarias que se efectúan en la misma. Con la finalidad de solucionar la problemática antes expuesta, se desarrolló un sistema utilizando herramientas de software libre como: Framework Junit que permite desarrollar aplicaciones de escritorio mediante el lenguaje programación Java, uno de los más utilizados en el desarrollo de aplicaciones tanto desktop y web; para la persistencia de los datos se empleó el sistema gestor de bases de datos MySQL.

El éxito del desarrollo de un software depende de la selección de una metodología que se identifique con la lógica de negocio del cliente, por ello se ve la imperiosa necesidad de implementar la Metodología SCRUM conjuntamente con TDD, permitiendo producir un software de manera iterativa e incremental, con un estricto control sobre los procesos, y favoreciendo la detección temprana de errores y mejorando el diseño.

1.2. Justificación

La investigación sobre la metodología TDD en aplicaciones de escritorio desarrolladas mediante Java, MySQL y Junit, permite escribir líneas de código necesarias para poder anticipar, identificar y corregir fallas durante la creación del software.

Las pruebas escritas en java, permiten que se ejecuten automáticamente las veces que sea necesario, al ejecutar las pruebas se comprueba si su ejecución fue correcta o no, también se mantiene la calidad de la arquitectura del software, se cambia el diseño sin cambiar la funcionalidad y sobre todo se conservan las pruebas unitarias previamente implementadas.

1.3. Objetivos

1.3.1. Objetivo General

Aplicar la metodología Test Driven Development en el desarrollo del sistema de control de inventario para el almacén Alejandra.

1.3.2. Objetivos específicos

- Investigar la metodología de desarrollo Test Driven Development.
- Implementar la metodología de desarrollo de pruebas unitarias Test Driven Development en el caso práctico.
- Desarrollar el sistema de control de inventario para el Almacén Alejandra utilizando el código refactorizado.

CAPÍTULO II

2. MARCO TEÓRICO

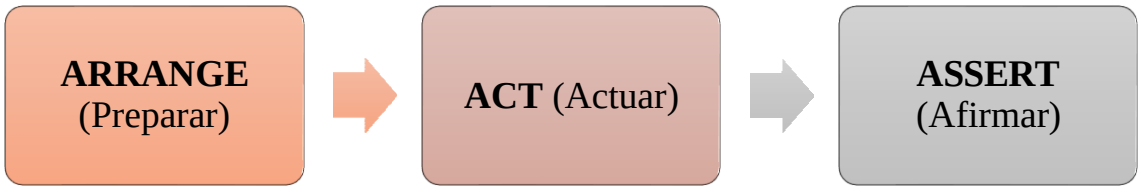
2.1. Introducción a las pruebas de software

Para los autores Lee (2020), Orozco (2018) y Torres (2017) las pruebas de software son una parte integral durante el periodo de desarrollo, a través de ellas el desarrollador puede estar seguro de las funcionalidades, el rendimiento y la experiencia del usuario. Estas pruebas pueden realizarse manualmente o a través de procesos automatizados, cuanto antes y más a menudo pueda llevar a cabo las pruebas, más probable es que identifique errores, garantizando que el software ha sido revisado y auditado a fondo, antes de que esté frente a los usuarios.

2.2. Tests de desarrollo

Un test generalmente debe cumplir con la estructura AAA (siglas en inglés):

Figura 1. Estructura de un test de desarrollo



Fuente: Elaboración propia

A continuación, se presenta la definición de cada término de la Figura 1, según Microsoft Developer Network (2016):

- **Arrange:** consiste en establecer los valores de los datos que se va a utilizar en las pruebas.
- **Act:** llama al método a probar con los parámetros preparados para tal fin.
- **Assert:** comprueba que el método ejecutado se comporta tal y como se tiene previsto.

El Framework Junit dispone de métodos específicos para realizar comprobaciones. (Veintimilla & Cuenca, 2014). En la Tabla 1, se describen cada uno de ellos.

Tabla 1. Métodos de JUnit para hacer comprobaciones

Método ASSERT () JUnit	**DESCRIPCIÓN**
assertTrue(expresión)	Comprueba que expresión evalúe a true
assertFalse(expresión)	Comprueba que expresión evalúe a false
assertEquals(esperado, real)	Comprueba que esperado sea igual a real
Assert Null(Objeto)	Comprueba que objeto sea null
Assert NotNull(Objeto)	Comprueba que objeto no sea null
AssertSame(Objeto_esperado, objetivo_real)	Comprueba que objeto_esperado y objeto_real sean el mismo objeto
Assert NotSame(Objeto_esperado, objetivo real)	Comprueba que el objetivo_esperado no sea el mismo objetivo que el objetivo_real
fail()	Hace que el test termine con fallo

Fuente: Elaboración propia

2.2.1. Test Unitarios

Este tipo de test específicamente se encargan de probar pequeñas partes del software de manera independiente. Se caracterizan por ser atómicas, independientes, inocuas y rápidas.

- **Atómico:** el test prueba la mínima cantidad de funcionalidad posible. Es decir, probará un solo comportamiento de un método de una clase. El mismo método puede presentar distintas respuestas ante distintas entradas o distinto contexto.
- **Independiente:** un test no puede depender de otros para producir un resultado satisfactorio. No puede ser parte de una secuencia de pruebas que se deba ejecutar en un determinado orden. Debe funcionar siempre igual independientemente de que se ejecuten otros tests o no.
- **Inocuo:** no altera el estado del sistema. Al ejecutarlo una vez, produce exactamente el mismo resultado que al ejecutarlo veinte veces. No altera la base de datos, ni envía emails ni crea ficheros, ni los borra. Es como si no se hubiera ejecutado.
- **Rápido:** se ejecuta un gran número de pruebas en pocos minutos y se ha demostrado que tener que esperar unos cuantos segundos cada rato, resulta muy improductivo. Un sólo test tendría que ejecutarse en una pequeña fracción de segundo.

2.2.2. Test de integración

Son los encargados de testear la integración entre dos o más componentes, para lograr este objetivo utilizan dependencias o datos reales, siempre y cuando no se afecte el estado del software antes de ejecutar el test.

2.2.3. Test del sistema

Moisés (2019) afirma que el objetivo de este test es realizar las pruebas al sistema, manipulándolo como si se tratara de un usuario final, iniciando en la interfaz gráfica hasta llegar al almacén de datos.

Blé (2020), en cambio asegura que en un proyecto con TDD, no es productivo escribir pruebas para todas las posibles formas de uso del sistema, pues bien, esta redundancia se traduce en el aumento del costo de mantenimiento de los tests.

2.3. Test Driven Development (TDD)

TDD es una metodología de diseño e implementación de software guiado por pruebas, creado a partir de la metodología ágil XP, por tanto, se considera una técnica de programación de software que consiste en dos procesos prácticas, 1) escribir una prueba y 2) refactorizar el código. (Haq, 2017)

El desarrollo orientado por pruebas (TDD), es una de las metodologías con mayor acogida en el campo laboral y profesional, además continúa expandiéndose debido a sus buenos resultados. La tendencia es integrar TDD independientemente de cualquier metodología ágil o tradicional y aprovechar los beneficios (deshacer los errores), asegurando la calidad del producto y protegerse de errores tanto malintencionados como humanos. (Amaya, 2013)

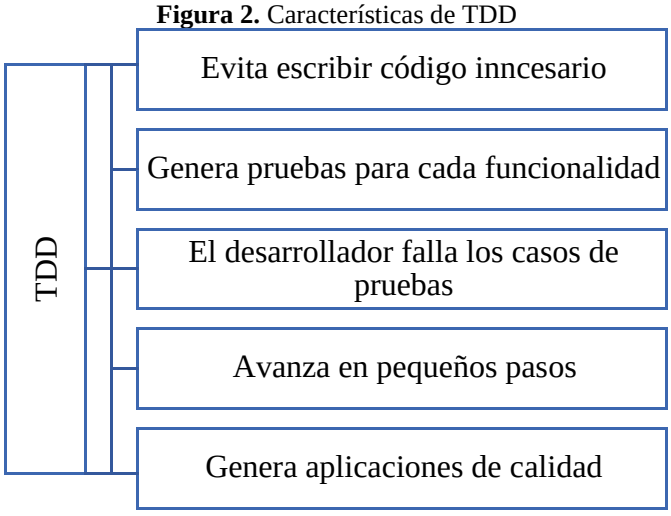
A través de la metodología TDD se pueden aplicar modelos de calidad, además, es considerada como una metodología de diseño con un proceso extenso, por esta razón, para alcanzar el objetivo TDD se debe crear hábitos en el desarrollador de software, esto permitirá alcanzar mejores resultados en los procedimientos. Mediante esta técnica se busca que el desarrollador entienda la lógica de negocio como fase inicial antes de escribir cualquier línea de código, sin tener una idea clara del objetivo que persigue el proyecto, logrando con esta premisa, obtener código limpio pero que funciona.

La metodología TDD tranquilamente puede ser utilizada en proyectos de diferente tamaño, un proyecto grande está dividido por varios subproyectos, para efecto resulta beneficioso trabajar con esta técnica como si fueran proyectos pequeños que posteriormente serán integrados para realizar la entrega del producto. TDD permite optimizar procesos, garantizando una importante mejoría en el rendimiento del equipo desarrollador, debido a que entrega como resultado final un producto de software robusto, mantenible y de calidad. Para ejecutar de manera correcta la metodología de trabajo TDD, se puede utilizar una metodología de desarrollo de software ágil como SCRUM o Programación Extrema, resulta muy beneficioso, debido a que, se puede trabajar correctamente con el listado de tareas que van a ser utilizadas para brindar la solución al proyecto. (Ress et al, 2013)

2.3.1. Principales características

Según Ress (2013) las principales características de TDD son:

Figura 2. Características de TDD



Fuente: Elaboración propia a partir de (Ress et al., 2013)

- **Evita escribir código innecesario**, escribir el mínimo posible, y si el código implementado pasa una prueba y falla, da una idea de cómo modificar los requerimientos, agregando nuevos.
- **Genera pruebas para cada funcionalidad,** permite que el desarrollador confíe en el código escrito. Esto facilita futuras modificaciones, porque si se logra pasar todas las pruebas se obtendrá un código que funcione correctamente.

- **Requiere que el desarrollador haga fallar los casos de prueba**, se intenta asegurar el funcionamiento de los casos de prueba y de esta forma detectar errores.
- **Proporciona un valor agregado** en el desarrollo de software, produciendo aplicaciones de calidad y en menos tiempo.
- **Tiene la capacidad de avanzar en pequeños pasos**, permite al desarrollador enfocarse en la tarea actual y hacer que supere la prueba.

2.3.2. Beneficios de TDD

Desde el punto de vista de Adewole (2018) trabajar con TDD presenta seis beneficios.

- Facilita el desarrollo de pruebas.
- Produce aplicaciones de software robustas con menos o ningún error.
- Las pruebas son una documentación en miniatura del código fuente, sirven como una forma rápida de comprender cómo funciona una parte del código.
- La estructura de una aplicación se puede entender fácilmente a partir de la prueba, habrá pruebas para todos los objetos, así como pruebas para los métodos de los objetos, mostrando su uso.
- Ayuda a escribir código elegante con buena abstracción, diseño flexible y arquitectura.
- El código se puede mantener fácilmente, es legible y tiene pruebas escritas para validar su comportamiento consistente de manera apropiada.

2.3.3. Ciclo TDD

Beck (2002) propone los seis pasos para el ciclo TDD.

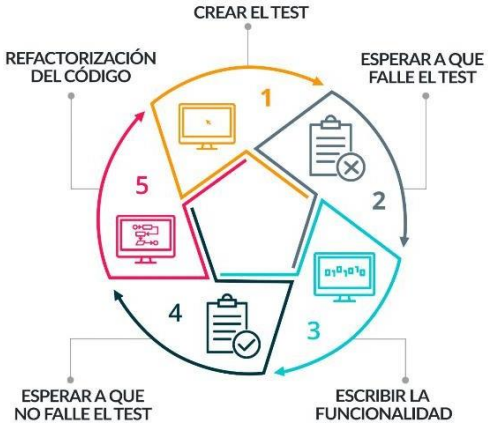
1. Crear test: escribir una prueba con el código más simple posible.
2. Ejecutar la prueba y comprobar que la misma falla.
3. Escribir código: escribir una nueva porción de código que brinde solución momentánea a la prueba unitaria.
4. Ejecutar las pruebas: automatizar las pruebas con el fin de verificar que todas las pruebas continúen al siguiente paso.

5. Refactorizar: Mejorar el código con el fin de eliminar código duplicado.

6. Repetir: Iniciar nuevamente con otra prueba unitaria, no obstante, se debe mencionar que este proceso se repite una y otra vez hasta finalizar el desarrollo del producto software.

En la Figura 3, se presentan gráficamente los pasos del ciclo TDD.

Figura 3. Ciclo de vida TDD



Fuente: (Beck, 2002)

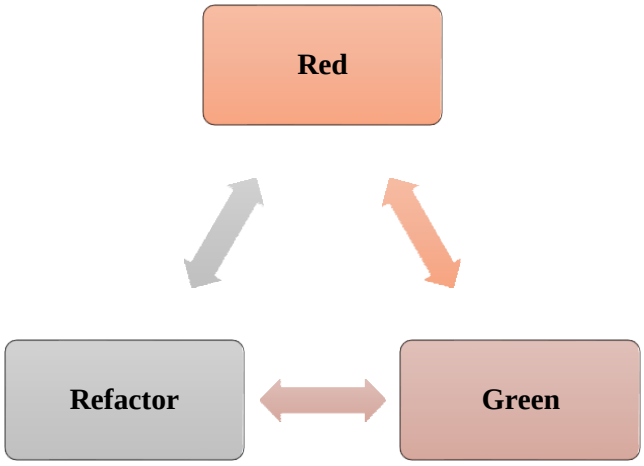
2.3.4. Ciclo Red-Green-Refactor (RGR)

Test Driven Development es un proceso que se basa en la repetición de un ciclo de desarrollo muy corto. Se basa en el primer concepto de *extreme programming* (XP) que fomenta el diseño simple con un significativo nivel de confianza. El procedimiento que impulsa este ciclo se llama *red-green-refactor*. (Farcic & García, 2015)

Castro (2016) afirma que el ciclo RGR se deriva del algoritmo TDD y comprende un proceso con tres estados.

1. **Red** crear una prueba que falle.
2. **Green** indica dar solución a la prueba creada, escribiendo cierta cantidad de código temporal.
3. **Refactor** elimina la duplicidad de código fuente, si existiera.

Figura 4. Ciclo RGR



Fuente: (Castro, 2016)

La Figura 4, presenta un marco que los desarrolladores usan para construir un conjunto de pruebas, escribir código de implementación y optimizar su base de código en ciclos de desarrollo cortos. La finalidad de los pasos mencionados es, que el desarrollo está intentando dirigirse hacia una arquitectura limpia, independiente de organismos externos. (Gałęzowski, 2016)

2.4. Metodología SCRUM

Es una metodología ágil usada para el desarrollo de software de forma iterativa e incremental; iterativa por que se desarrolla en bloques de tiempos cortos y fijos, reciben el nombre de Sprints; e incremental por que se obtienen funcionalidades del producto final al terminar cada iteración. (Salazar et al., 2018)

2.4.1. Beneficios y componentes de SCRUM

Según Laínez (2015) los beneficios son:

- Comunicación.
- Trabajo en equipo.
- Flexibilidad.
- Proveer un software funcional de manera incremental.

Los componentes son:

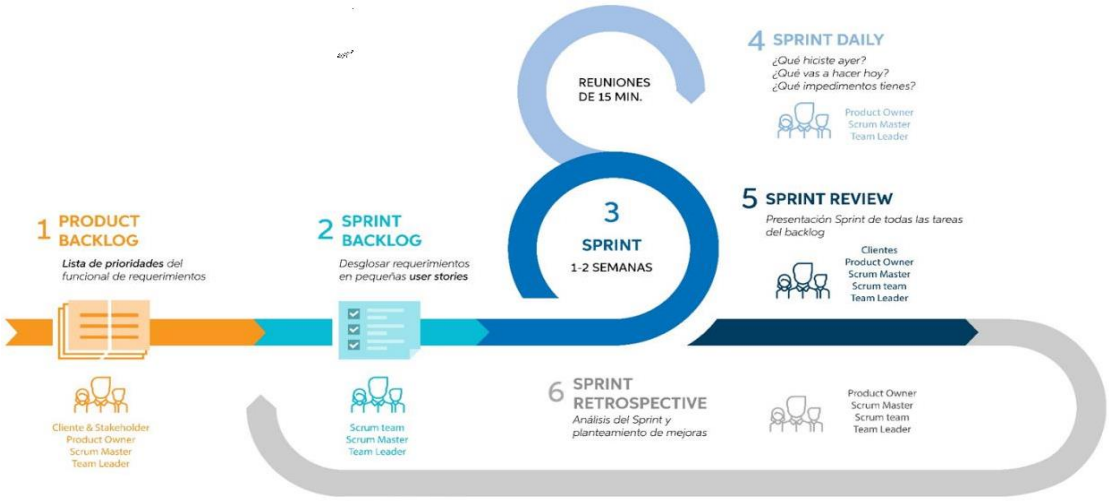
- Backlog.
- Equipos de desarrollo.
- Sprints.
- Reuniones diarias.
- Reuniones de revisiones.
- Presentación de demos.

2.4.2. Roles de la metodología SCRUM

- Product Owner: representa al cliente, responsable de optimizar y maximizar el trabajo, determina los objetivos del proyecto con el propósito de garantizar que el equipo trabaje de manera adecuada para poder lograr los objetivos planteados.
- Scrum Máster: persona que lidera el equipo guiándolo para que no tenga ninguna dificultad para cumplir con las tareas y funciones, además permite que el equipo se mantenga activo y productivo.
- Scrum Team: se encargan de desarrolla y entrega el producto. Su trabajo es imprescindible, y presenta una estructura horizontal autoorganizada capaz de autogestionarse a sí misma.

2.4.3. Ciclo de vida SCRUM

Figura 5. Ciclo de vida Scrum



Fuente: (Salazar et al., 2018)

La metodología SCRUM es un framework adaptable, iterativo, rápido, flexible y eficaz, diseñado para ofrecer un valor considerable en forma rápida a lo largo del proyecto. SCRUM garantiza la transparencia en la comunicación asertiva y crea un ambiente de responsabilidad colectiva y de progreso continuo.

2.5. Herramientas de desarrollo

2.5.1. Java

Es un lenguaje de programación de alto nivel orientado a objetos de reciente creación. Fue desarrollado por la empresa Sun Microsystems a principios de los años 90, y presentado oficialmente en mayo de 1995 en la conferencia Sun World. (Garrido, 2015)

Sznajdleder (2013) y Sznajdleder (2016) afirma que Java es un lenguaje de programación de propósitos generales, se puede utilizar para desarrollar el mismo tipo de aplicaciones que se programa con otros lenguajes como: C, VB, PHP, JavaScript o Pascal.

Tabla 2. Características Java

Característica	Descripción
Familiar	Facilita la migración de aquellos desarrolladores ya familiarizados con lenguajes como C o C++.
Sencillo	Puede tener una curva de aprendizaje dura, el conocer otros lenguajes de programación similares facilita su aprendizaje.
Multiplataforma	Fue diseñado para ser (WORA), es decir, escribir y compilar una vez en una plataforma y ejecutar en cualquier otra, sin necesidad de modificar el código fuente, ni siquiera de recompilar.
Alto rendimiento	No son tan rápidos como los desarrollados con otros lenguajes que se compilan de forma nativa para una plataforma concreta, sin embargo, tampoco son tan lentos como los lenguajes que son exclusivamente interpretativos.
Robusto	Durante la compilación se comprueba la sintaxis y ciertas situaciones que en otros lenguajes no, pudiendo dar resultados inesperados. Posteriormente durante la ejecución se vuelve a dar otras comprobaciones, por lo que resulta un poco robusto el programa.
Orientado a objetos	Sigue este paradigma de programación, que es el más utilizado en la actualidad.
Distribuido	Dispone de una librería de clases que permite la comunicación entre programas ejecutados en ordenadores remotos conectados por red.
Concurrente	Permite el desarrollo de aplicaciones concurrentes o multi-hilo para conseguir un mejor rendimiento.

Fuente: Elaboración propia a partir de (Sznajdleder P. , 2013), (Sznajdleder P. A., 2016) y (Garrido, 2015)

2.5.2. MySQL

Es un sistema gestor de bases de datos (SGBD, DBMS por sus siglas en inglés) muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas, disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales (no menos importantes) contar con un alto grado de estabilidad y un rápido desarrollo. (Casillas, Ginestà, & Pérez, 2014)

Tabla 3. Ventajas y desventajas de MySQL

Ventajas	Desventajas
MySQL es de uso libre y gratuito.	Al ser de Software Libre, muchas de las soluciones para las deficiencias del software no están documentados ni presentan documentación oficial.
Software con Licencia GPL.	Muchas de sus utilidades tampoco presentan documentación.
Entorno con seguridad y encriptación.	No es el más intuitivo de los programas que existen actualmente para todos los tipos de desarrollos.

Fuente: Elaboración propia a partir de (Casillas, Ginestà, & Pérez, 2014)

2.5.3. JUnit

Es un framework de código abierto, creado y desarrollado por Erich Gamma y Kent Beck, para la creación de pruebas unitarias en el lenguaje Java. Su propósito es servir de base para la creación de código de automatización de pruebas. Es muy utilizado para la práctica de TDD y su mismo modelo se utilizó en la creación de frameworks de prueba para otros lenguajes, a estos frameworks generalmente se los conoce como XUnit. (Pozo et al., 2011)

Para Guerra & Lema (2018) las características principales de dichos marcos son la ejecución de casos de prueba y la visualización de resultados de ejecución se encuentran definidos en la Tabla 4.

Tabla 4. Anotaciones del framework JUnit

Anotación	Descripción
@Test public void method()	La anotación @Test identifica un método como método de prueba
@Test (expected = Exception.class)	Falla si el método no lanza la excepción nombrada
@Test(timeout=100)	Falla si el método tarda más de 100 milisegundos
@Before public void method()	Este método se ejecuta antes de cada prueba. Se utiliza para preparar la prueba. entorno (por ejemplo, leer datos de entrada, inicializar la clase)
@After public void method()	Este método se ejecuta después de cada prueba. Se usa para limpiar la prueba. entorno (por ejemplo, eliminar datos temporales, restaurar valores predeterminados). También puede ahorre memoria limpiando costosas estructuras de memoria.
@BeforeClass public static void method()	Este método se ejecuta una vez, antes del inicio de todas las pruebas. Es usado para realizar actividades que requieren mucho tiempo, por ejemplo, para conectarse a una base de datos. Los métodos marcados con esta anotación deben definirse como estáticos para trabajar con JUnit.
@AfterClass public static void method()	Este método se ejecuta una vez, una vez finalizadas todas las pruebas. Esta usado para realizar actividades de limpieza, por ejemplo, para desconectarse de una base de datos. Los métodos anotados con esta anotación deben definirse como estáticos para trabajar con JUnit.
@Ignore or @Ignore("Why disabled")	Este método se ejecuta una vez, una vez finalizadas todas las pruebas. Esta usado para realizar actividades de limpieza, por ejemplo, para desconectarse de una base de datos. Los métodos anotados con esta anotación deben definirse como estáticos para trabajar con JUnit.

Fuente: Elaboración propia a partir de (Guerra & Lema, 2018)

CAPÍTULO III

3. METODOLOGIA

La investigación ha sido abordada desde el enfoque cuantitativo y de tipo de investigación cuasi experimental. Tiene como atención primordial utilizar la metodología Test Driven Development, para evaluar la calidad del código de las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos. El código debe cumplir con una buena legibilidad, debe estar ausente de duplicaciones, redundancias y conservar un alto nivel de mantenibilidad. Su principal objetivo es obtener código limpio que trabaje.

3.1. Identificación de las Variables

3.1.1. Variable independiente

Metodología Test Driven Development.

3.1.2. Variable dependiente

Sistema de gestión y control de inventario.

3.2. Tipo de investigación

Aplicada: Debido a que la investigación busca dar solución a un problema en concreto, con el objeto de encontrar soluciones o respuestas que puedan aplicarse, en este caso optimizar el proceso de control de inventario del almacén Alejandra.

Explicativa: Se aplica porque se requiere partir desde el origen del problema, aquí se plantea una relación causa-efecto, se establece la relación causal, es decir el análisis de las causas y efectos que se ha generado el no llevar un sistema de control de inventarios.

3.3. Técnicas de investigación

Para el desarrollo del trabajo de investigación, se utilizarán 3 técnicas: la observación, entrevista y análisis documental.

- **Observación. –** permitió identificar los procesos de compra, venta y control de productos que actualmente maneja el Almacén Alejandra.

- **La entrevista. –** comunicación interpersonal establecida entre los desarrolladores y el dueño de la empresa, a fin de obtener respuestas verbales a los interrogantes planteados sobre el problema propuesto, la entrevista estuvo enfocada en definir requerimientos e identificar el funcionamiento de cada uno de los procesos.
- **En el análisis documental. –** es un procedimiento sistemático para revisar documentos dentro de una perspectiva de estudio. El análisis documental estará enfocado en la revisión de literatura acerca de la metodología TDD.

3.4. Operacionalización de variables

Tabla 5. Matriz de Consistencia

Variable	Tipo	Descripción	Dimensión	Indicadores
Metodología Test Driven Development	Independiente	El desarrollo basado en pruebas es una técnica para crear software de forma incremental, en pocas palabras, no se escribe ningún código de producción sin antes escribir una prueba unitaria fallida. (Grenning, 2011)	Metodología	• Pruebas Unitarias. • Líneas de código. • JUnit fases: 1. Rojo – pruebas fallidas. 2. Amarillo – refactorización de código. 3. Verde – pruebas aceptadas.
Sistema de gestión y control de inventario	Dependiente	Es una herramienta de gestión, empleada para registrar las cantidades de mercancías existentes en un negocio, así como para determinar el costo de los productos vendidos. (Chile, 2019)	Software	• Número de módulos generados. • Utilizando el código refactorizado.

Fuente: Elaboración propia

3.5. Metodología de desarrollo ágil de software SCRUM

En este apartado se describen el desarrollo del sistema de inventarios, los modelos de ingeniería del software, prototipos, implementación de la metodología de desarrollo orientado por pruebas (TDD), la construcción de la aplicación y la evaluación de calidad de la aplicación. En el presente proyecto se consideran tres fases: planificación, desarrollo y cierre, las mismas que se describen en los siguientes apartados.

3.5.1. Fase de planificación

En esta fase se realiza una planificación detallada de las principales actividades a seguir en el proceso de desarrollo de software, entre las cuales constan: recolección de información, selección de herramientas, pruebas TDD, implementación del sistema y capacitación de usuarios.

3.5.1.1. Personal involucrado en el proyecto

La metodología Scrum establece los siguientes roles:

Scrum Máster es la persona que se encargará de coordinar el equipo y asignar las tareas a realizar.

Team se denomina al grupo de trabajo, son las personas que están a cargo de desarrollar el código del sistema, a su vez, para dar cumplimento deben seguir la planificación y realizar las historias de usuario de cada funcionalidad, que fueron establecidas en cada Sprint. El usuario es el destinatario final quien solicito el sistema.

Tabla 6. Personal involucrado en el desarrollo del aplicativo

Rol	Persona	Contactos
Product Owner	Gerente del Almacén Alejandra	danielasatan17@gmail.com
Scrum Master	Ing. Miryan Estela Narváez Vilema Tutor de Tesis	miryan.narvaez@unach.edu.ec
Team	Ronald David Macas Cando Ángel Mauricio Ramos Cepeda	

Fuente: Elaboración propia

Roles del sistema

La Tabla 7, describe los tipos de roles para los usuarios.

Tabla 7. Roles del sistema

Rol	Descripción	Responsabilidades
Administrador	Persona responsable de administrar sistema.	Gestiona el personal de la empresa, compras, ventas, administrar el almacén, realizar el control de productos y generar informes, reportes gráficos y en formato pdf.
Vendedor	Persona que se encarga del área de ventas.	Gestionar las ventas.

Fuente: Elaboración propia

3.5.1.2. Pila del producto (Product Backlog)

Product Backlog se utiliza para listar las funcionalidades determinadas, en conjunto con el Product Owner luego de realizar varias reuniones planificadas. Con esta herramienta se logra dar una estimación y prioridad a los requerimientos planteados de acuerdo con las necesidades requeridas por el negocio. Esta herramienta utiliza una tarjeta de trabajo conocida como historia de usuario.

3.5.1.3. Product Backlog

Una vez definido los requerimientos de acuerdo con su prioridad y tomando en cuenta la complejidad de cada uno, se emplea la especificación de requerimientos, permitiendo transformar a historias de usuario. En las reuniones efectuadas con el cliente y analizado los resultados se obtuvo treinta y siete requerimientos, de los cuales se tiene; ocho historias técnicas y veinte nueve historias de usuario. La duración del proyecto tiene 6 meses desde la primera reunión con el dueño del local hasta llegar a la implantación del sistema.

La Tabla 8, detalla las historias de usuario e historias técnicas utilizadas en desarrollo del sistema**.**

Tabla 8. Historias de usuario y técnicas

Código	Descripción	Esfuerzo
HT-01	Establecer los requerimientos del sistema	18
HT-02	Establecer la arquitectura del sistema	18
HT-03	Diseño de Estándar de Codificación	6

HT-04	Diseño del estándar de interfaz	18
HT-05	Realizar la Base Datos (Modelo entidad relación, lógico, físico, script diccionario de datos)	12
HU-01	Ingresar categoría	12
HU-02	Modificar categoría	12
HU-03	Agregar vigencia de categoría	12
HU-04	Gestión productos	12
HU-05	Agregar productos	12
HU-06	Modificar productos	12
HU-07	Buscar para el módulo almacén	12
HU-08	Establecer vigencia de producto	12
HU-09	Reportes para módulo almacén	12
HU-10	Crear sesión de usuarios	18
HU-11	Gestionar roles	12
HU-12	Asignar funcionalidad de acuerdo con el rol de usuario	12
HU-14	Buscar y visualizar usuarios empleados de la empresa	6
HU-15	Gestionar proveedores	12
HU-16	Gestionar compras a proveedores	12
HU-17	Consultar compras por fecha, mes	12
HU-18	Buscar proveedores	6
HU-19	Historial de precios	6
HU-20	Reportes para el módulo proveedores	12
HU-21	Gestionar clientes	12
HU-22	Gestionar venta	24
HU-23	Consultar ventas por día, fecha, mes	12
HU-24	Reportes para el módulo ventas	12
HU-25	Administrar caja	24
Hu-26	Historial de caja	12
HU-27	Generar Inventario	24

HU-28	Resumen de saldos y movimiento de productos.	30
HU-29	Graficas estadísticas de compras y ventas.	30
HT-07	Capacitación de usuarios	18
HT-08	Documentación final del Sistema	24

Fuente: Elaboración propia

Con las historias de usuario descritas en el Product Backlog, se realiza varias acciones como: ingresar, modificar, listar, buscar, reportes y visualización de gráficos estadísticos. Todas estas se encuentran distribuidas en los Sprint.

3.5.1.4. Sprint Backlog

Con el objetivo de establecer un cronograma de actividades del proyecto y que además permita coordinar con el usuario y el personal encargado las fechas de los entregables del sistema, se realiza una distribución de Sprint teniendo como fecha de inicio 19 de abril y fecha de finalización 7 de septiembre de 2021, detalladas en la Tabla 9.

Tabla 9. Sprint Backlog

ID	Descripción	Fecha Inicio	Fecha Fin	Esfuerzo
SPRINT 1		**19/04/21**	**04/05/21**	**60**
HT-01	Establecer los requerimientos del sistema	19/04/21	23/04/21	18
HT-02	Establecer la arquitectura del sistema	23/04/21	27/04/21	18
HT-03	Diseño de Estándar de Codificación	27/04/21	01/05/21	6
HT-04	Diseño del estándar de interfaz	01/05/21	04/05/21	18
SPRINT 2		**04/05/21**	**19/05/21**	**60**
HT-05	Realizar la Base Datos (Modelo entidad relación, lógico, físico, script diccionario de datos)	04/05/21	07/05/21	12
HU-01	Ingresar categoría	07/05/21	10/05/21	12
HU-02	Modificar categoría	10/05/21	13/05/21	12
HU-03	Agregar vigencia de categoría	13/05/21	16/05/21	12
HU-04	Gestión productos	16/05/21	19/05/21	12
SPRINT 3		**19/05/21**	**03/06/21**	**60**
HU-05	Agregar productos	19/05/21	22/05/21	12
HU-06	Modificar productos	22/05/21	25/05/21	12
HU-07	Buscar para el módulo almacén	25/05/21	28/05/21	12
HU-08	Establecer vigencia de producto	28/05/21	31/05/21	12
HU-09	Reportes para modulo almacén	31/05/21	03/06/21	12
SPRINT 4		**03/06/21**	**18/06/21**	**60**
HU-10	Crear sesión de usuarios	03/06/21	06/06/21	18
HU-11	Gestionar roles	06/06/21	09/06/21	12
HU-12	Asignar funcionalidad de acuerdo con el rol de usuario	09/06/21	12/06/21	12
HU-13	Activar estado de vigencia de la funcionalidad asignada	12/06/21	15/06/21	12
HU-14	Buscar y visualizar usuarios empleados de la empresa	15/06/21	18/06/21	6
SPRINT 5		**18/06/21**	**03/07/21**	**60**
HU-15	Gestionar proveedores	18/06/21	21/06/21	12
HU-16	Gestionar compras a proveedores.	21/06/21	23/06/21	12
HU-17	Consultar compras por fecha, mes.	23/06/21	25/06/21	12

ID	Descripción	Fecha Inicio	Fecha Fin	Esfuerzo
HU-18	Buscar proveedores	25/06/21	28/06/21	6
HU-19	Historial de precios	28/06/21	30/06/21	6
HU-20	Reportes para el módulo proveedores	30/06/21	03/07/21	12
SPRINT 6		**03/07/21**	**18/07/21**	**60**
HU-21	Gestionar clientes	03/07/21	07/07/21	12
HU-22	Gestionar venta	07/07/21	10/07/21	24
HU-23	Consultar ventas por día, fecha, mes	10/07/21	14/07/21	12
HU-24	Reportes para el módulo ventas	14/07/21	18/07/21	12
SPRINT 7		**18/07/21**	**02/08/21**	**60**
HU-25	Administrar caja	18/07/21	23/07/21	24
HU-26	Historial de caja	23/07/21	28/07/21	12
HU-27	Generar Inventario	28/07/21	02/08/21	24
SPRINT 8		**02/08/21**	**17/08/21**	**60**
HU-28	Resumen de saldos y movimiento de productos	02/08/21	10/08/21	30
HU-29	Graficas estadísticas de compras y ventas	10/08/21	17/08/21	30
SPRINT 9		**17/08/21**	**01/09/21**	**60**
HT-07	Capacitación de usuarios	17/08/21	22/08/21	18
HT-08	Documentación final del Sistema	22/08/21	01/09/21	24

Fuente: Elaboración propia

Cada 10 días laborables se realiza una reunión con el director del proyecto, donde se realiza la entrega del avance.

CAPITULO IV

4. DESARROLLO

4.1. Fase de desarrollo

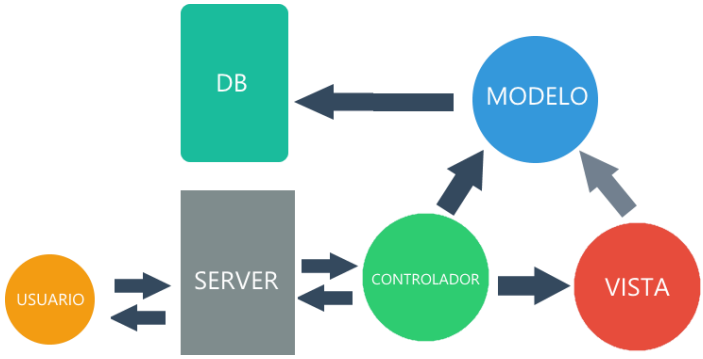
En esta fase se describen las actividades realizadas para el desarrollo del sistema de inventarios.

4.1.1. Arquitectura del sistema

La arquitectura en la cual se desarrolla el sistema de compra y venta se denomina cliente – servidor, utilizando el patrón Modelo, Vista, Controlador (MVC); para el almacenamiento de la información se utiliza el sistema gestor de base de datos MySQL.

El patrón MVC tiene como objetivo separar el código fuente en tres capas diferentes tal como se aprecia en la Figura 6.

Figura 6. Arquitectura de la aplicación



Fuente: Elaboración propia

4.1.2. Estándar de codificación

Con el objetivo de lograr una forma de programación uniforme, entendible que facilite el mantenimiento del sistema, se utiliza el estilo de escritura Camel-Case. Notación que usa la primera palabra en minúscula y las demás en mayúscula. En la Tabla 10, se describen las características de los componentes a utilizar.

Tabla 10. Características de los componentes utilizados

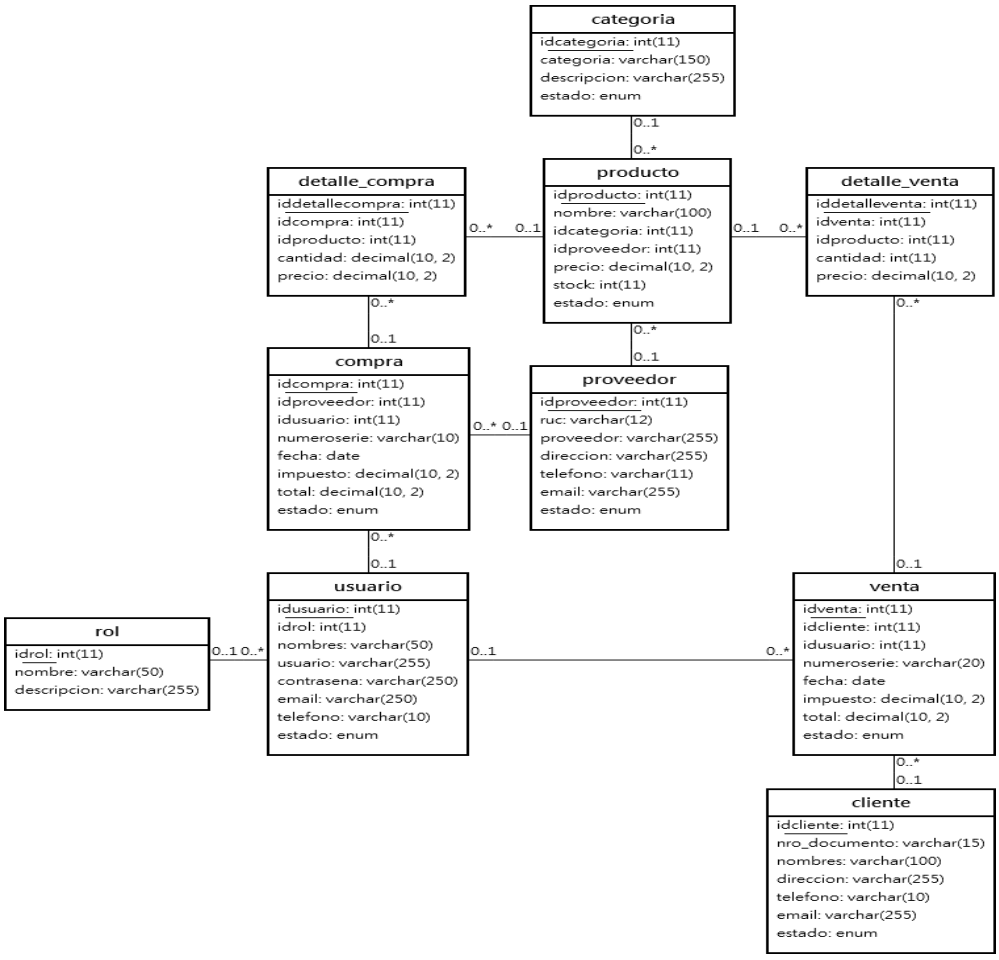
Componente	Nombre	Descripción
Clases	categoriaController	Nombre de la clase y luego nombre del módulo al que pertenece.
Variables	String categoría	Tipo de variable y luego asignar un nombre al atributo
Métodos	categoriaBuscar ()	Nombre de la clase seguido del nombre de método.

Fuente: Elaboración propia

4.1.3. Modelo de base de datos

El modelo conceptual del diseño de la base de datos define las entidades, atributos y relaciones que determinan el almacenamiento de la información, se muestra el modelo en la Figura 7.

Figura 7. Modelo físico de base de datos

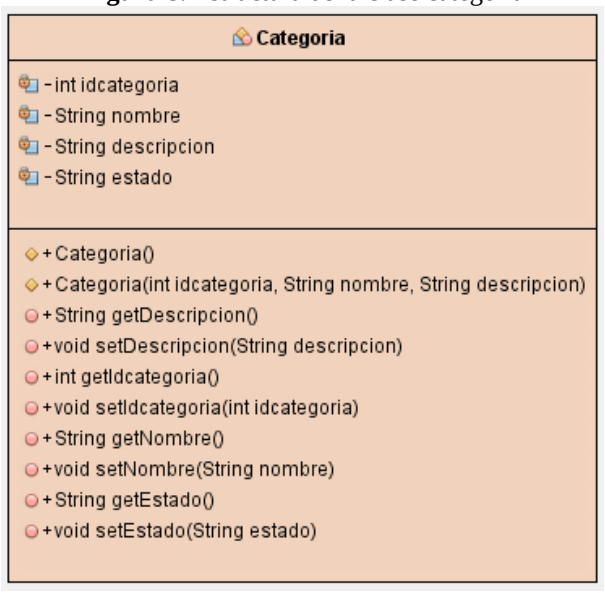


Fuente: Elaboración propia

4.1.4. Diagramas de clases

Es un modelo importante en la ingeniería de software, debido a que, se plasman las clases, atributos, métodos y relaciones que definen en su totalidad las características que debe tener las fuentes del proyecto, se presentan las clases con sus respectivos métodos, en la Figura 8.

Figura 8. Estructura de la clase categoría

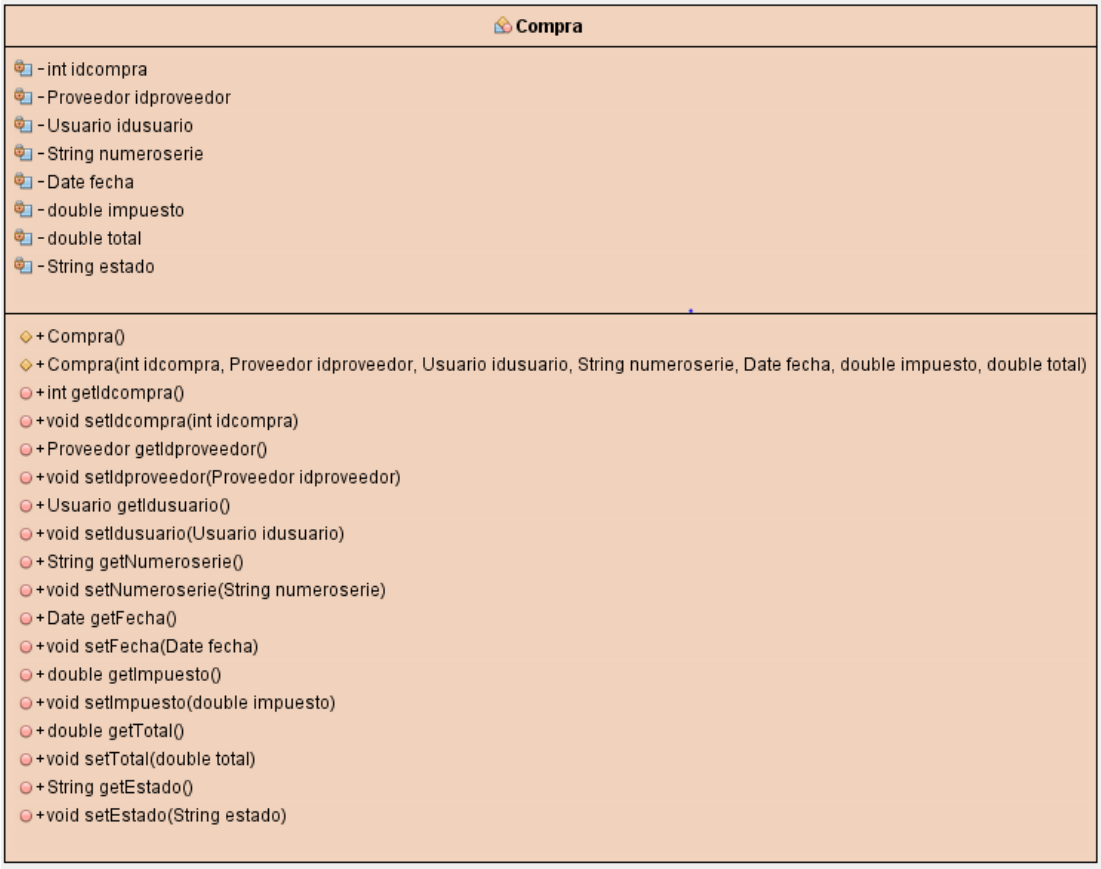


Fuente: Elaboración propia

Figura 9. Estructura de la clase Cliente

Fuente: Elaboración propia

Figura 10. Estructura de la clase Compra



Fuente: Elaboración propia

El modelo permite diseñar un bosquejo claro de las entidades, atributos y métodos con los que cuenta el sistema. Los métodos que se definen en cada una de las clases son los necesarios para un CRUD.

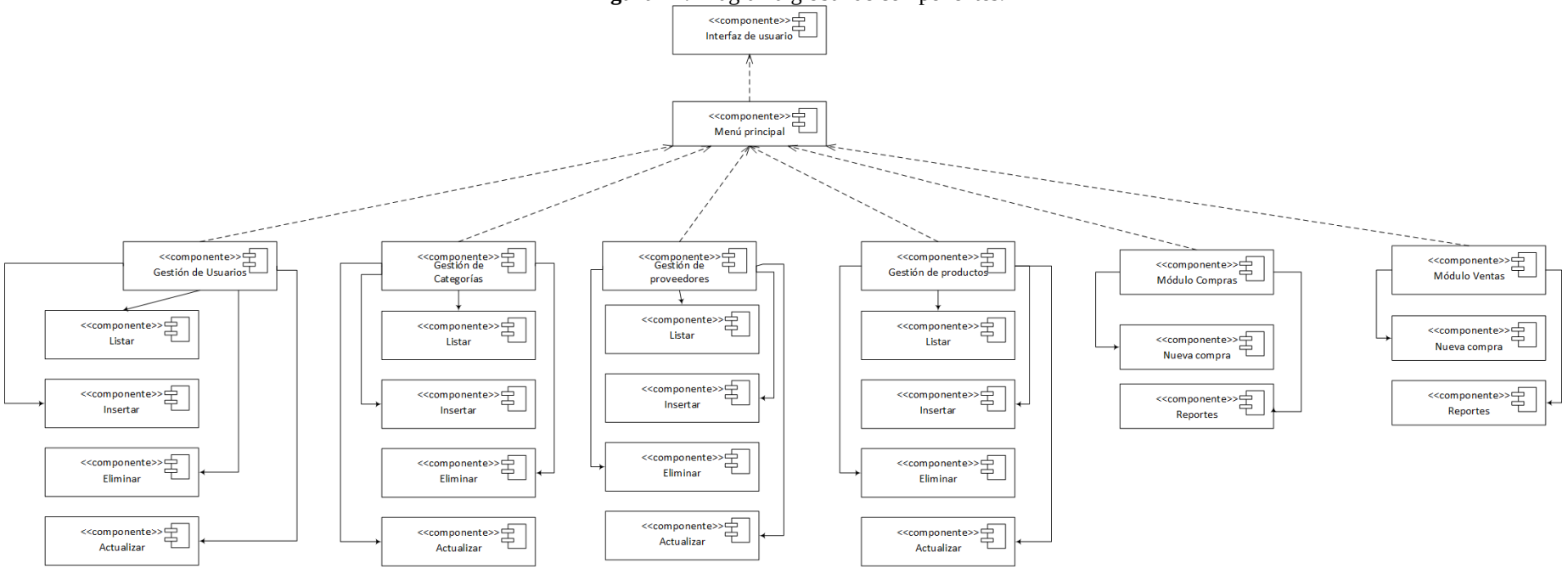
Las clases con sus respectivos métodos se encuentran en el *Anexo 5*.

4.1.5. Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Un diagrama de componentes tiene un nivel más alto de abstracción que un diagrama de clases, prácticamente un componente se implementa de una o más clases u objetos en tiempo de ejecución.

El diagrama de la Figura 11 representa las partes integrantes de la aplicación, así como la dependencia entre los módulos de la aplicación.

Figura 11. Diagrama global de componentes.



Fuente: Elaboración propia

4.1.6. Pruebas TDD

4.1.6.1. Especificación del requisito en pruebas unitarias

Una vez definidas las historias de usuario y los diseños del software se proceden a expresar dichas funciones en códigos de prueba. El Framework JUnit provee la API necesaria para realizar las pruebas unitarias de cada requisito del software. Cada prueba permite evaluar si un método dentro de un módulo funciona de forma correcta y comprobar el comportamiento de los métodos de cada módulo.

Para realizar las pruebas unitarias a las funciones del sistema, se debe crear la clase de la entidad a analizar. De acuerdo con el proceso de implementación del marco de trabajo TDD la prueba que se implemente en las fuentes del proyecto debe fallar como primer paso, para luego realizar la refactorización y corregir los inconvenientes presentados.

A continuación, como ejemplo se define la prueba “Agregar productos”, posteriormente se detallan cada uno de los pasos, hasta obtener la ejecución satisfactoria. El proceso para ejecutar una prueba con la técnica TDD es el siguiente:

a. Escribir una prueba unitaria

Nótese que dentro de la prueba unitaria se incluye la etiqueta **@Test** y el método **assertEquals** los mismos que indican que se está aplicando un test unitario.

Figura 12. Prueba unitaria para el método "agregar productos"

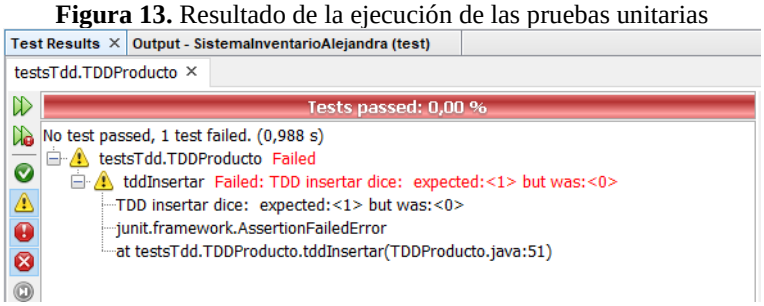
```
@Before
public void init() {
    pdao = new ProductoDAO();
    objProducto = new Producto();
    cdao = new CategoriaDAO();
    objCategoria = new Categoria();
    provdao = new ProveedorDAO();
    objProveedor = new Proveedor();
}

/*
insertar un nuevo producto
 */
@Test
public void tddInsertar() throws Exception {
    int result = 0;
    objProducto.setNombre("test nombre insertar");
    objCategoria = cdao.buscarCategoria(5);
    objProducto.setIdcategoria(objCategoria);
    objProveedor = provdao.buscarProveedor(7);
    objProducto.setIdproveedor(objProveedor);
    objProducto.setPrecio(10);
    objProducto.setStock(10);
    result = pdao.agregar(objProducto);
    Assert.assertEquals(1, result);
}
```

Fuente: Elaboración propia

b. Ejecutar las pruebas

Consiste en ejecutar las pruebas y verificar si pasan o no, cabe mencionar que no siempre las pruebas se ejecutarán satisfactoriamente, ya que el objetivo de TDD es hacerlas fallar. En la Figura 13, se muestra un error que debe ser refactorizado.

Figura 13. Resultado de la ejecución de las pruebas unitarias



Fuente: Elaboración propia

c. Refactorización del código

Para corregir el error encontrado durante la ejecución del test, ubicar la clase Java y la línea específica donde está mal codificada. En este caso, el problema se soluciona agregando un bloque excepciones *try* y *catch*, dentro del método agregar producto de la clase ProductoDAO. A continuación, se presenta el código refactorizado en la Figura 14.

Figura 14. Refactorización del método Agregar en la clase ProductoDAO

```
public int agregar(Producto producto) {
    int r = 0;
    String sql = "INSERT INTO producto(nombre,idcategoria,idproveedor,precio,stock) values (?,?,?,?,?)";
    try {
        con = conectar.getConnection();
        ps = con.prepareStatement(sql);
        ps.setString(1, producto.getNombre());
        ps.setInt(2, producto.getIdcategoria().getIdcategoria());
        ps.setInt(3, producto.getIdproveedor().getIdproveedor());
        ps.setDouble(4, producto.getPrecio());
        ps.setInt(5, producto.getStock());
        r = ps.executeUpdate();
        con.close();
        if (r == 1) {
            return 1;
        } else {
            return 0;
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    return r;
}
```

Fuente: Elaboración propia

De forma similar se modifica el código de la prueba unitaria, agregando una excepción y las cláusulas de *try* y *cacth*.

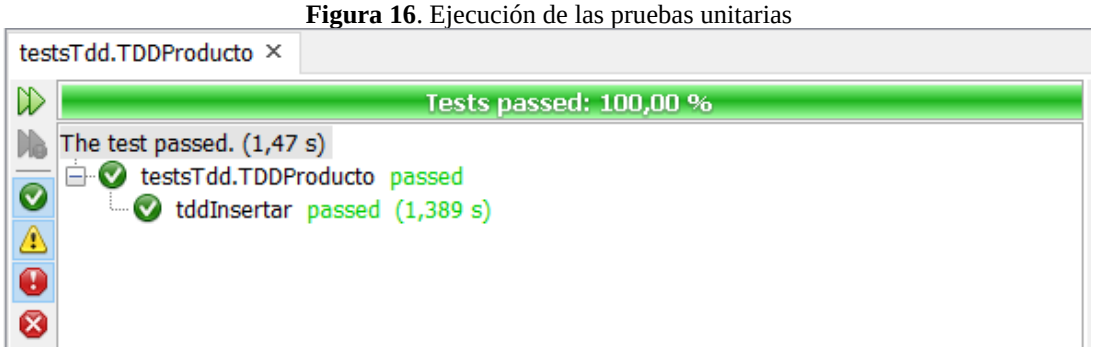
Figura 15. Refactorización de la prueba unitaria

```
@Test
public void tddInsertar() throws Exception {
    int result = 0;
    try {
        objProducto.setNombre("test nombre insertar");
        objCategoria =  cdao.buscarCategoria(5);
        objProducto.setIdcategoria(objCategoria);
        objProveedor =  provdao.buscarProveedor(7);
        objProducto.setIdproveedor(objProveedor);
        objProducto.setPrecio(10);
        objProducto.setStock(10);
        result = pdao.agregar(objProducto);
    } catch (Exception e) {
        System.out.println("public void tddInsertar() dice: " + e.getMessage
    }
    Assert.assertEquals("TDD insertar dice: ", 1, result);
}
```

Fuente: Elaboración propia

Finalmente, el código se ejecuta correctamente.

Figura 16. Ejecución de las pruebas unitarias



Fuente: Elaboración propia

Al finalizar la implementación de la metodología de desarrollo de software orientado por pruebas, es importante señalar que para efectos del presente documento solo se muestran algunos ejemplos de pruebas unitarias, para identificar el proceso que tiene el marco de trabajo TDD, se detalla en el *Anexo 7*.

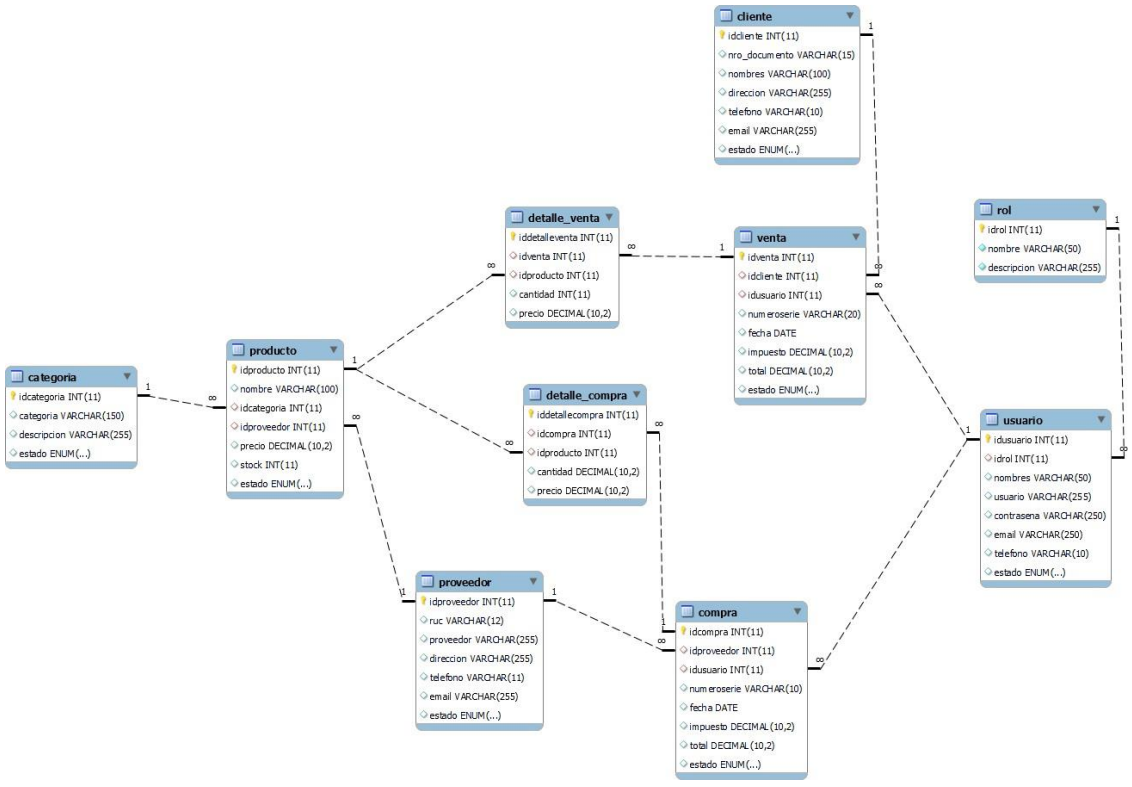
4.1.7. Sistema de control de inventarios

Luego de finalizar las pruebas unitarias y entender por completo el rumbo que persigue el proyecto, se procede a codificar el software. La metodología TDD precisa en sus procesos, que al contar con las pruebas unitarias totalmente correctas y organizadas, los códigos son trasladados como funcionalidades de las fuentes finales del sistema.

4.1.7.1. Diseño de base de datos

Con el objetivo de mantener los datos organizados, manipular, almacenar e integridad de la información que se genera en el almacén Alejandra, se realizó el diseño de la base de datos, de acuerdo con los requerimientos obtenidos para el desarrollo del sistema, se puede identificar las entidades, atributos y sus relaciones en la Figura 16**,** se muestra el modelo físico.

Figura 17. Diagrama de bases de datos



Fuente: Elaboración propia

4.1.7.2. Diccionario de datos

El diccionario de datos permite visualizar la nomenclatura de aquellos datos que se van a utilizar en el aplicativo, este denota el nombre, tipo de dato, valores aceptados y clave. En este diccionario se listan los tipos de datos utilizados para cada entidad proveedor, creada para el funcionamiento del sistema. El diccionario de datos se muestra en el *Anexo 4.*

4.1.7.3. Historias de usuario

Las historias representan los requisitos provenientes del Sprint Backlog, cada una cuenta con un identificador, nombre, descripción y responsable; así como sus tareas de ingeniería y pruebas de aceptación. Una historia de usuario comprueba el correcto cumplimiento de una funcionalidad.

Todas las historias de usuarios fueron plasmadas tomando en cuenta los requisitos que caracterizan a un sistema de control de inventarios. Al finalizar el desarrollo del sistema se obtuvieron 29 historias de usuario y 8 historias técnicas.

A continuación, se describen 10 historias de usuario y las demás se detallan en el *Anexo 6.*

Tabla 11. Historia de usuario 01

HISTORIA DE USUARIO 01	
ID: HU-01	Nombre Historia: Ingresar categoría.
Usuario: Administrador	Sprint Asignado: 02
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito ingresar en la base de datos las diferentes categorías de los productos.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando el registro se ha realizado exitosamente.	

Fuente: Elaboración propia

Tabla 12. Historia de usuario 02

HISTORIA DE USUARIO 02	
ID: HU-02	Nombre Historia: Modificar categoría
Usuario: Administrador	Sprint Asignado: 02
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito modificar la información de las categorías almacenadas en la base de datos.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando el registro se ha realizado exitosamente.	

Fuente: Elaboración propia

Tabla 13. Historia de usuario 03

HISTORIA DE USUARIO 03	
ID: HU-03	Nombre Historia: Agregar vigencia de categoría
Usuario: Administrador	Sprint Asignado: 02
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito ingresar la vigencia de una categoría.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando el registro se ha realizado exitosamente.	

Fuente: Elaboración propia

Tabla 14. Historia de usuario 04

HISTORIA DE USUARIO 04	
ID: HU-04	Nombre Historia: Gestión productos.
Usuario: Administrador	Sprint Asignado: 02
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito un formulario donde pueda gestionar los productos.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando el registro se ha realizado exitosamente.	

Fuente: Elaboración propia

Tabla 15. Historia de usuario 05

HISTORIA DE USUARIO 05	
ID: HU-05	Nombre Historia: Ingresar productos.
Usuario: Administrador	Sprint Asignado: 03
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito ingresar productos especificando todas sus características para poder controlar de cada uno su stock en bodega.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación una vez ingresado el producto.	

Fuente: Elaboración propia

Tabla 16. Historia de usuario 06

HISTORIA DE USUARIO 06	
ID: HU-06	Nombre Historia: Modificar productos
Usuario: Administrador	Sprint Asignado: 03
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito actualizar la información de los productos especificando todas sus características.	

Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación una vez actualizado el producto.

Fuente: Elaboración propia

Tabla 17. Historia de usuario 07

HISTORIA DE USUARIO 07	
ID: HU-07	Nombre Historia: Buscar para el módulo almacén
Usuario: Administrador	Sprint Asignado: 03
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito un formulario para la realizar búsquedas en el módulo almacén.	
Pruebas de Aceptación: ✓ Mostrar un mensaje con el total de registros obtenidos.	

Fuente: Elaboración propia

Tabla 18. Historia de usuario 08

HISTORIA DE USUARIO 08	
ID: HU-08	Nombre Historia: Establecer vigencia de producto.
Usuario: Administrador	Sprint Asignado: 03
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito ingresar la vigencia de los productos.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación una vez ingresado el producto.	

Fuente: Elaboración propia

Tabla 19. Historia de usuario 09

HISTORIA DE USUARIO 09	
ID: HU-09	Nombre Historia: Reportes para módulo almacén
Usuario: Administrador	Sprint Asignado: 02
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito tener reportes de los productos, proveedores y categorías del módulo almacén. Los reportes se pueden descargar en diferentes formatos como PDF, PNG, XML.	
Pruebas de Aceptación: ✓Se muestra un mensaje de confirmación, posteriormente se presentan los reportes.	

Fuente: Elaboración propia

Tabla 20. Historia de usuario 10

HISTORIA DE USUARIO 10	
ID: HU-10	Nombre Historia: Crear sesión de usuarios

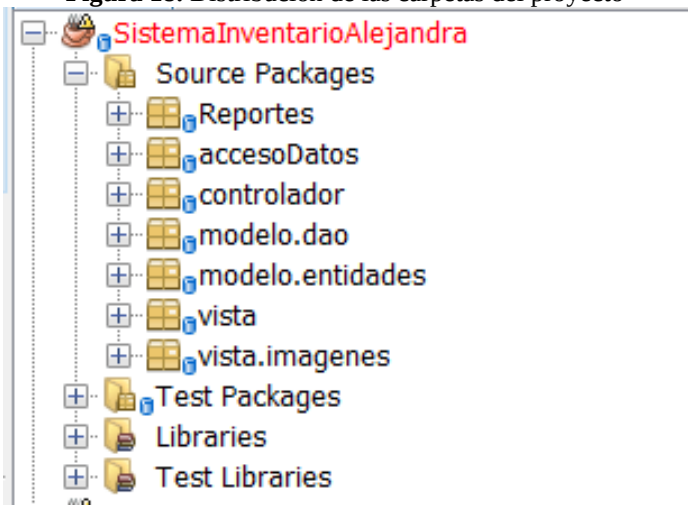
Usuario: Administrador	Sprint Asignado: 02
Fecha Inicio:	Fecha Fin:
Descripción: Como usuario del sistema necesito que mi sesión sea capturada mientras dure use la aplicación.	
Pruebas de Aceptación: ✓ se muestran los datos del usuario logueado.	

Fuente: Elaboración propia

4.1.7.4. Codificación del software

El software se desarrolló utilizando el IDE NetBeans en la versión 8.2 el mismo que permite crear aplicaciones de Junit. El proyecto tiene la siguiente estructura.

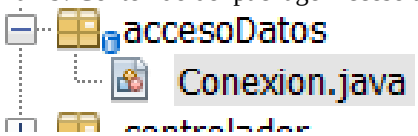
Figura 18. Distribución de las carpetas del proyecto



Fuente: Elaboración propia

Acceso a datos: Contiene las clases que realizan la comunicación hacia y desde el servidor de base de datos. La Figura 19, muestra las clases incluidas en este paquete.

Figura 19. Contenido del package Acceso a datos



Fuente: Elaboración propia

Figura 20. Contenido de la clase conexión

```
package accesoDatos;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.swing.JOptionPane;

public class Conexion {
    Connection con;

    public Connection getConnection(){
        try {
            String db = "jdbc:mysql://localhost:3306/sistemapos";
            con = DriverManager.getConnection(db, "root", "");
            return con;
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, e.toString());
        }
        return null;
    }
}
```

Fuente: Elaboración propia

Desarrollo de la capa modelo, con el fin de organizar el código está capa se dividió en dos: entidades y DAO.

- **Entidades:** este paquete contiene las entidades de la aplicación junto con sus atributos y métodos de acceso. La Figura 21, muestra las clases incluidas dentro del package Entidades.

Figura 21. Clases que conforman el package entidades

Fuente: Elaboración propia

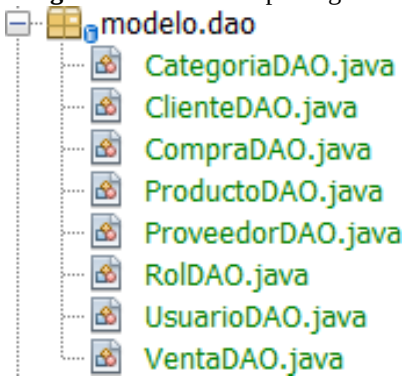
Figura 22. Código de la clase Cliente

```java
package modelo.entidades;

public class Cliente {

    private int idcliente;
    private String nro_documento;
    private String nombres;
    private String direccion;
    private String telefono;
    private String email;
    private String estado;

    public Cliente() {
    }

    public Cliente(int idcliente, String nro_documento, String nombres, String direccion, String telefono, String email) {
        this.idcliente = idcliente;
        this.nro_documento = nro_documento;
        this.nombres = nombres;
        this.direccion = direccion;
        this.telefono = telefono;
        this.email = email;
    }

    /**
     * @return the idcliente
     */
    public int getIdcliente() {
        return idcliente;
    }

    /**
     * @param idcliente the idcliente to set
     */
    public void setIdcliente(int idcliente) {
        this.idcliente = idcliente;
    }

    /**
     * @return the nro_documento
     */
    public String getNro_documento() {
        return nro_documento;
    }
```

Fuente: Elaboración propia

- **DAO:** contiene las operaciones que se realizan con las entidades, generalmente incluyen las funciones de CRUD y otras que realizan alguna función específica solicitada por los usuarios, en la Figura 23 se muestra las clases que incluye el package DAO.

Figura 23. Clases del package DAO



Fuente: Elaboración propia

Figura 24. Código clase ProductoDAO y definición del método Listar producto

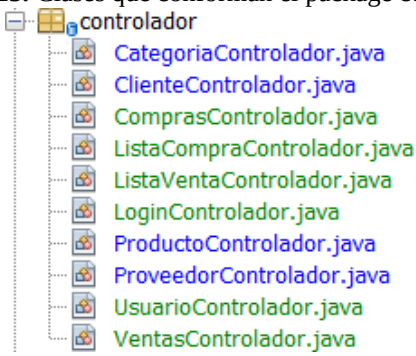
```
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import modelo.entidades.Categoria;
import accesoDatos.Conexion;
import java.sql.SQLException;
import modelo.entidades.Producto;
import modelo.entidades.Proveedor;

public class ProductoDAO {

    PreparedStatement ps;
    ResultSet rs;
    Connection con;
    Conexion conectar = new Conexion();
    Producto pro = new Producto();

    public List listar() {
        List<Producto> datos = new ArrayList<>();
        try {
            con = conectar.getConnection();
            ps = con.prepareStatement("SELECT * FROM producto ORDER BY estado ASC");
            rs = ps.executeQuery();
            while (rs.next()) {
                Producto p = new Producto();
                p.setIdproducto(rs.getInt(1));
                p.setNombre(rs.getString(2));
                CategoriaDAO cdao = new CategoriaDAO();
                Categoria categoria = cdao.buscarCategoria(rs.getInt(3));
                p.setIdcategoria(categoria);
                ProveedorDAO pdao = new ProveedorDAO();
                Proveedor proveedor = pdao.buscarProveedor(rs.getInt(4));
                p.setIdproveedor(proveedor);
                p.setPrecio(rs.getDouble(5));
                p.setStock(rs.getInt(6));
                p.setEstado(rs.getString(7));
                datos.add(p);
            }
            con.close();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return datos;
    }
```

Fuente: Elaboración propia

Desarrollo de la capa controlador: En esta capa las diferentes clases reciben los eventos desde la vista (formularios) y ejecutan la respectiva acción. El controlador define el comportamiento de la aplicación, despacha las peticiones del usuario y selecciona las vistas de presentación siguiente, basándose en la información introducida por el usuario y el resultado de las operaciones realizadas por el modelo. Es decir, interpreta las entradas del usuario y las mapea en acciones a ser efectuadas por el modelo.

La Figura 25, presenta los diferentes controladores de la aplicación.

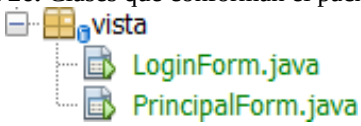
Figura 25. Clases que conforman el package controlador



Fuente: Elaboración propia

Desarrollo de la capa vista: contiene los formularios que se encargan de acceder a los datos del modelo, especifica cómo se deben presentar esos datos y actualiza la presentación de los mismos cuando ocurren cambios en el modelo. La Figura 26, muestra los diferentes controladores de la aplicación.

Figura 26. Clases que conforman el package vista



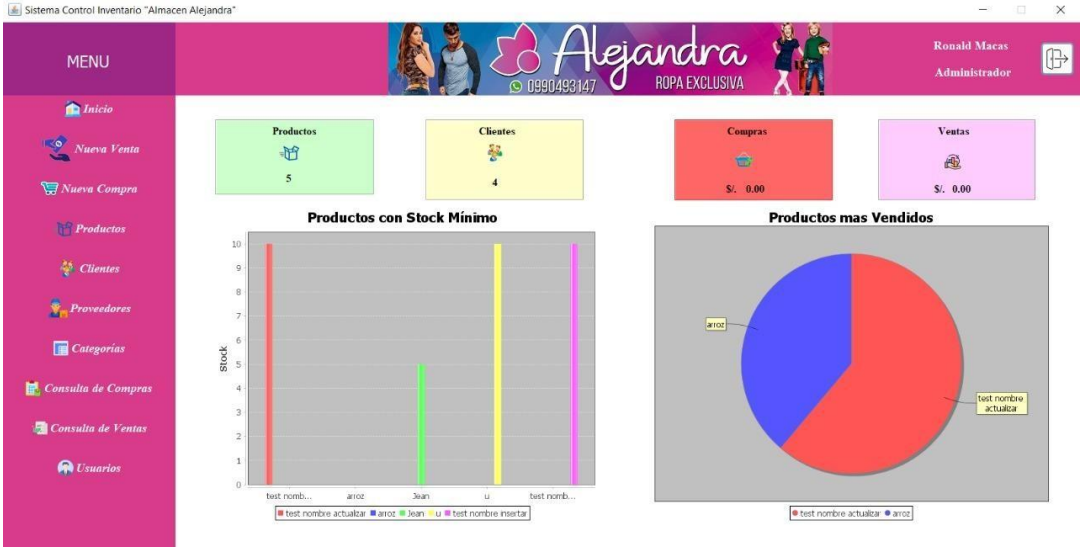
Fuente: Elaboración propia

A continuación, se presentan algunas vistas del sistema en un entorno de pruebas.

Figura 27. Formulario de acceso al sistema (Login)

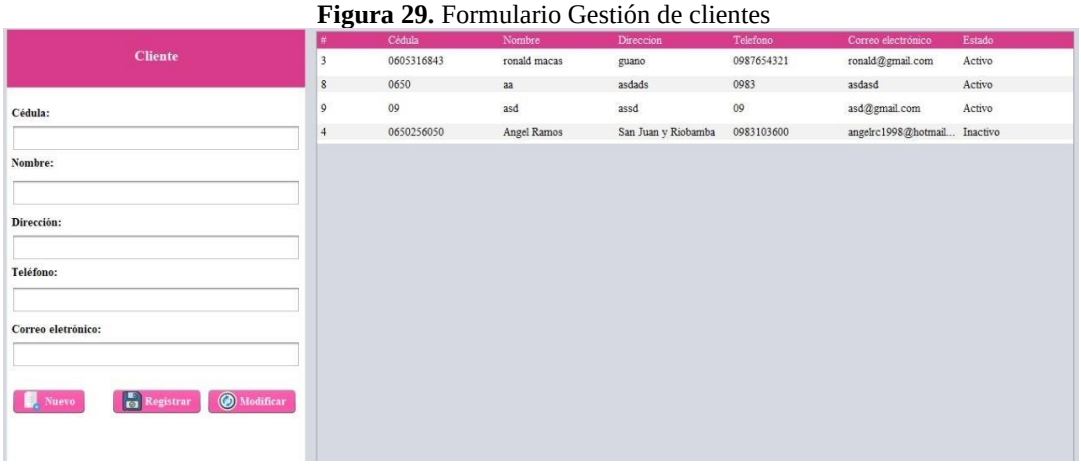
Fuente: Elaboración propia

Figura 28. Menú principal del sistema



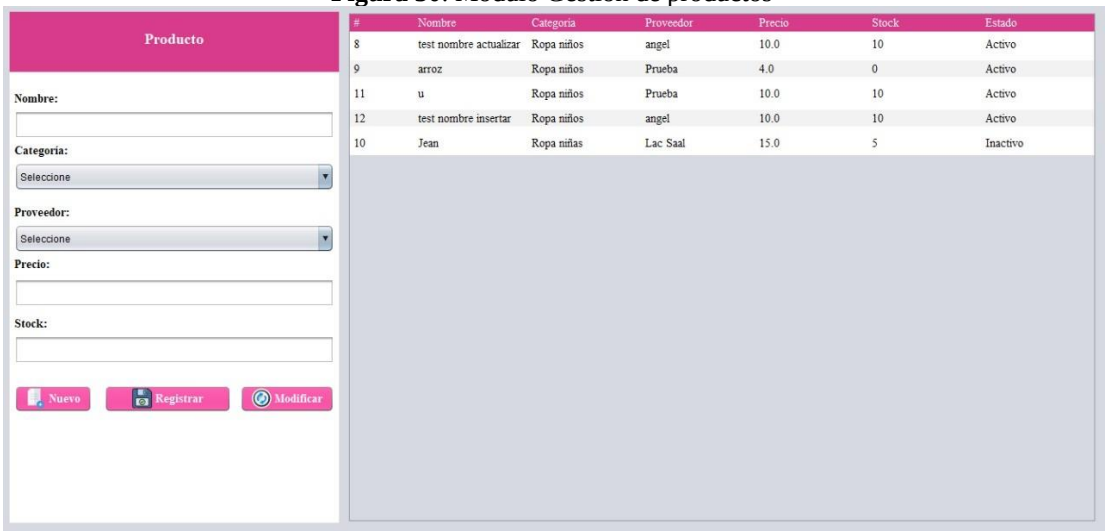
Fuente: Elaboración propia

Figura 29. Formulario Gestión de clientes



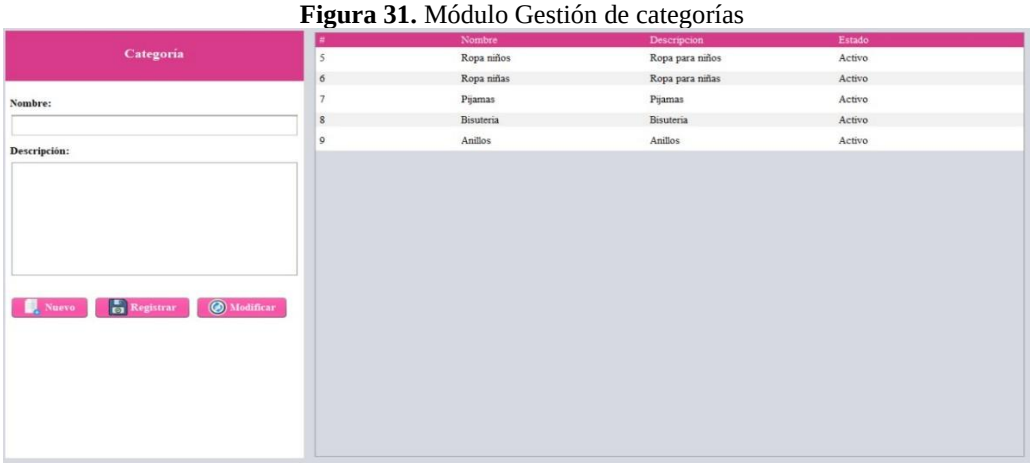
Fuente: Elaboración propia

Figura 30. Módulo Gestión de productos



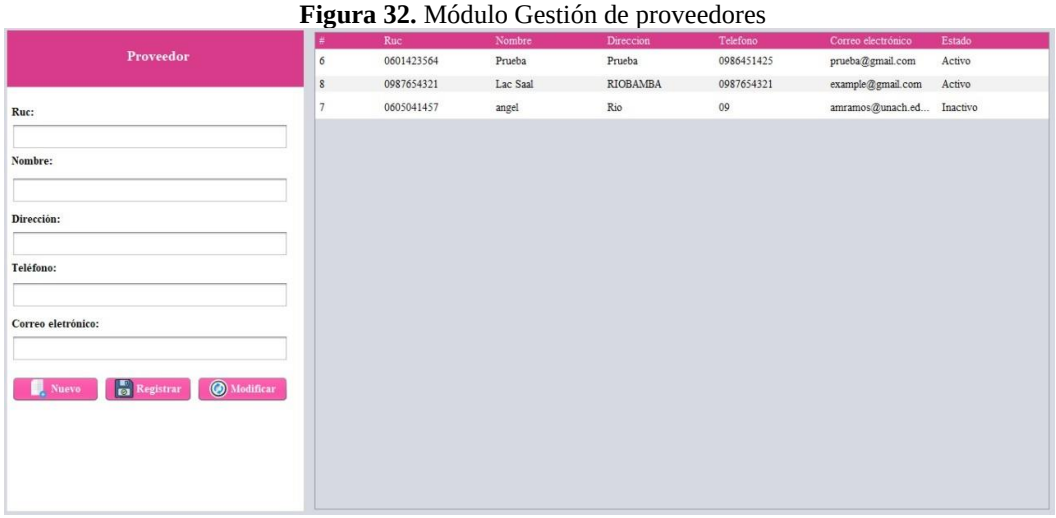
Fuente: Elaboración propia

Figura 31. Módulo Gestión de categorías



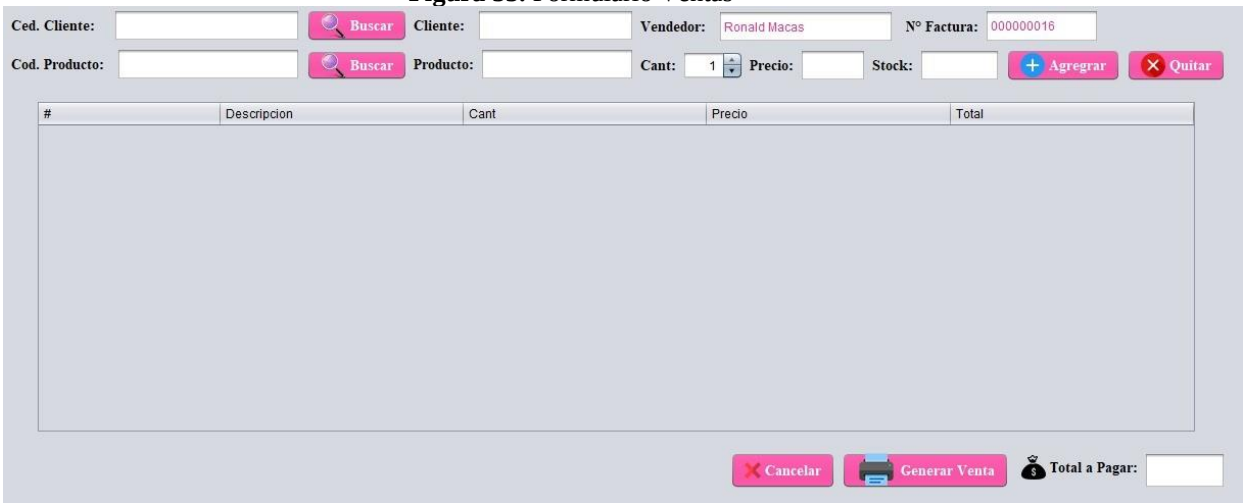
Fuente: Elaboración propia

Figura 32. Módulo Gestión de proveedores



Fuente: Elaboración propia

Figura 33. Formulario Ventas



Fuente: Elaboración propia

4.1.7.5. Manual de Usuario

Es un documento que sirve de guía para el Product Owner, donde se encuentra detallado paso a paso cada una de las funcionalidades que realiza el aplicativo. Ver el *Anexo 1*.

4.1.8. Fase de cierre

En esta fase se detalla todas las actividades que se realizan para la finalización del desarrollo del sistema, estableciendo para el mismo el denominado Sprint BurnDown Chart, el cual consiste en una representación gráfica del trabajo concluido y las actividades pendientes del proyecto.

Para validar la funcionalidad o conformidad de la elaboración de cada historia de usuario se realizaron pruebas de funcionalidad por cada historia de usuario y se verificaron que cumplan con lo establecido en el Product Backlok y Sprint Backlok.

La Tabla 21, muestra el TaskBoard donde se aprecia que los Sprints y las Historias de Usuario finalizadas.

Tabla 21. Taskboard del proyecto

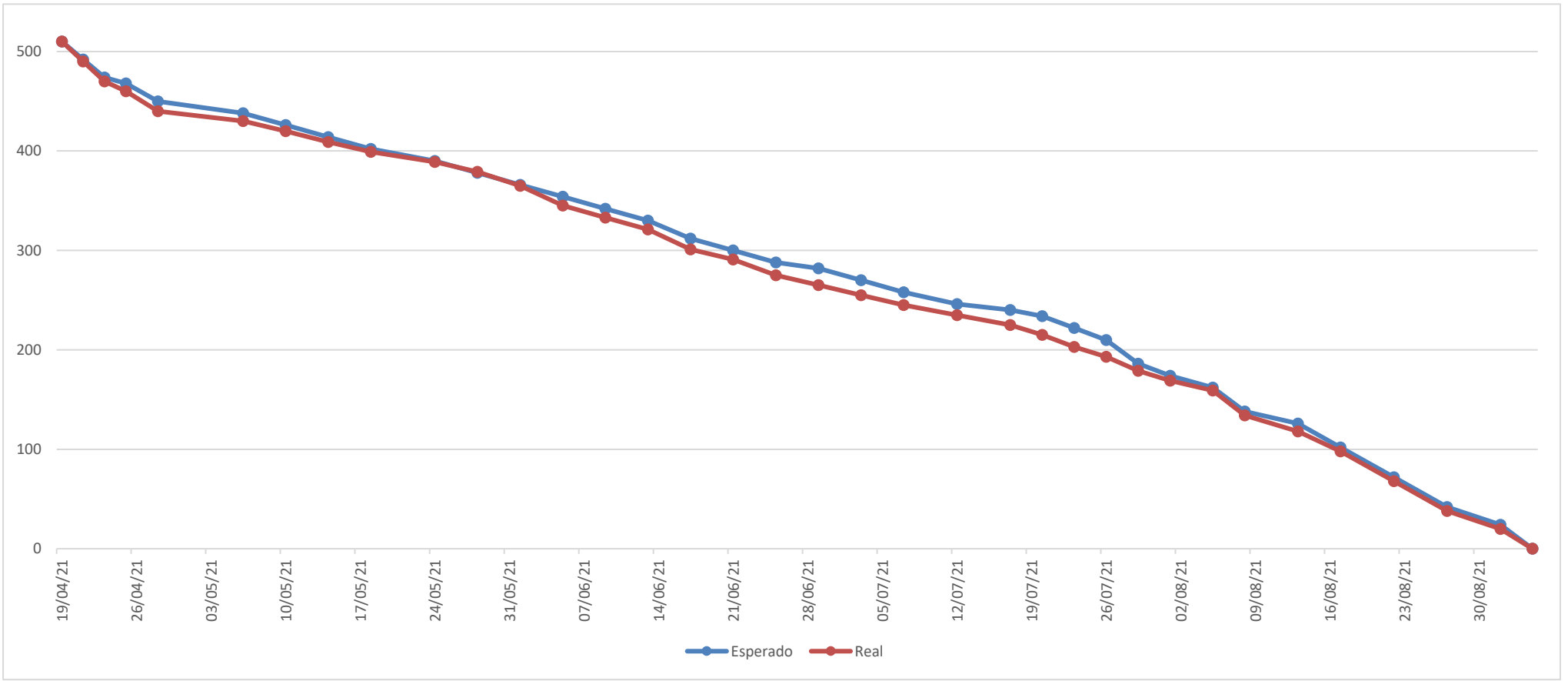
ID	DESCRIPCIÓN	ESTADO		
		PENDIENTE	EN CURSO	FINALIZADO
SPRINT 1				
HT-01	Establecer los requerimientos del sistema			X
HT-02	Establecer la arquitectura del sistema			X
HT-03	Diseño de Estándar de Codificación			X
HT-04	Diseño del estándar de interfaz			X
SPRINT 2				
HT-05	Realizar la Base Datos (Modelo entidad relación, lógico, físico, script diccionario de datos)			X
HU-01	Ingresar categoría			X
HU-02	Modificar categoría			X
HU-03	Agregar vigencia de categoría			X
HU-04	Gestión productos			X
SPRINT 3				
HU-05	Agregar productos			X
HU-06	Modificar productos			X
HU-07	Buscar para el módulo almacén			X
HU-08	Establecer vigencia de producto			X
HU-09	Reportes para modulo almacén			X
SPRINT 4				
HU-10	Crear sesión de usuarios			X
HU-11	Gestionar roles			X
HU-12	Asignar funcionalidad de acuerdo con el rol de usuario			X
HU-13	Activar estado de vigencia de la funcionalidad asignada			X
HU-14	Buscar y visualizar usuarios empleados de la empresa			X

SPRINT 5				
HU-15	Gestionar proveedores			X
HU-16	Gestionar compras a proveedores.			X
HU-17	Consultar compras por fecha, mes.			X
HU-18	Buscar proveedores			X
HU-19	Historial de precios			X
HU-20	Reportes para el módulo proveedores			X
SPRINT 6				
HU-21	Gestionar clientes			X
HU-22	Gestionar venta			X
HU-23	Consultar ventas por día, fecha, mes			X
HU-24	Reportes para el módulo ventas			X
SPRINT 7				
HU-25	Administrar caja			X
HU-26	Historial de caja			X
HU-27	Generar Inventario			X
SPRINT 8				
HU-28	Resumen de saldos y movimiento de productos			X
HU-29	Graficas estadísticas de compras y ventas			X
SPRINT 9				
HT-07	Capacitación de usuarios			X
HT-08	Documentación final del Sistema			X

Fuente: Elaboración propia.

En la Figura 34, se aprecia el Burn Down Chart del proyecto donde se muestra que las actividades de las historias de usuario se desarrollaron fuera del tiempo previsto, pero aún se encuentra dentro de la línea base del mismo.

Figura 1. Burn Down Chart final



Fuente: Elaboración propia

CAPÍTULO V

5. RESULTADOS Y DISCUSION

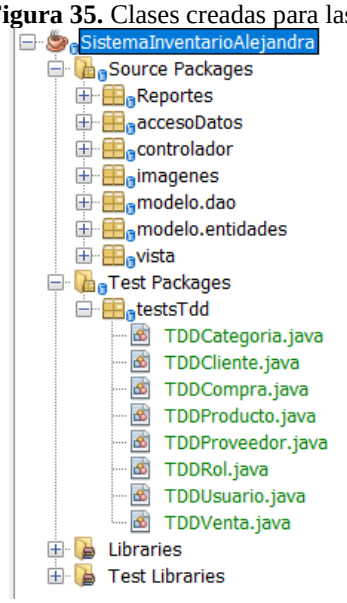
5.1. Resultados

5.1.1. JUnit como “framework” para realizar pruebas unitarias

Como ya se mencionó en la fundamentación teórica “Test Driven Development” es una técnica de programación extrema, donde primero se desarrollan las pruebas unitarias antes de escribir cualquier línea de código funcional.

Para cada uno de los puntos funcionales del sistema se creó una clase de pruebas en el cual se implementaron las pruebas unitarias necesarias. En la Figura 35, se muestra el paralelismo.

Figura 35. Clases creadas para las pruebas TDD



Fuente: Elaboración propia

Cada clase de prueba contiene una sección, para la declaración de variables globales a usar en toda la clase y contiene una operación definida en su correspondiente clase DAO, es decir la implementación de cada uno de los métodos que contienen las pruebas unitarias para el código funcional.

Cabe destacar que, para que un método de prueba sea exitoso o no, se utiliza los métodos *assert*, para comparar los valores esperados con los valores reales, o directamente hacer que el método de prueba falle.

5.1.1.1. Pruebas unitarias a la clase Producto

A continuación, se presentan las pruebas definidas en la clase TDDProducto.

Figura 36. Declaración del método init

```
/*
inicilaizar los objetos
 */
@Before
public void init() {
    pdao = new ProductoDAO();
    objProducto = new Producto();
    cdao = new CategoriaDAO();
    objCategoria = new Categoria();
    provdao = new ProveedorDAO();
    objProveedor = new Proveedor();
}
```

Fuente: Elaboración propia

@Before es la anotación que indica que el método adjunto se ejecutará antes que cualquier prueba en la clase. Se utiliza principalmente para configurar algunos objetos necesarios para la correcta ejecución de las pruebas.

Figura 37. Definición de la prueba para el método insertar

```
@Test
public void tddInsertar() throws Exception {
    int result = 0;
    try {
        objProducto.setNombre("test nombre insertar");
        objCategoria =  cdao.buscarCategoria(5);
        objProducto.setIdcategoria(objCategoria);
        objProveedor =  provdao.buscarProveedor(7);
        objProducto.setIdproveedor(objProveedor);
        objProducto.setPrecio(10);
        objProducto.setStock(10);
        result = pdao.agregar(objProducto);
    } catch (Exception e) {
        System.out.println("public void tddInsertar() dice: " + e.getMessage
    }
    Assert.assertEquals("TDD insertar dice: ", 1, result);
}
```

Fuente: Elaboración propia

@Test indica que el método adjunto es una prueba unitaria. Eso le permite usar cualquier nombre de método para realizar una prueba.

Figura 38. Definición de la prueba “listar todos los productos”

```
/*
Listar todos los productos
*/
@Test
public void tddListar() {
    List<Producto> lstproductos = new ArrayList<>();
    try {
        lstproductos = dao.listar();
    } catch (Exception e) {
        System.out.println("Tdd listar dice: " + e.getMessage());
    }
    Assert.assertNotNull("Tdd listar productos", lstproductos);
}
```

Fuente: Elaboración propia

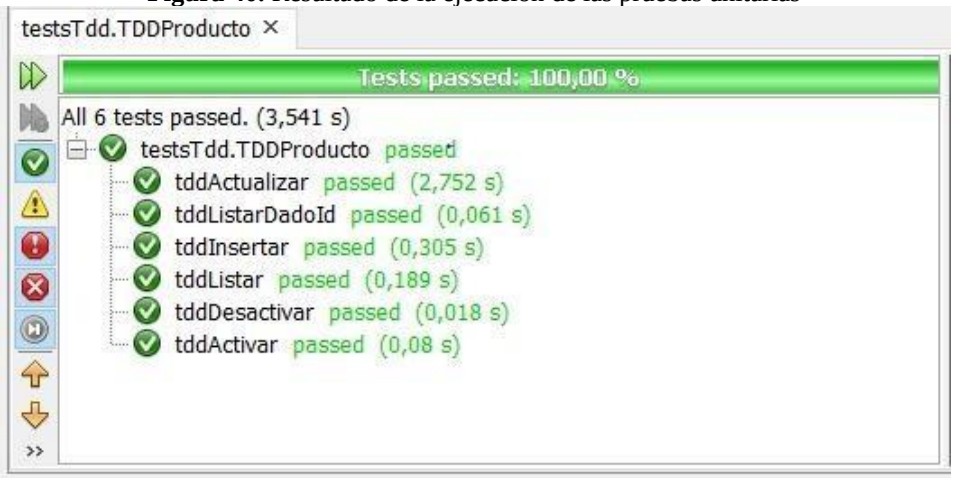
Figura 39. Definición de la prueba "buscar dado el id del producto"

```
/*
buscar producto por id
*/
@Test
public void tddListarDadoId() {
    objProducto = pdao.buscarProducto(8);
    Assert.assertNotNull("Tdd listar productos dado Id: ", objProducto.getIdproducto());
}
```

Fuente: Elaboración propia

El resultado obtenido por las pruebas tanto exitosas como erróneas se verá de forma similar como la Figura 40.

Figura 40. Resultado de la ejecución de las pruebas unitarias



Fuente: Elaboración propia

5.1.1.2. Pruebas unitarias a la clase Cliente

Al igual que la clase Producto se plantearon pruebas unitarias para la clase Cliente con las operaciones básicas de un CRUD, junto con otras operaciones necesarias para

determinar el funcionamiento de la aplicación en diferentes escenarios. A continuación, se presentan las pruebas realizadas y los resultados obtenidos.

Figura 41. Código fuente de la prueba listar Clientes

```
/*
Listar todos los clientes
 */
@Test
public void tddListar() {
    List<Cliente> lstclientes = new ArrayList<>();
    try {
        lstclientes = cdao.listar();
    } catch (Exception e) {
        System.out.println("Tdd listar dice: " + e.getMessage());
    }
    Assert.assertNotNull("Tdd listar clientes", lstclientes);
}
```

Fuente: Elaboración propia

Figura 42. Código fuente de la prueba unitaria, buscar cliente por su identificador

```
/*
buscar cliente por id
 */
@Test
public void tddListarDadoId() {
    objCliente = cdao.buscarCliente(3);
    Assert.assertNotNull("Tdd listar clientes dado Id: ", objCliente.getIdcliente());
}
```

Fuente: Elaboración propia

Figura 43. Código fuente del método insertar

```
/*
insertar un nuevo cliente
 */
@Test
public void tddInsertar() throws Exception {
    int result = 0;
    try {
        objCliente.setNro_documento("0123456789");
        objCliente.setNombres("test nombre insertar");
        objCliente.setDireccion("test direccion insertar");
        objCliente.setTelefono("0987654321");
        objCliente.setEmail("testinsertar@test.com");
        result = cdao.agregar(objCliente);
    } catch (Exception e) {
        System.out.println("public void tddInsertar() dice: " + e.getMessage
    }
    Assert.assertEquals("TDD insertar dice: ", 1, result);
}
```

Fuente: Elaboración propia

Con el objetivo de determinar el correcto funcionamiento de la aplicación con respecto a la inserción, actualización y búsqueda masiva de datos, se implementaron tres pruebas que ejecutan cada una de las operaciones antes descritas, dentro de un ciclo repetitivo.

Figura 44. Código fuente del test inserción masiva

```
/*
creación de un método que inserta 200 usuarios de prueba
 */
@Test
public void fullInsert() {
    boolean cont = false;
    int band = 0;
    String numberId = "0603977";
    String names = "test ";
    String address = "test ";
    String phone = "0996824";
    try {
        for (int i = 0; i < 200; i++) {
            cliente = new Cliente();
            cliente.setNro_documento(numberId + transformNumber(i));
            cliente.setNombres(names + transformNumber(i));
            cliente.setDireccion(address + transformNumber(i));
            cliente.setTelefono(phone + transformNumber(i));
            cliente.setEmail("test" + transformNumber(i) + "@test.com");
            band = clienteDAO.agregar(cliente);
            cont = true;
        }
    } catch (Exception e) {
        System.out.println("error: " + e.getMessage());
    }
    Assert.assertTrue(cont);
}
```

Fuente: Elaboración propia

Figura 45. Código fuente del test actualización masiva

```
/*
actualización en masa
 */
@Test
public void fullUpdate() {
    boolean band = false;
    int upd = 0;
    String names = "update test ";
    String address = "update test ";

    try {
        clientes = clienteDAO.listar();
        for (int i = 0; i < clientes.size(); i++) {
            cliente = new Cliente();
            cliente = clienteDAO.listarID(clientes.get(i).getNro_documento());
            System.out.println("cédula: " + clientes.get(i).getNro_documento());
            cliente.setNombres(names + transformNumber(i + 1));
            cliente.setDireccion(address + transformNumber(i + 1));
            cliente.setEmail("update_test" + transformNumber(i + 1) + "@test.com");
            upd = clienteDAO.Actualizar(cliente);
            if (upd == 1) {
                band = true;
            } else {
                band = false;
            }
        }
    } catch (Exception e) {
        System.out.println("error: " + e.getMessage());
    }
    Assert.assertTrue(band);
}
```

Fuente: Elaboración propia

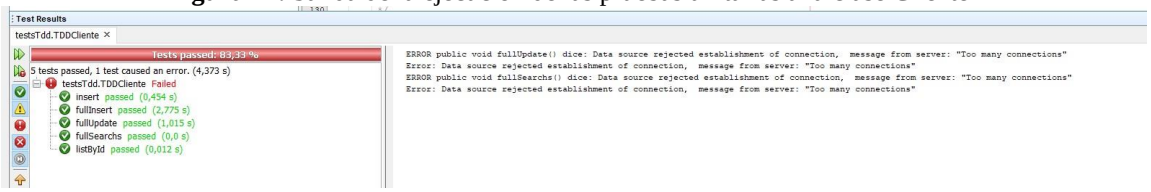
Figura 46. Test para las búsquedas masivas de clientes

```
/*
consulta masiva de datos
*/
@Test
public void fullSearchs(){
    try {
        for (int i=0; i<999;i++){
            clientes=clienteDAO.listar();
            System.out.println("clientes almacenados: "+clientes.size());
        }
    } catch (Exception e) {
        System.out.println(" public void fullSearchs() dice: "+e.getMessage());
    }
}
```

Fuente: Elaboración propia

Al ejecutar las pruebas definidas para la clase cliente, arroja un mensaje de error que se visualiza en la Figura 47.

Figura 47. Salida de la ejecución de las pruebas unitarias a la clase Cliente



Fuente: Elaboración propia

Estos errores se presentan debido a que existe muchas conexiones a la base de datos, la solución consiste en agregar una línea código para cerrar la conexión a la base de datos al finalizar cualquier operación.

Figura 48. Refactorización de las operaciones en la clase Cliente DAO

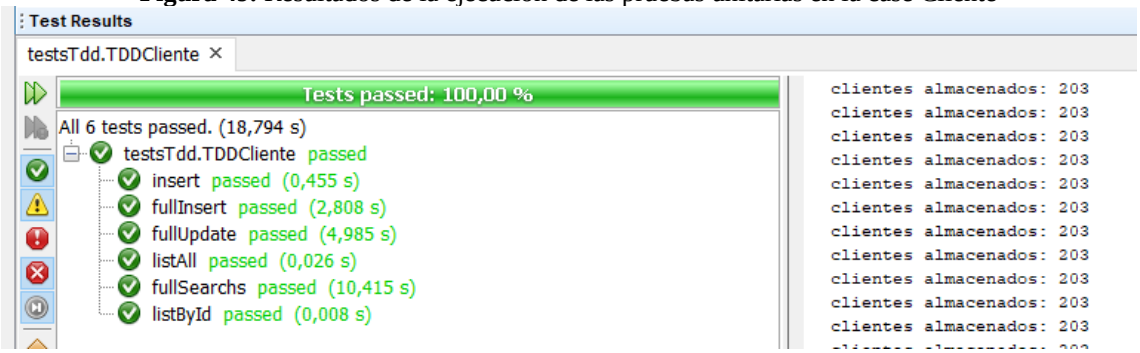
```
public int agregar(Cliente cliente) throws Exception {
    String sql = "INSERT INTO cliente(nro_documento,nombres,direccion,telefono,email) values (?,?,?,?,?)";
    try {
        con = conectar.getConnection();
        ps = con.prepareStatement(sql);
        ps.setString(1, cliente.getNro_documento());
        ps.setString(2, cliente.getNombres());
        ps.setString(3, cliente.getDireccion());
        ps.setString(4, cliente.getTelefono());
        ps.setString(5, cliente.getEmail());
        ps.executeUpdate();
        con.close(); // CERRAR LA CONEXIÓN A LA BASE DE DATOS
    } catch (Exception e) {
        throw e;
    }
    return 1;
```

Fuente: Elaboración propia

Nota: Esta operación se realizó en todas las clases DAO.

Finalizada la refactorización de los métodos definidos en todas las clases DAO del aplicativo, las pruebas unitarias se ejecutaron correctamente.

Figura 49. Resultados de la ejecución de las pruebas unitarias en la case Cliente



Fuente: Elaboración propia

5.2. Discusión

Después del estudio de la técnica del desarrollo dirigido por pruebas conjuntamente con el framework JUnit, se determina que la técnica no simplemente abarca el testing de la aplicación, sino que conduce el diseño de la misma, mejorando el paradigma de desarrollo del software, obteniendo un código de calidad.

El framework JUnit, permite que los sistemas sean más adaptables al cambio debido a que es fácil detectar y aislar las pruebas unitarias realizadas en cada parte del sistema.

El uso de TDD según Storani (2008) se complica en situaciones en las que se requieren pruebas completas de funcionalidad del sistema, por ejemplo, en el uso de GUI's, programas que trabajan con Bases de Datos relacionales y algunos que trabajan en diferentes configuraciones de red.

Aplicar la técnica TDD involucra un cambio de paradigma de desarrollo de software, ya que requiere la escritura de pruebas unitarias antes de la escritura del código fuente, lo cual resulta bastante complicado para los desarrolladores en el corto tiempo de entrenamiento, no obstante, no se puede descartar la posibilidad de que, incrementando la experiencia en la escritura de casos de pruebas, se pueda hacer uso de esta técnica en el futuro tanto en el ámbito académico como en el ámbito laboral.

TDD puede utilizarse en las pruebas de regresión, es decir, los desarrolladores pueden usar TDD para determinar qué cambios alteraron las funcionalidades de la aplicación. A diferencia de las pruebas unitarias, TDD no realiza pruebas de forma aislada sino por el contrario cubre funcionalidades consecutivas y permite la reutilización del código a futuro.

CONCLUSIONES

A través de un profundo y riguroso análisis de la documentación existente sobre la metodología de Desarrollo de Software Orientado por Pruebas (TDD), se concluye que esta metodología es un marco de trabajo que se centra en el diseño e implementación de pruebas unitarias, que generalmente parten de la especificación de requerimientos funcionales del software o en el caso de la Metodología Scrum parte de las historias de Usuario. TDD puede integrarse con cualquier Framework, en el caso de Junit, se determinó que su uso no abarca únicamente el testeo de la aplicación, sino que sirve de la base para el diseño de la misma permitiendo verificar que la lógica del código esté bien y que superará satisfactoriamente dichas pruebas.

La ejecución de las pruebas unitarias que se realizaron como parte de la implementación metodología de desarrollo orientado por pruebas (TDD) permitieron esclarecer correctamente la lógica de negocio del software, dando una pincelada inicial de la visión del software final; al finalizar la implementación de TDD, las pruebas unitarias se convierten en funcionalidades del software. Mediante estas pruebas, se dividió al código en fragmentos pequeños, donde el equipo desarrollador pudo probar distintas partes del software sin esperar a que otras estén implementadas.

Se desarrolló e implementó el sistema de control de inventario para el Almacén Alejandra utilizando el código refactorizado, el cual fue el resultado de la implementación de las pruebas unitarias definidas en a través de la Metodología TDD. Scrum y TDD encajaron de manera consistente en el desarrollo del caso práctico, permitiendo tener el control y la óptima distribución de las actividades de trabajo, a la vez permitió al equipo desarrollador centrarse en actividades prioritarias, dando como resultado una aplicación robusta y a la vez amigable con el usuario final. La combinación de Java y MySQL

RECOMENDACIONES

La metodología TDD es un marco de trabajo centrado en la ejecución de pruebas unitarias las cuales permiten tener un código funcional independientemente del Framework y metodología con las que se esté trabajando, sin embargo, es recomendable utilizar usar TDD con una metodología ágil ya sea Scrum o XP.

Al usar TDD es recomendable crear las pruebas unitarias antes de la implementación del sistema, ya que si se realizan las mismas después del desarrollo se está orientando el diseño hacia las prácticas tradicionales, por lo que se pierde las principales características de la implementación de la metodología TDD.

Cuando se dirige TDD hacia pequeñas pruebas de aceptación es importante dividir el proyecto por módulos de funcionalidad, de esta forma se cubre el testing de todo el sistema, y se evita que este proceso se torne complejo y sea necesaria la aplicación de herramientas adicionales de pruebas como Test Studio.

El sistema de control de inventario para el Almacén Alejandra es una aplicación Desktop la cual puede incrementar sus funcionalidades mediante el desarrollo de un aplicativo móvil que gestione el inventario de forma remota. Además, debido al constante nacimiento de nuevas tecnologías es recomendable implementar APIs que se integren con futuros aplicativos.

BIBLIOGRAFÍA

Abenza, P. G. (30 de Junio de 2015). *Google Libros.* Obtenido de Google Libros: https://books.google.com.ec/books?hl=es&lr=&id=4v8QCgAAQBAJ&oi=fnd&pg=PP1&dq=related:1J5QvpYXGFAJ:scholar.google.com/&ots=le5W2wmZqs&sig=njE8bi8I-f_vXJQcAIykKdm_5Vs&redir_esc=y#v=onepage&q&f=false

Adewole, A. (2018). *C# and .NET Core Test Driven Development: Dive into TDD to create flexible, maintainable, and production-ready .NET Core applications.* Packt Publishing.

Adewole, A. (18 de Mayo de 2018). *Google Libros.* Obtenido de Google Libros: https://books.google.com.ec/books?id=kF9dDwAAQBAJ&printsec=frontcover&dq=C%23+and+.NET+Core+Test+Driven+Development&hl=es&sa=X&ved=2ahUKEwiZ3-eInrfrAhVJiFkKHa6lAv0Q6AEwAHoECAMQAg#v=onepage&q=C%23%20and%20.NET%20Core%20Test%20Driven%20Development&f=false

Amaya, Y. D. (2013). Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual. *Revista de Tecnología - Journal Technology*, *12*(2), 111-124.

Beck, K. (2002). *Test Driven Development: By Example.* Addison-Wesley.

Blé, C. (2020). *Diseño Ágil con TDD.* Lean Mind. Obtenido de https://leanpub.com/tdd-en-castellano

Casillas, L. A., Ginestà, M. G., & Pérez, Ó. (2014). Bases de datos en MySQL. *Universitat oberta de Catalunya*.

Castro, D. V. (2016). *Sistema de capacitación virtual para la implementación de la metodología de desarrollo de software test driven devolopmen. [Tesis de Pregrado]*. Universidad Central del Ecuador. Obtenido de http://www.dspace.uce.edu.ec/bitstream/25000/7752/1/T-UCE-0011-298.pdf

Chile, D. (19 de Agosto de 2019). *defontana*. Obtenido de defontana: https://www.defontana.com/cl/como-funciona-un-sistema-de-

inventario/#:~:text=En%20l%C3%ADneas%20generales%2C%20un%20sistema,costo%20de%20los%20productos%20vendidos.

Farcic, V., & García, A. (2015). *Test-Driven Java Development.* Packt Publishing.

Gałęzowski, G. (2016). *Test-Driven Development: Extensive Tutorial.* Lean Publishing. Obtenido de https://openlibra.com/es/book/download/test-driven-development-extensive-tutorial

Garrido, P. (2015). *Comenzando a programar con JAVA (Spanish Edition).* Universitas Miguel Hernández.

Grenning , J. (2020). *Test Driven Development for Embedded C: Building Hihg Quality Embedded Software (Pragmatic Programmers).* Pragmatic Bookshelf.

Grenning, J. W. (2011). *Test Driven Development for Embedded C.* Pragmatic bookshelf.

Guerra, O. P., & Lema, L. E. (2018). *Implementar un sistema para procesar los datos que se levantan en el inventario de salud con información de las afecciones a la salud, clasificación c10, utilizando una plataforma java, postgresql. [Tesis de pregrado].* Obtenido de https://dspace.ups.edu.ec/handle/123456789/15335

Haq, M. Z. (2017). *Angular Test-Driven Development* (Vol. 2nd Revised Edition). Packt Publishing.

Laínez, J. R. (2015). *Desarrollo de Software ÁGIL: Extreme Programming y Scrum.* Createspace Independent Publishing Platform.

Lee , G. (16 de Octubre de 2016). *Tipos de pruebas de software: diferencias y ejemplos.* Obtenido de https://www.loadview-testing.com/es/blog/tipos-de-pruebas-de-software-diferencias-y-ejemplos/

Microsoft Developers Network. (2021). *Conceptos básicos de las pruebas unitarias.* Obtenido de https://docs.microsoft.com/es-es/visualstudio/test/unit-test-basics?view=vs-2019&redirectedfrom=MSDN&viewFallbackFrom=vs-2015

Orozco Alvarez, E. A. (2018). *Desarrollo del módulo PIM-PSM versión 5.0 de la herramienta jMDA [Tesis de pregrado].* Universidad Central "Martha Abreu" de

Las Villas. Obtenido de https://dspace.uclv.edu.cu/bitstream/handle/123456789/9953/Tesis%20Eduardo%20Orozco.pdf?sequence=1&isAllowed=y

Parra, D. C., & Ramírez, J. M. (2018). *DISEÑO DESARROLLO E IMPLEMENTACION DE SOFTWARE Y APLICATIVO MOVIL PARA LA ADMINISTRACION Y GESTION DE VENTA Y PREVENTA DE LA DISTRIBUIDORA BUITRAGO.* Universidad Piloto de Colombia - Ingeniería de Sistemas.

Pozo, T., Aucancela, C., Hinojosa, C., & Abdelrahaman, A. (2011). Sistema Web de Asignación de Aulas de los Laboratorios de Computación de la ESPE, Aplicando la Metodología Agile Unified Process (AUP), utilizando el Framework Junit. *Revista DECC Report, Tendencias en Computación, 3*(1), 6-14. Obtenido de https://journal.espe.edu.ec/ojs/index.php/geeks/article/download/252/229

Ress, A. P., de Oliveira, R., & Salerno, M. S. (2013). Test-Driven Development as an Innovation Value Chain. *Journal of Technology Management & Innovation, 8*(1). Obtenido de http://dx.doi.org/10.4067/S0718-27242013000300010

Salazar, J. C., Tovar, Á., Linares, J. C., Lozano, A., & Valbuena, Y. L. (2018). Scrum versus XP: similitudes. *TIA, 6*(2), 29-37. Obtenido de https://revistas.udistrital.edu.co/index.php/tia/article/view/10496

Storani, M. (20 de Junio de 2008). *TDD – Test Driven Development*. Obtenido de https://mauriziostorani.wordpress.com/2008/06/20/tdd-test-driven-development/

Sznajdleder, P. (2013). *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones* (Segunda ed.). Buenos Aires: Alfaomega Grupo Editor Argentino. Obtenido de https://www.academia.edu/14584688/Java_a_fondo_Sznajdleder_Pablo

Sznajdleder, P. A. (13 de Diciembre de 2016). *Google Libros.* Obtenido de Google Libros: https://books.google.com.ec/books?hl=es&lr=&id=WcL2DQAAQBAJ&oi=fnd&pg=PT5&dq=lenguaje+de+programaci%C3%B3n+java&ots=iPBgEDHxN5&sig=8WN0VR0vRuH7Q8WYOeoSyNUXnMg&redir_esc=y#v=onepage&q=lenguaje%20de%20programaci%C3%B3n%20java&f=false

Torres-Corredor, O. I. (2017). *Aplicación y evaluación de la metodología desarrollo orientado por pruebas (TDD), caso de estudio: spot.co. [Tesis de Maestría]*. Universidad Internacional de la Rioja. Obtenido de https://reunir.unir.net/handle/123456789/6562

Vaca et al. (2014). Test-Driven Development - Una aproximación para entender su utilidad en el proceso de desarrollo de Software. *WICC 2014 XVI Workshop de Investigadores en Ciencias de la Computación*, 570 - 574. Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/41604/Documento_completo.pdf?sequence=1

Veintimilla, A. J., & Cuenca, L. F. (2014). *Estudio de la técnica Test Driven Development (TDD) y desarrollo del sistema para la administración de consultorios médicos. [Tesis de pregrado]*. Universidad de las Fuerzas Armadas ESPE. . Obtenido de https://repositorio.espe.edu.ec/bitstream/21000/9094/1/T-ESPE-048066.pdf

ANEXOS

Anexo 1: Manual de usuario

MANUAL DE USUARIO

SOFTWARE DE CONTROL DE INVENTARIO DEL ALMACÉN ALEJANDRA

Versión 1

Importante: *Este Programa se instala en un sólo computador. Una vez instalado NO DEBE borrar el directorio, ni cambiar su nombre, ni moverlo dentro del disco duro donde fue instalado, ni copiarlo en otro computador, pues no podrá ser desinstalado y por lo tanto no podrá volver a instalarlo nuevamente y la única forma de hacerlo será modificando el programa instalador.*

Nota: *El Programa puede tener perfeccionamientos y actualizaciones que no están señaladas en este manual, debido a que estos cambios se realizan con mayor rapidez en el software que en el manual.*

I. Ingreso al sistema

Al dar doble click sobre el ícono de la aplicación, se presenta el formulario de inicio de sesión, donde el usuario debe ingresar sus credenciales (usuario y contraseña).

Figura 50. Formulario "Inicio de sesión"

Fuente: Autores

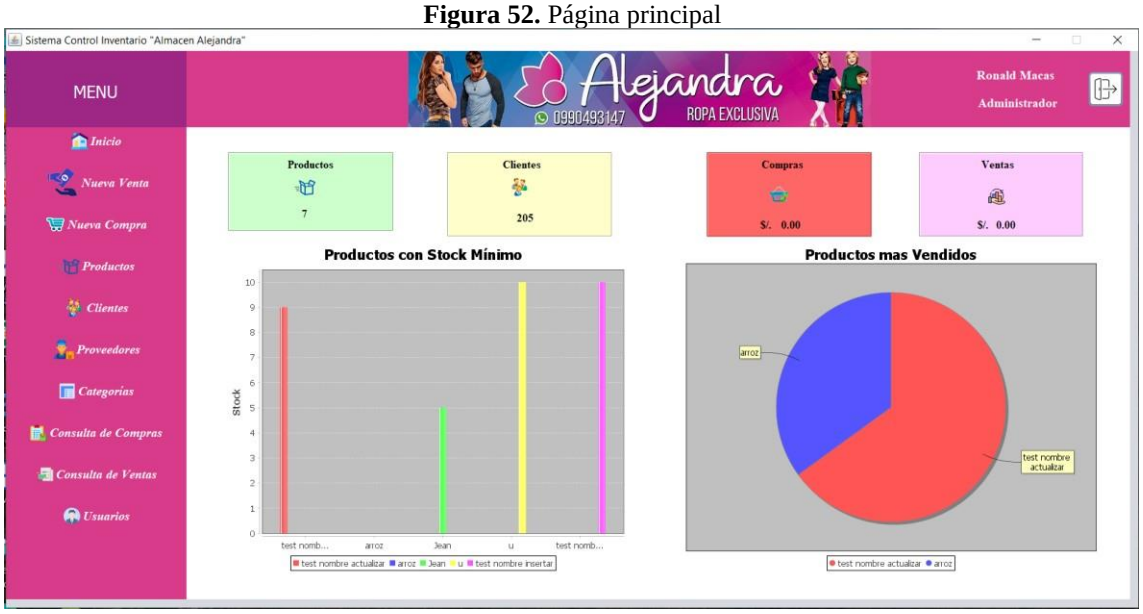
En caso de ingresar datos erróneos, la aplicación presentará un cuadro de diálogo indicando que las credenciales son incorrectas.

Figura 51. Advertencia del ingreso de datos erróneos

Fuente: Autores

Al ingresar los datos correctos, el software mostrará el formulario central. En donde se presenta el menú principal y las estadísticas concernientes a Productos, Clientes, Compras y Ventas.

Figura 52. Página principal



Fuente: Autores

En las siguientes secciones, se describen cada una de las funcionalidades del menú principal.

II. Compras

El acceso al módulo Ventas se lo realiza mediante un *click* sobre el botón 

posteriormente se presenta el siguiente formulario.

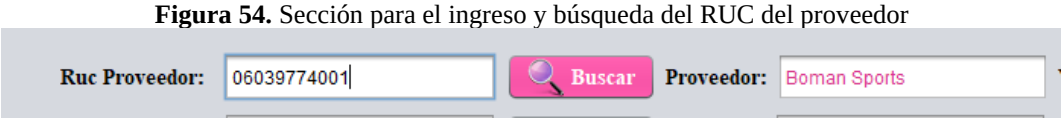
Figura 53. Formulario para registrar compras

Fuente: Autores

En el formulario realizar compras, el usuario debe ingresar el RUC del proveedor y presionar el botón buscar 

inmediatamente se presentará el nombre del proveedor, si se encuentra registrado.

Figura 54. Sección para el ingreso y búsqueda del RUC del proveedor



Fuente: Autores

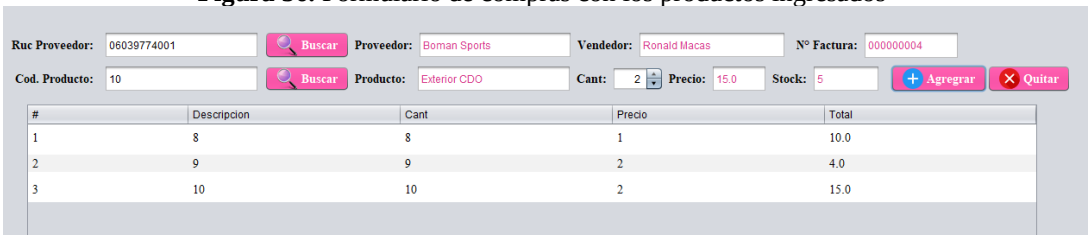
De forma similar, ingresar el código del producto presionar en el botón buscar y se visualiza la descripción del producto.

Figura 55. Formulario de compras con los productos ingresados

Fuente: Autores

A continuación, ingresar la cantidad de productos que desea comprar y seleccionar la opción **agregar**. Este proceso se repite hasta tener la cantidad de productos deseada.

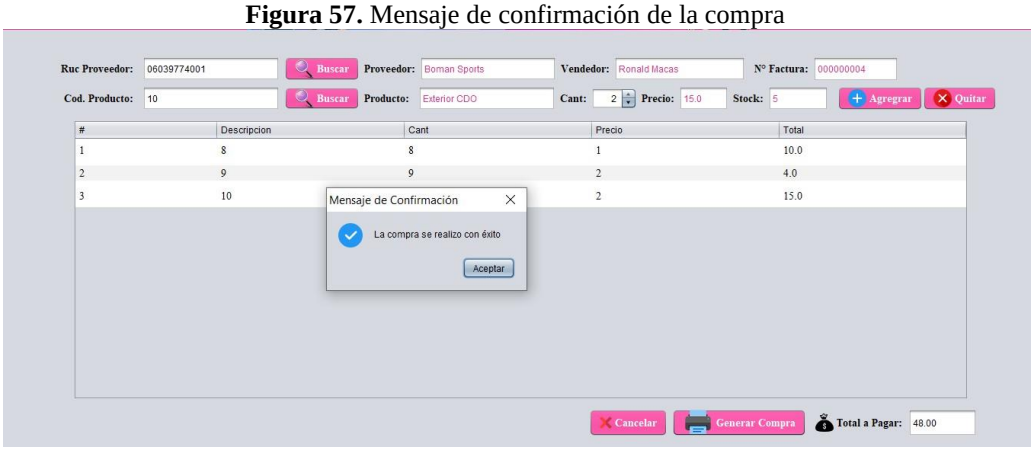
Figura 56. Formulario de compras con los productos ingresados



Fuente: Autores

Para finalizar la compra seleccionar el botón 

Figura 57. Mensaje de confirmación de la compra



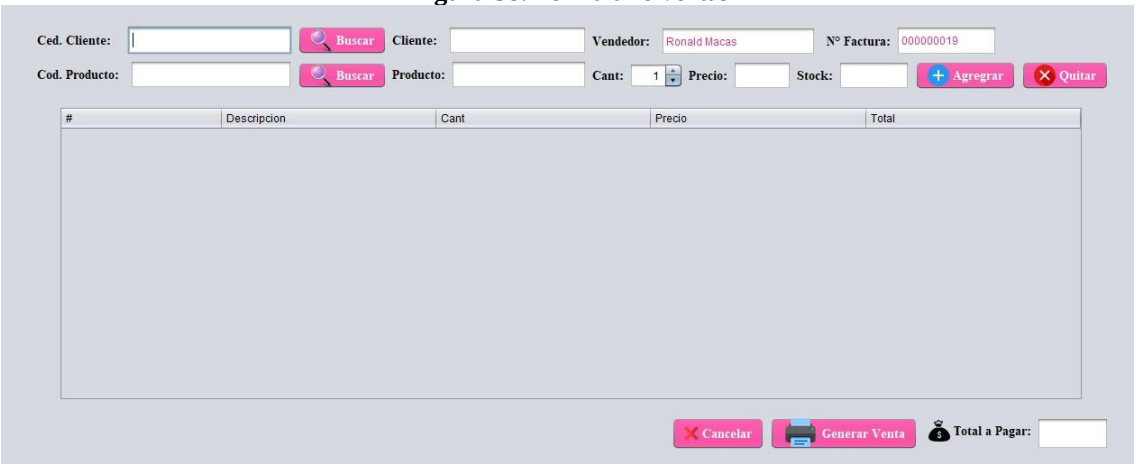
Fuente: Autores

Si el proceso fue exitoso, se presentará el diálogo de confirmación.

III. Venta

El módulo de ventas tiene un funcionamiento similar al módulo Compras, para accederlo, dar un click en el submenú: 

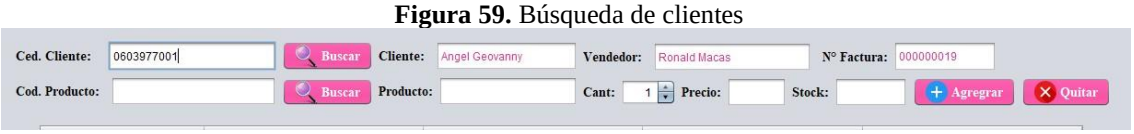
Figura 58. Formulario ventas



Fuente: Autores

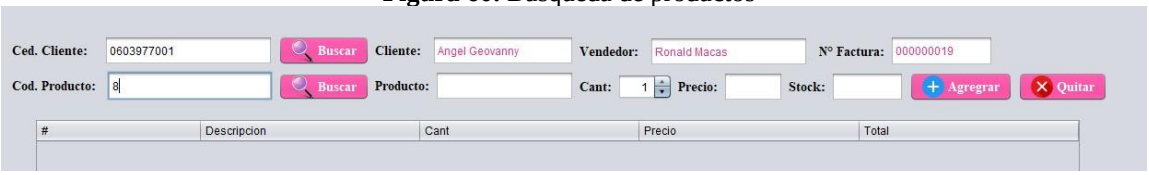
En la parte superior del formulario, se presentan dos campos para ingresar la cédula de ciudadanía del cliente y el botón de búsqueda, de forma similar se presenta un campo para la búsqueda de los productos a través de su código.

Figura 59. Búsqueda de clientes



Fuente: Autores

Figura 60. Búsqueda de productos



Fuente: Autores

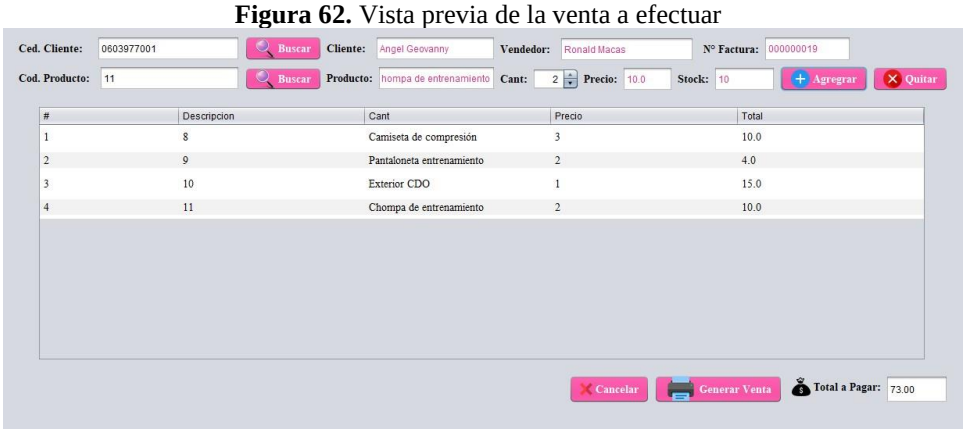
Posterior a la búsqueda de los productos, se presentan el precio y el stock disponible. A continuación, el usuario debe ingresar la cantidad de productos que desea vender.

Figura 61. Ingreso de la cantidad de productos a vender

Fuente: Autores

Finalizado el ingreso de productos y sus respectivos precios, el usuario del sistema procederá a generar la venta, presionado el botón 

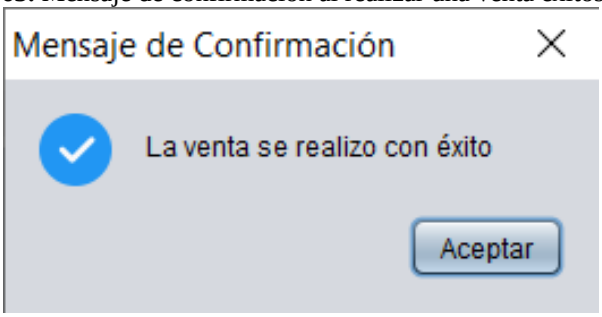
Figura 62. Vista previa de la venta a efectuar



Fuente: Autores

Cuando el proceso se ha realizado exitosamente se presenta el mensaje de confirmación.

Figura 63. Mensaje de confirmación al realizar una venta exitosamente



Fuente: Autores

IV. Categorías

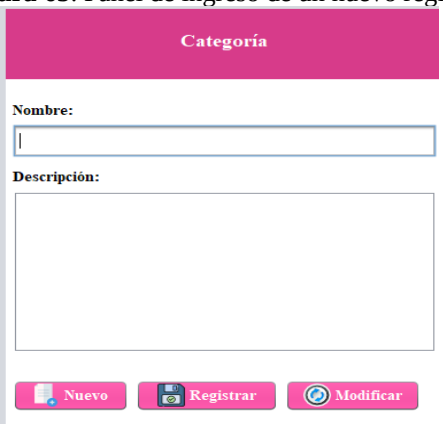
A través de esta opción el administrador del sistema, tiene la posibilidad de agrupar los productos mediante categorías. El acceso a este módulo se lo realiza a través del submenú 

Figura 64. Módulo Gestión de Categoría

Fuente: Autores

El ingreso de una nueva categoría se la realiza a través del siguiente formulario, el cual se presenta luego de presionar la opción 

Figura 65. Panel de ingreso de un nuevo registro



Fuente: Autores

Por cada nueva categoría se pide ingresar el Nombre y la descripción.

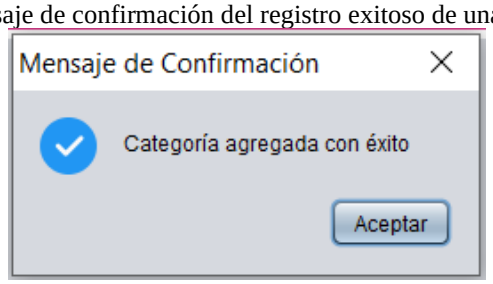
Figura 66. Panel previo al registro de datos

Fuente: Autores

Finalizado el ingreso del nombre y la categoría, presionar el botón 

se presentará el respectivo mensaje de confirmación.

Figura 67. Mensaje de confirmación del registro exitoso de una nueva categoría



Fuente: Autores

En el siguiente gráfico, se observa la categoría ingresada anteriormente.

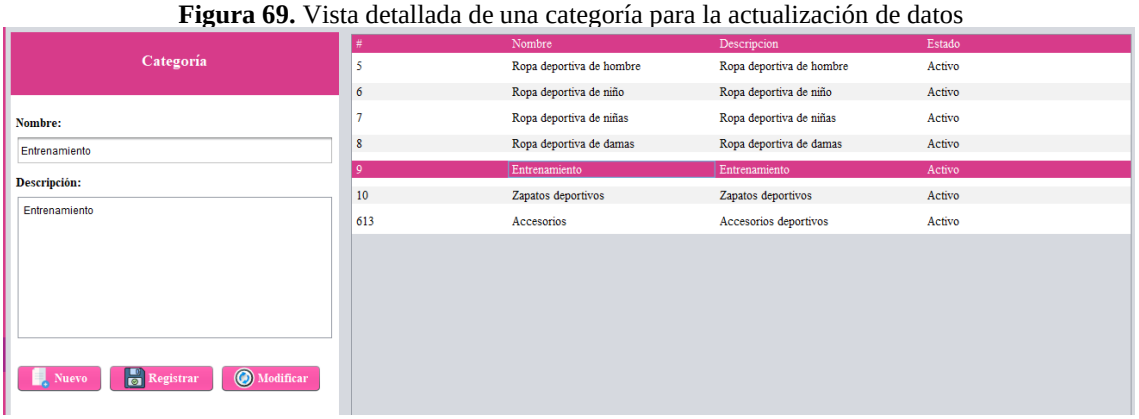
Figura 68. Listado de categorías

#	Nombre	Descripcion	Estado
5	Ropa deportiva de hombre	Ropa deportiva de hombre	Activo
6	Ropa deportiva de niño	Ropa deportiva de niño	Activo
7	Ropa deportiva de niñas	Ropa deportiva de niñas	Activo
8	Ropa deportiva de damas	Ropa deportiva de damas	Activo
9	Entrenamiento	Entrenamiento	Activo
10	Zapatos deportivos	Zapatos deportivos	Activo
613	Accesorios	Accesorios deportivos	Activo

Fuente: Autores

Para actualizar una categoría existente, seleccionar la categoría a actualizar y presionar el botón 

Figura 69. Vista detallada de una categoría para la actualización de datos

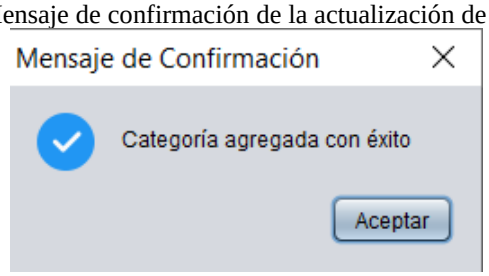


Fuente: Autores

Una vez que se hayan modificado los datos, presionar nuevamente el botón 

para registrar los cambios en la base de datos y se presentará el mensaje de confirmación.

Figura 70. Mensaje de confirmación de la actualización de una categoría



Fuente: Autores

V. Productos

Los productos se encuentran almacenados por categorías, el acceso a este módulo se lo obtiene mediante el botón 

seguidamente se presentará el formulario gestión de productos.

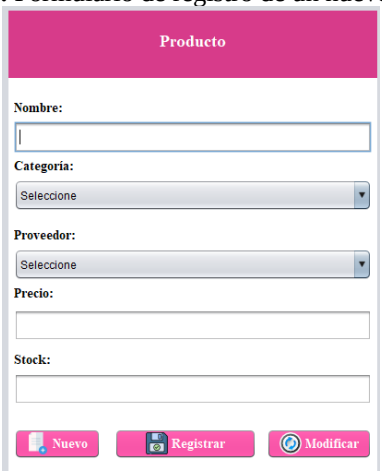
Figura 71. Módulo Gestión de Productos

Fuente: Autores

El registro de un nuevo producto se lo realiza a través del botón 

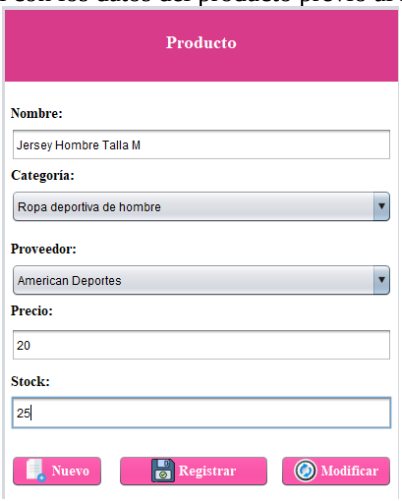
posteriormente se presenta el formulario para ingresar los datos del producto como son, nombre, categoría, proveedor, precio y las unidades en stock.

Figura 72. Formulario de registro de un nuevo producto



Fuente: Autores

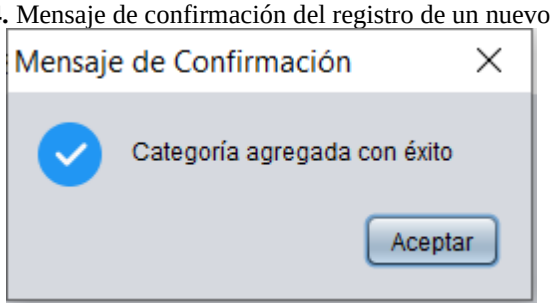
Figura 73. Panel con los datos del producto previo al registro de datos



Fuente: Autores

Finalizado el llenado de los campos, se presentará el mensaje de confirmación.

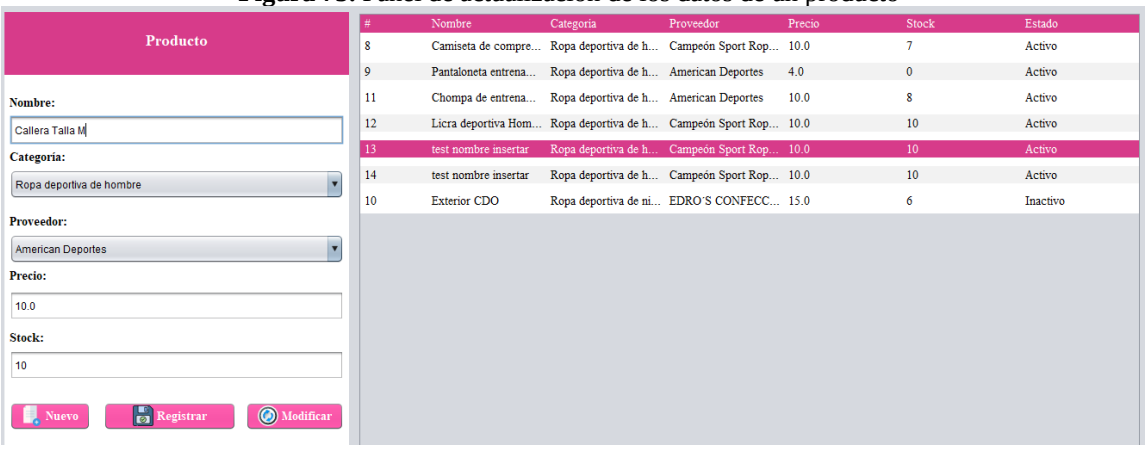
Figura 74. Mensaje de confirmación del registro de un nuevo producto



Fuente: Autores

Para la actualización o edición de un producto, en primer lugar, seleccionar el producto a modificar, y presionar el botón 

Figura 75. Panel de actualización de los datos de un producto

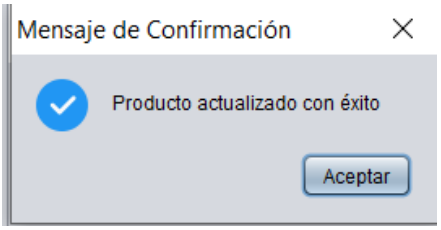


Fuente: Autores

Finalizado el ingreso de los nuevos datos, los datos se registrarán al presionar en el botón 

y se presentará el respectivo mensaje de confirmación.

Figura 76. Mensaje de confirmación de la actualización de los datos de un producto



Fuente: Autores

Los datos se visualizarán inmediatamente en el formulario gestión de productos.

Figura 77. Listado de productos

#	Nombre	Categoria	Proveedor	Precio	Stock	Estado
8	Camiseta de compre...	Ropa deportiva de h...	Campeón Sport Rop...	10.0	7	Activo
9	Pantaloneta entrena...	Ropa deportiva de h...	American Deportes	4.0	0	Activo
11	Chompa de entrena...	Ropa deportiva de h...	American Deportes	10.0	8	Activo
12	Licra deportiva Hom...	Ropa deportiva de h...	Campeón Sport Rop...	10.0	10	Activo
13	Callera Talla M	Ropa deportiva de h...	American Deportes	10.0	10	Activo
14	test nombre insertar	Ropa deportiva de h...	Campeón Sport Rop...	10.0	10	Activo
10	Exterior CDO	Ropa deportiva de ni...	EDRO´S CONFECC...	15.0	6	Inactivo

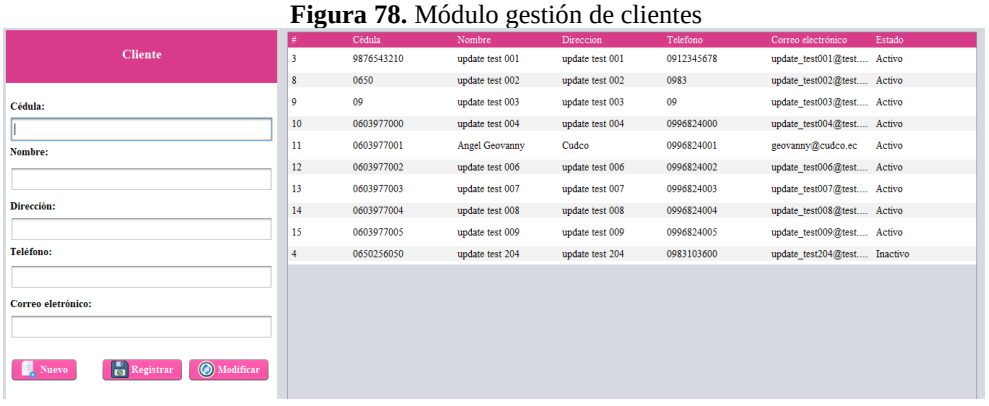
Fuente: Autores

VI. Clientes

Este módulo tiene el mismo principio que los módulos anteriores, el acceso a este módulo es a través del botón 

. Al presionar este botón se despliega el módulo gestión de usuarios.

Figura 78. Módulo gestión de clientes



Fuente: Autores

El acceso al formulario de ingreso es a través del botón 

en el formulario de registro llenar los campos cédula, nombres, dirección, teléfono y correo electrónico.

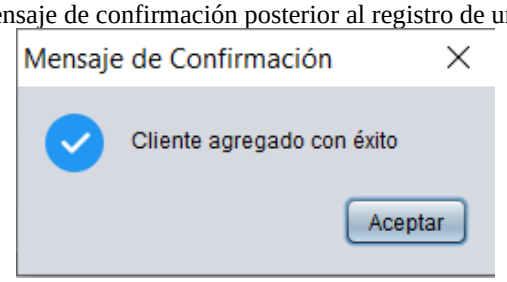
Figura 79. Panel de ingreso de datos del cliente

Fuente: Autores

Finalizado el ingreso de datos en cada campo, dar click en el botón 

esta acción presentará el mensaje de confirmación.

Figura 80. Mensaje de confirmación posterior al registro de un nuevo cliente



Fuente: Autores

Para la modificación de los datos de un cliente registrado previamente, primero debe seleccionar la fila correspondiente al cliente a modificar y presionar el botón 

Figura 81. Listado de clientes

#	Cédula	Nombre	Direccion	Telefono	Correo electrónico	Estado
3	9876543210	update test 001	update test 001	0912345678	update_test001@test....	Activo
8	0650	update test 002	update test 002	0983	update_test002@test....	Activo
9	09	update test 003	update test 003	09	update_test003@test....	Activo
10	0603977000	update test 004	update test 004	0996824000	update_test004@test....	Activo
11	0603977001	Angel Geovanny	Cudco	0996824001	geovanny@cudco.ec	Activo
12	0603977002	update test 006	update test 006	0996824002	update_test006@test....	Activo
13	0603977003	update test 007	update test 007	0996824003	update_test007@test....	Activo
14	0603977004	update test 008	update test 008	0996824004	update_test008@test....	Activo
15	0603977005	update test 009	update test 009	0996824005	update_test009@test....	Activo
211	0601233219	Juan Pérez	Guano	0996824308	juan.perez@gmail.com	Activo
4	0650256050	update test 204	update test 204	0983103600	update_test204@test....	Inactivo

Fuente: Autores

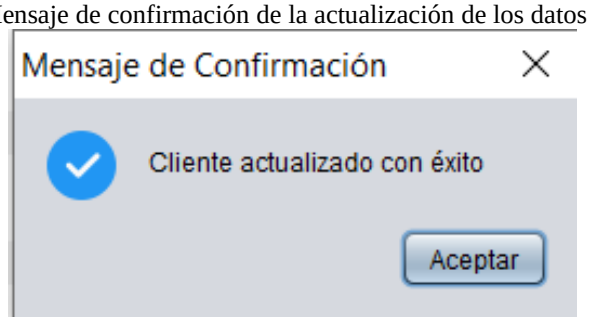
En el panel de la izquierda se presentarán los datos del cliente seleccionado.

Figura 82. Selección del cliente a modificar

Fuente: Autores

Una vez que se han ingresado los nuevos campos, éstos se guardarán al presionar el botón  lo cual se ratificará al presentarse el mensaje de confirmación.

Figura 83. Mensaje de confirmación de la actualización de los datos de un cliente



Fuente: Autores

Inmediatamente los datos se visualizarán en el listado de clientes en el formulario gestión de clientes.

Figura 84. Listado de clientes

#	Cédula	Nombre	Direccion	Telefono	Correo electrónico	Estado
3	9876543210	update test 001	update test 001	0912345678	update_test001@test....	Activo
8	0650	update test 002	update test 002	0983	update_test002@test....	Activo
9	09	update test 003	update test 003	09	update_test003@test....	Activo
10	0603977000	update test 004	update test 004	0996824000	update_test004@test....	Activo
11	0603977001	Angel Cudco	Salcedo	0996824001	geovanny@cudco.ec	Activo
12	0603977002	update test 006	update test 006	0996824002	update_test006@test....	Activo
13	0603977003	update test 007	update test 007	0996824003	update_test007@test....	Activo
14	0603977004	update test 008	update test 008	0996824004	update_test008@test....	Activo
15	0603977005	update test 009	update test 009	0996824005	update_test009@test....	Activo
211	0601233219	Juan Pérez	Guano	0996824308	juan.perez@gmail.com	Activo
4	0650256050	update test 204	update test 204	0983103600	update_test204@test....	Inactivo

Fuente: Autores

VII. Proveedores

El ingreso al módulo proveedores, se lo realiza a través del botón 

. Para ingresar un nuevo proveedor se dispone de un panel con los campos, RUC, nombre, dirección, teléfono y correo electrónico, los cuales son campos obligatorios.

Figura 85. Módulo Gestión de Proveedores

Fuente: Autores

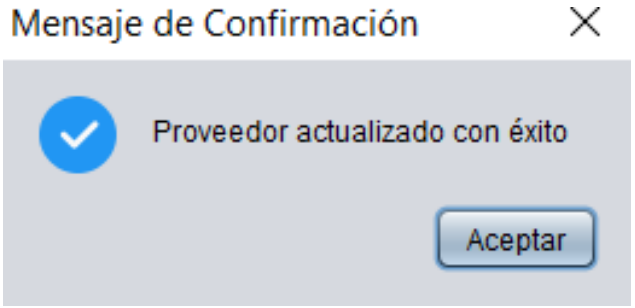
Figura 86. Registro de los datos de un nuevo proveedor

Fuente: Autores

Finalizado el ingreso de datos, estos se almacenarán en la base de datos al presionar el botón 

se desplegará el mensaje de confirmación.

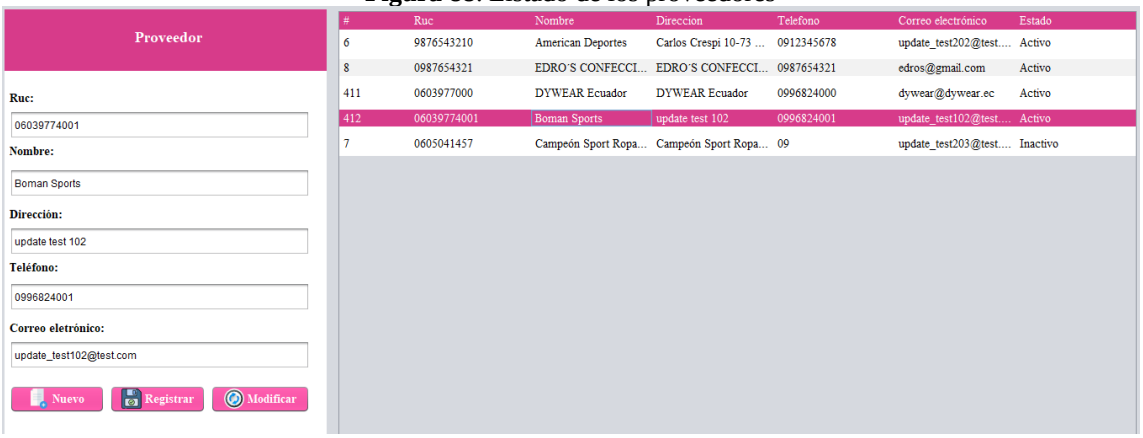
Figura 87. Mensaje de confirmación del registro de los datos del proveedor



Fuente: Autores

Pata la actualización de datos, seleccionar la fila que contiene los datos del usuario a modificar e inmediatamente en el panel de la izquierda se mostrarán los datos del cliente.

Figura 88. Listado de los proveedores

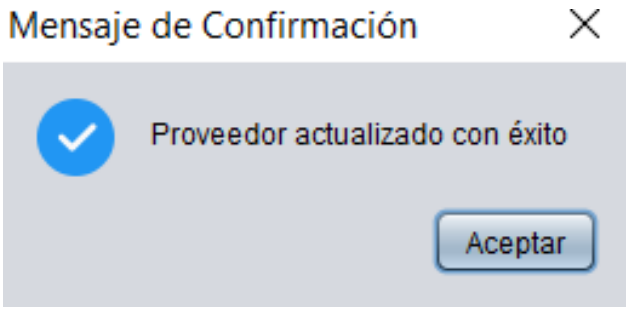


Fuente: Autores

Los cambios tendrán efecto al presionar el botón 

y se presentará el mensaje de confirmación.

Figura 89. Mensaje de confirmación de la actualización de datos de un proveedor



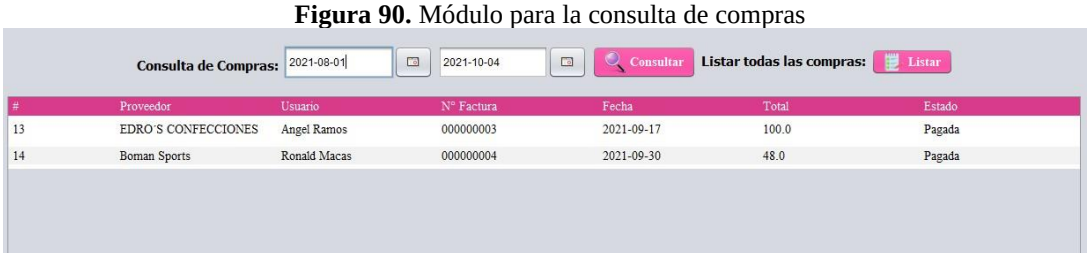
Fuente: Autores

VIII. Consulta de compras

A través del botón 

se presenta un módulo de consulta, donde se puede consultar las compras realizadas dentro de un rango de fechas.

Figura 90. Módulo para la consulta de compras



Fuente: Autores

En primer lugar, ingresar la fecha de inicio.

Figura 91. Campo para el ingreso de la fecha de inicio

Fuente: Autores

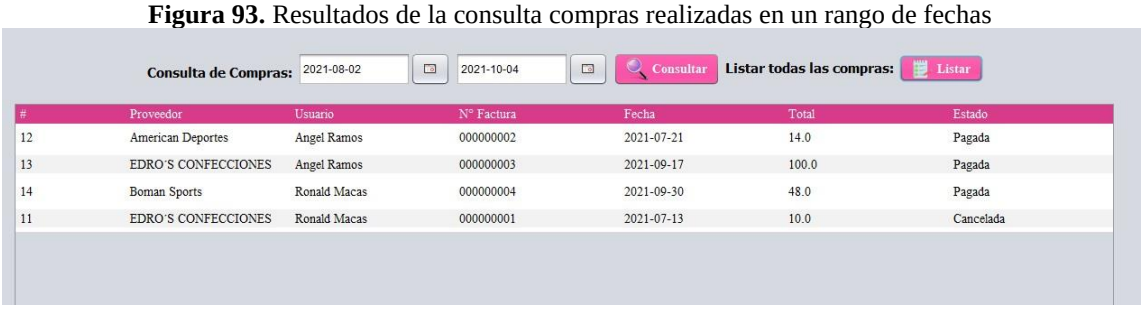
De forma similar ingresar la fecha de finalización.

Figura 92. Campo para ingresar la fecha de finalización

Fuente: Autores

Para obtener los resultados esperados, presionar el botón 

Figura 93. Resultados de la consulta compras realizadas en un rango de fechas



#	Proveedor	Usuario	N° Factura	Fecha	Total	Estado
12	American Deportes	Angel Ramos	000000002	2021-07-21	14.0	Pagada
13	EDRO´S CONFECCIONES	Angel Ramos	000000003	2021-09-17	100.0	Pagada
14	Boman Sports	Ronald Macas	000000004	2021-09-30	48.0	Pagada
11	EDRO´S CONFECCIONES	Ronald Macas	000000001	2021-07-13	10.0	Cancelada

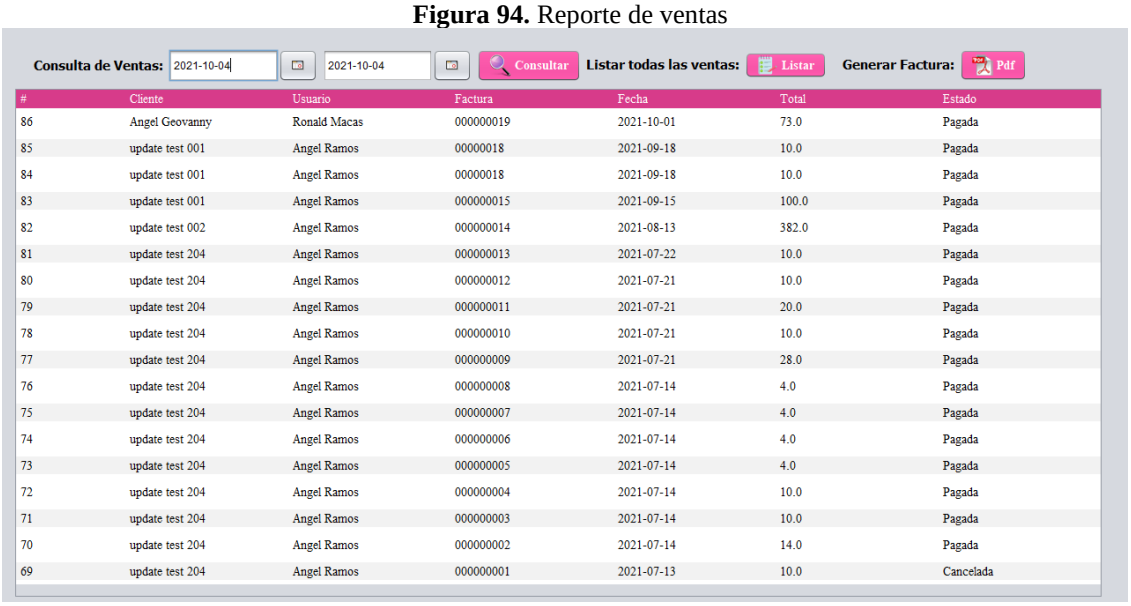
Fuente: Autores

IX. Consulta de Ventas

El acceso a este módulo se lo realiza a través del submenú 

. En este módulo el administrador puede consultar las ventas realizadas en un rango de fechas.

Figura 94. Reporte de ventas



#	Cliente	Usuario	Factura	Fecha	Total	Estado
86	Angel Geovanny	Ronald Macas	000000019	2021-10-01	73.0	Pagada
85	update test 001	Angel Ramos	00000018	2021-09-18	10.0	Pagada
84	update test 001	Angel Ramos	00000018	2021-09-18	10.0	Pagada
83	update test 001	Angel Ramos	000000015	2021-09-15	100.0	Pagada
82	update test 002	Angel Ramos	000000014	2021-08-13	382.0	Pagada
81	update test 204	Angel Ramos	000000013	2021-07-22	10.0	Pagada
80	update test 204	Angel Ramos	000000012	2021-07-21	10.0	Pagada
79	update test 204	Angel Ramos	000000011	2021-07-21	20.0	Pagada
78	update test 204	Angel Ramos	000000010	2021-07-21	10.0	Pagada
77	update test 204	Angel Ramos	000000009	2021-07-21	28.0	Pagada
76	update test 204	Angel Ramos	000000008	2021-07-14	4.0	Pagada
75	update test 204	Angel Ramos	000000007	2021-07-14	4.0	Pagada
74	update test 204	Angel Ramos	000000006	2021-07-14	4.0	Pagada
73	update test 204	Angel Ramos	000000005	2021-07-14	4.0	Pagada
72	update test 204	Angel Ramos	000000004	2021-07-14	10.0	Pagada
71	update test 204	Angel Ramos	000000003	2021-07-14	10.0	Pagada
70	update test 204	Angel Ramos	000000002	2021-07-14	14.0	Pagada
69	update test 204	Angel Ramos	000000001	2021-07-13	10.0	Cancelada

Fuente: Autores

Para lo cual, ingresar la fecha de inicio.

Figura 95. Ingreso de la fecha de inicio

Fuente: Autores

Posteriormente ingresar la fecha de finalización.

Figura 96. Ingreso de la fecha de finalización

Fuente: Autores

Finalmente, para visualizar los resultados, presionar el botón 

y se presenta una tabla con el detalle de las ventas efectuadas.

Figura 97. Reporte de las ventas realizadas en un periodo de tiempo

Fuente: Autores

Anexo 2: Manual técnico

MANUAL TÉCNICO

SOFTWARE DE CONTROL DE INVENTARIO DEL ALMACÉN ALEJANDRA

Versión 1

I. REQUISITOS DEL SISTEMA

1.1. Requerimientos de hardware

- Equipo, teclado, mouse, monitor, dispositivo móvil.
- Memoria RAM 2 GB (equipo y dispositivo móvil).
- Tarjeta de red LAN y/o Wireless.
- Procesador 1.4 GHz.

1.2. Requerimientos de software

- Sistema operativo (Windows 7 en adelante).
- Java 8.0.
- Conexión internet local.
- Adobe Reader.

II. HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

2.1. JAVA

Es una herramienta de desarrollo orientada a objetos, fue diseñado para que no dependieran en muchas implementaciones, el cual permite a los desarrolladores ejecutar en cualquier dispositivo sin necesidad de recompilar el código, el cual se considera multiplataforma. (Parra & Ramírez, 2018)

2.2. Servidor de base de datos (MySQL)

Es uno de los más característicos y por tener la opción de código abierto a nivel mundial, siendo una de las más populares antes ORACLE y Microsoft SQL Server principalmente en entornos de desarrollo web. (Parra & Ramírez, 2018)

2.3. NetBeans

Permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las API de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. (Garrido, 2015)

Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

NetBeans IDE permite el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

III. INSTALACIÓN DE APLICACIONES

Requisitos generales pre-instalación para el sistema de escritorio

Para ejecutar el programa de escritorio se necesita de Java 8.0 instalado con las siguientes características para la ejecución del programa de escritorio.

- Soporte en procesador Intel 1.4.0 GHz o superior.
- Memoria RAM 2 GB en adelante.
- Espacio en disco: 124 MB
- El programa se descarga del siguiente enlace en la página oficial de Java https://www.java.com/es/download/.

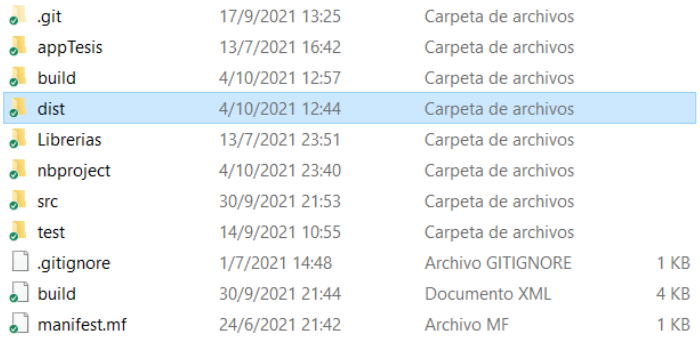
Figura 98. Descarga de Java 8.0

Fuente: Autores

Instalación y ejecución del programa de escritorio

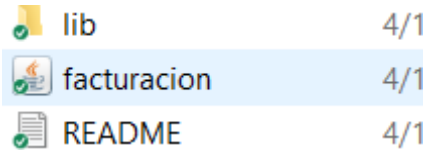
Luego de tener instalado Java 8.0, se procede a ejecutar el programa compilado, el cual se encuentra dentro de la carpeta *dist.*

Figura 99. Directorio del software de facturación en el computador



Fuente: Autores

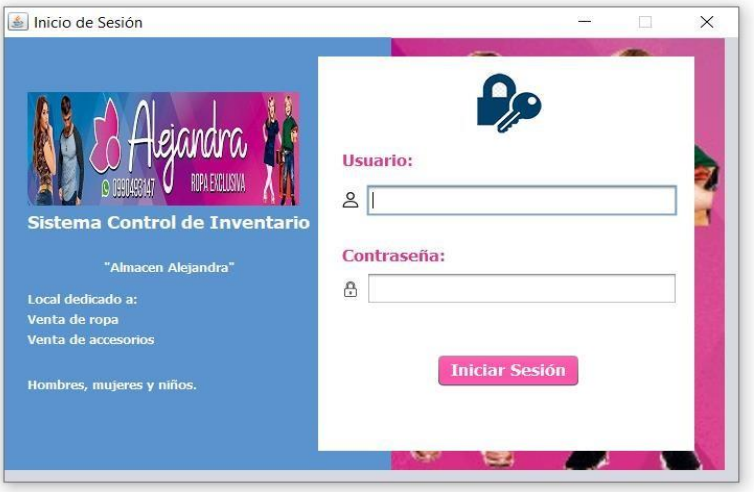
Figura 100. Ubicación del ejecutable



Fuente: Autores

Para confirmar la ejecución del programa de escritorio, se mostrará la interfaz de ingreso al sistema.

Figura 101. Formulario de ingreso al sistema



Fuente: Autores

Anexo 3: Fotografías de las reuniones con los usuarios

Anexo 4: Diccionario de datos

En el presente documento recolecta las características lógicas de los datos a usar en el desarrollo del sistema de control de inventario del almacén Alejandra.

[1] Tabla: Categoría

Tabla 22. Categoría

Columna	Tipo	Nulo	Predeterminado
idcategoria *(Primaria)*	int(11)	No	
Categoría	varchar(150)	Sí	*NULL*
descripcion	varchar(255)	Sí	*NULL*
estado	enum('Activo', 'Inactivo')	Sí	Activo

Fuente: Elaboración propia

Tabla 23. Constraints de categoría

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	idcategoria	3	A	No	

Fuente: Elaboración propia

[2] Tabla: Cliente

Tabla 24. Descripción del cliente

Columna	Tipo	Nulo	Predeterminado
idcliente *(Primaria)*	int(11)	No	
nro_documento	varchar(15)	Sí	*NULL*
nombres	varchar(100)	Sí	*NULL*
direccion	varchar(255)	Sí	*NULL*
telefono	varchar(10)	Sí	*NULL*
email	varchar(255)	Sí	*NULL*
estado	enum('Activo', 'Inactivo')	Sí	Activo

Fuente: Elaboración propia

Tabla 25. Constraints presentes en cliente

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	idcliente	2	A	No	
fk_persona_cliente	BTREE	No	No	direccion	2	A	Sí	

Fuente: Elaboración propia

[3] Tabla: Compra

Tabla 26. Campos compra

Columna	Tipo	Nulo	Predeterminado	Enlaces a
idcompra *(Primaria)*	int(11)	No		
idproveedor	int(11)	Sí	*NULL*	proveedor -> idproveedor
idusuario	int(11)	Sí	*NULL*	usuario -> idusuario
numeroserie	varchar(10)	Sí	*NULL*	
fecha	date	Sí	current_timestamp()	
impuesto	decimal(10,2)	Sí	12.00	
total	decimal(10,2)	Sí	*NULL*	
estado	enum('Pagada', 'Cancelada')	Sí	Pagada	

Fuente: Elaboración propia

Tabla 27. Constraints presentes en la compra

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	idcompra	0	A	No	
proveedor_id	BTREE	No	No	idproveedor	0	A	Sí	
idusuario	BTREE	No	No	idusuario	0	A	Sí	

Fuente: Elaboración propia

[4] Tabla: detalle_compra

Tabla 28. Campos detalle_compra

Columna	Tipo	Nulo	Predeterminado	Enlaces a
iddetallecompra *(Primaria)*	int(11)	No		
idcompra	int(11)	Sí	*NULL*	compra -> idcompra
idproducto	int(11)	Sí	*NULL*	producto -> idproducto
cantidad	decimal(10,2)	Sí	*NULL*	
precio	decimal(10,2)	Sí	*NULL*	

Fuente: Elaboración propia

Tabla 29. Constraints presentes en detalle_compra

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	iddetallecompra	0	A	No	
compra_id	BTREE	No	No	idcompra	0	A	Sí	
producto_id	BTREE	No	No	idproducto	0	A	Sí	

Fuente: Elaboración propia

[5] Tabla: detalle_venta

Tabla 30. Campos de detalle_venta

Columna	Tipo	Nulo	Predeterminado	Enlaces a
iddetalleventa *(Primaria)*	int(11)	No		
idventa	int(11)	Sí	*NULL*	venta -> idventa
idproducto	int(11)	Sí	*NULL*	producto -> idproducto
cantidad	int(11)	Sí	*NULL*	
precio	decimal(10,2)	Sí	*NULL*	

Fuente: Elaboración propia

Tabla 31. Constraints presentes en detalle_venta

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	iddetalleventa	9	A	No	
venta_id	BTREE	No	No	idventa	9	A	Sí	
producto_id	BTREE	No	No	idproducto	4	A	Sí	

Fuente: Elaboración propia

[6] Tabla: producto

Tabla 32. Campos producto

Columna	Tipo	Nulo	Predeterminado	Enlaces a
idproducto *(Primaria)*	int(11)	No		
nombre	varchar(100)	Sí	*NULL*	
idcategoria	int(11)	Sí	*NULL*	categoria -> idcategoria
idproveedor	int(11)	Sí	*NULL*	proveedor -> idproveedor
precio	decimal(10,2)	Sí	*NULL*	
stock	int(11)	Sí	*NULL*	
estado	enum('Activo', 'Inactivo')	Sí	Activo	

Fuente: Elaboración propia

Tabla 33. Constraints presentes en producto

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	idproducto	3	A	No	
fk_cateogria_producto	BTREE	No	No	precio	3	A	Sí	
k_unidad_producto	BTREE	No	No	stock	3	A	Sí	
idcategoria	BTREE	No	No	idcategoria	3	A	Sí	
idproveedor	BTREE	No	No	idproveedor	3	A	Sí	

Fuente: Elaboración propia

[7] Tabla: proveedor

Tabla 34. Campos proveedor

Columna	Tipo	Nulo	Predeterminado
idproveedor *(Primaria)*	int(11)	No	
ruc	varchar(12)	Sí	*NULL*
proveedor	varchar(255)	Sí	*NULL*
direccion	varchar(255)	Sí	*NULL*
telefono	varchar(11)	Sí	*NULL*
email	varchar(255)	Sí	*NULL*
estado	enum('Activo', 'Inactivo')	Sí	Activo

Fuente: Elaboración propia

Tabla 35. Constraints de proveedor

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	idproveedor	3	A	No	
fk_proveedor_persona	BTREE	No	No	telefono	3	A	Sí	

Fuente: Elaboración propia

[8] Tabla: rol

Tabla 36. Campos rol

Columna	Tipo	Nulo
idrol *(Primaria)*	int(11)	No
nombre	varchar(50)	No
descripcion	varchar(255)	No

Fuente: Elaboración propia

Tabla 37. Constraints rol

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	idrol	2	A	No	

Fuente: Elaboración propia

[9] Tabla: usuario

Tabla 38. Campos Usuario

Columna	Tipo	Nulo	Predeterminado	Enlaces a
idusuario *(Primaria)*	int(11)	No		
idrol	int(11)	Sí	*NULL*	rol -> idrol
nombres	varchar(50)	Sí	*NULL*	
usuario	varchar(255)	Sí	*NULL*	
contrasena	varchar(250)	Sí	*NULL*	

Columna	Tipo	Nulo	Predeterminado	Enlaces a
email	varchar(250)	Sí	*NULL*	
telefono	varchar(10)	Sí	*NULL*	
estado	enum('Activo', 'Inactivo')	Sí	Activo	

Fuente: Elaboración propia

Tabla 39. Constraints cliente

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	idusuario	2	A	No	
usuario_nombre	BTREE	Sí	No	nombres	2	A	Sí	
usuario_email	BTREE	Sí	No	email	2	A	Sí	
FK__rol	BTREE	No	No	idrol	2	A	Sí	

Fuente: Elaboración propia

[10] Tabla: venta

Tabla 40. Campos venta

Columna	Tipo	Nulo	Predeterminado	Enlaces a
idventa *(Primaria)*	int(11)	No		
idcliente	int(11)	Sí	*NULL*	cliente -> idcliente
idusuario	int(11)	Sí	*NULL*	usuario -> idusuario
numeroserie	varchar(20)	Sí	0	
fecha	date	Sí	current_timestamp()	
impuesto	decimal(10,2)	Sí	12.00	
total	decimal(10,2)	Sí	*NULL*	
estado	enum('Pagada', 'Cancelada')	Sí	Pagada	

Fuente: Elaboración propia

Tabla 41. Constraints venta

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	idventa	8	A	No	
fk_cliente_venta	BTREE	No	No	idcliente	2	A	Sí	
idusuario	BTREE	No	No	idusuario	2	A	Sí	

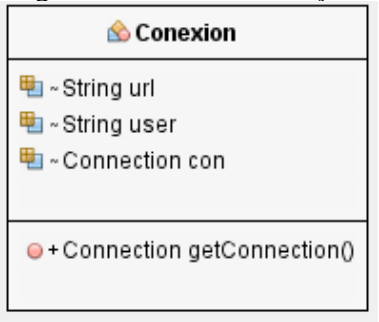
Fuente: Elaboración propia

Anexo 5: Clases del sistema

A continuación, se presentan las clases utilizadas en el desarrollo del sistema de control de inventarios para el almacén Alejandra.

- **Clase: conexión,** realiza la comunicación y envío de información entre la base de datos y el aplicativo. Esta clase tiene los atributos y métodos.

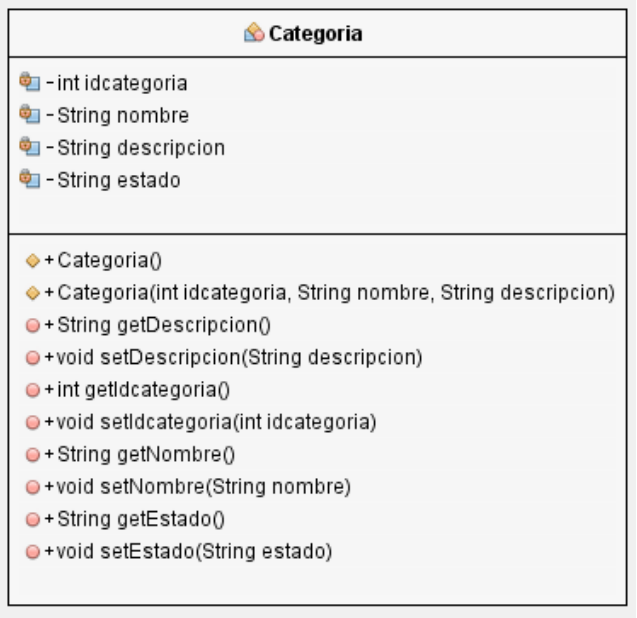
Figura 102. Clase conexion.java



Fuente: Elaboración propia

- **Clase: categoría,** a través de esta clase es posible la gestión de las diferentes categorías a las cuales pertenecen los diferentes productos. En la figura 102, se presentan los atributos y métodos pertenecientes a esta clase.

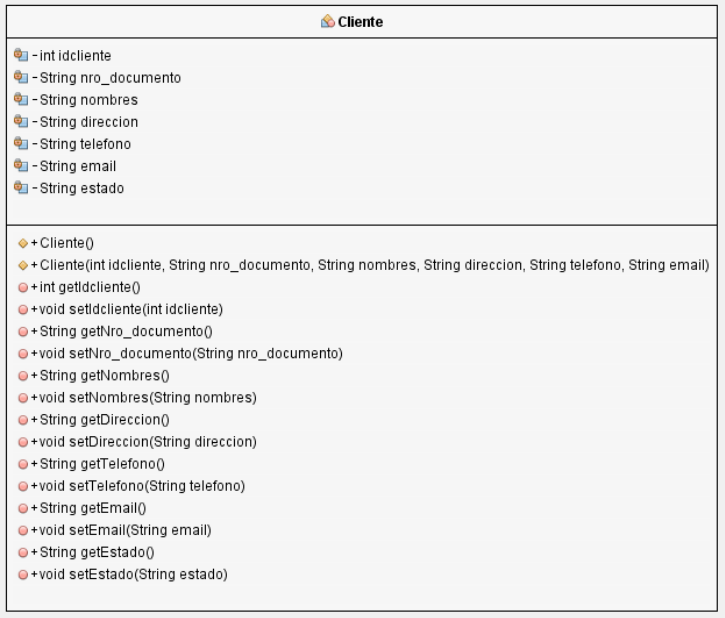
Figura 103. Contenido del archivo categoria.java



Fuente: Elaboración propia

- **Clase: cliente,** permite gestionar la información de los clientes. En la Figura 103, se presenta los atributos y métodos pertenecientes a dicha clase.

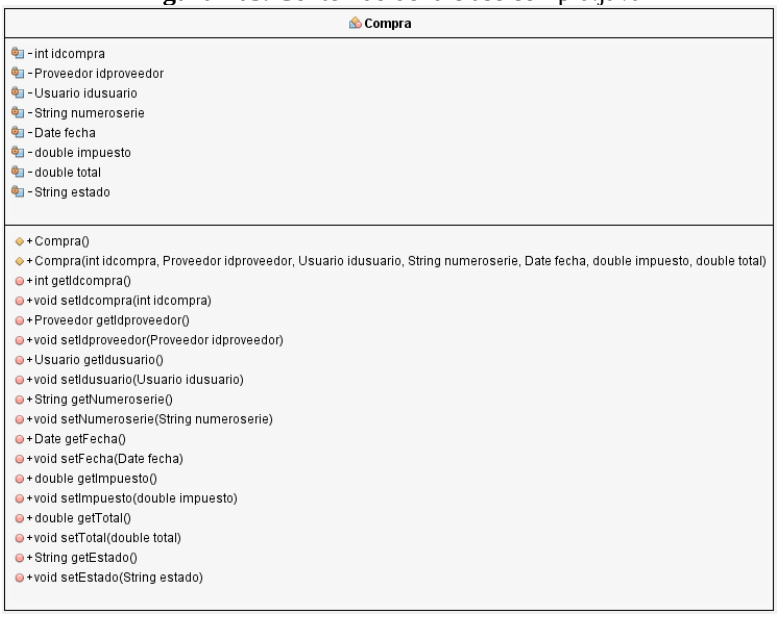
Figura 104. Contenido del archivo cliente.java



Fuente: Elaboración propia

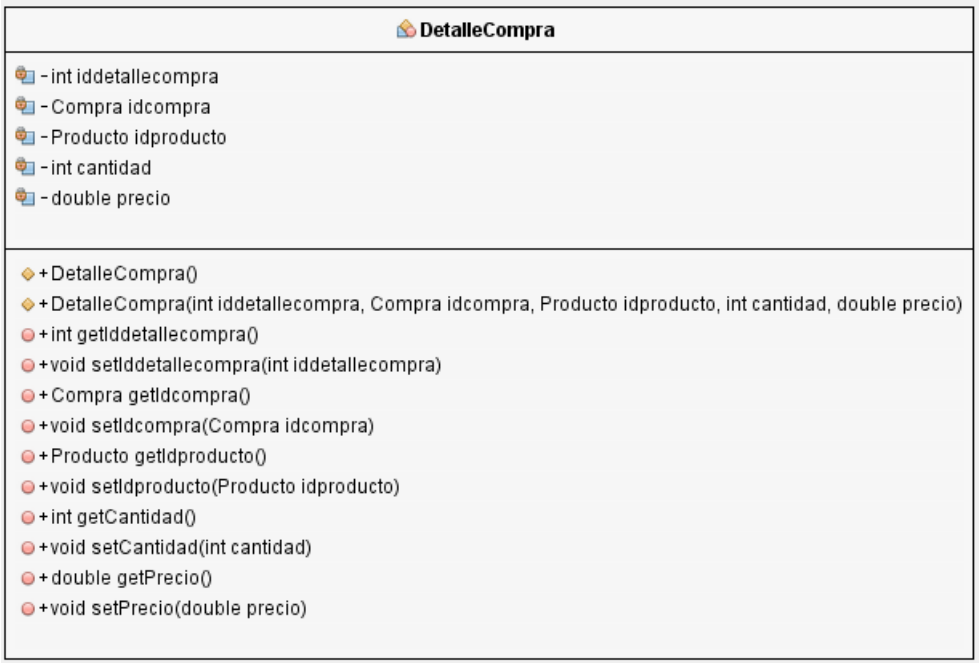
- **Clase: compra,** a través de esta clase, el sistema registra las compras realizadas por los clientes.

Figura 105. Contenido de la clase compra.java



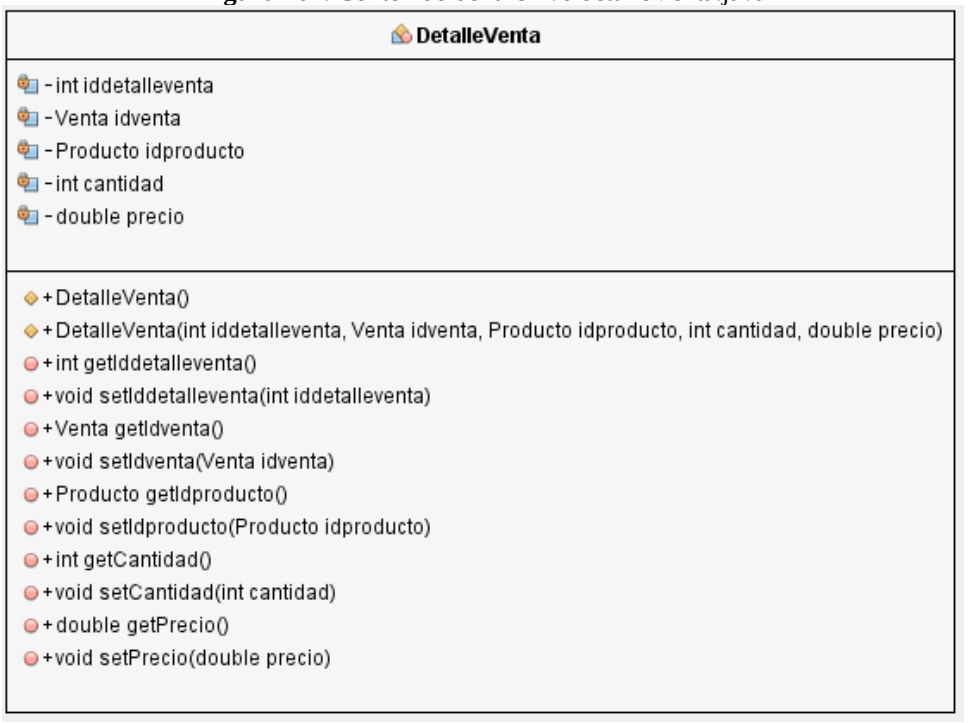
Fuente: Elaboración propia

Figura 106. Contenido del archivo detalleCompra.java



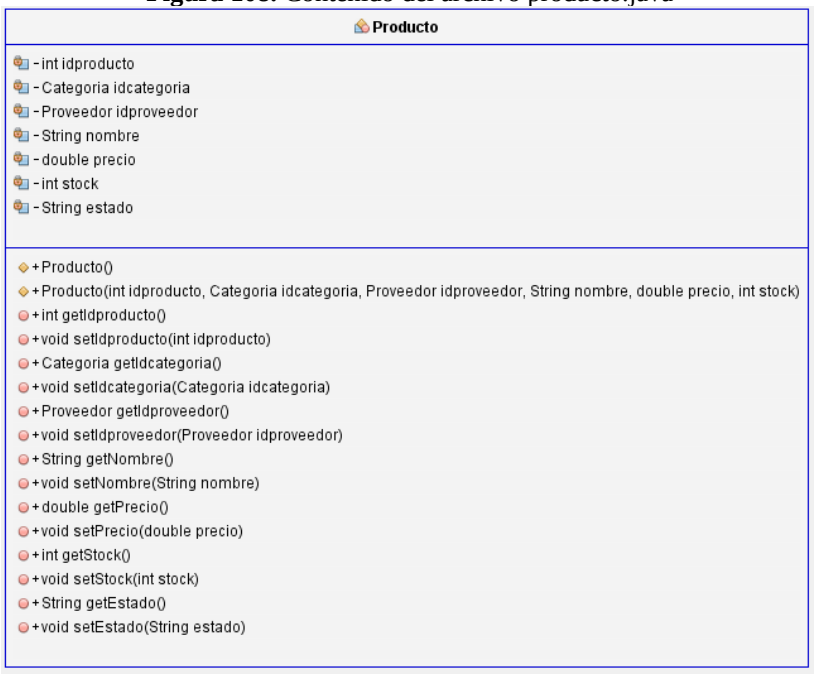
Fuente: Elaboración propia

Figura 107. Contenido del archivo detalleVenta.java



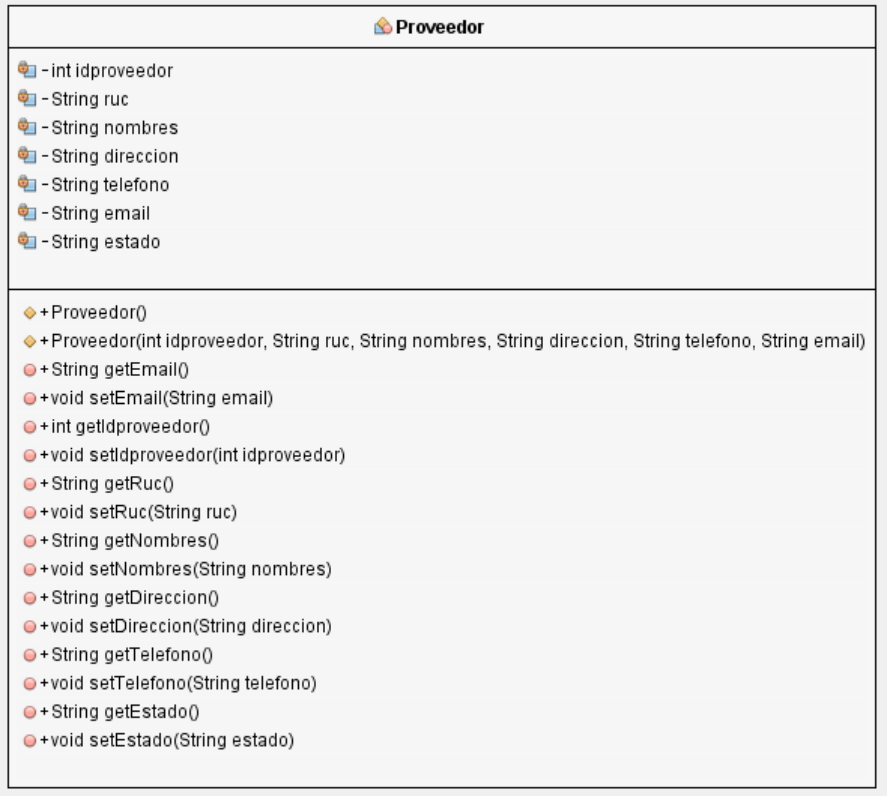
Fuente: Elaboración propia

Figura 108. Contenido del archivo producto.java



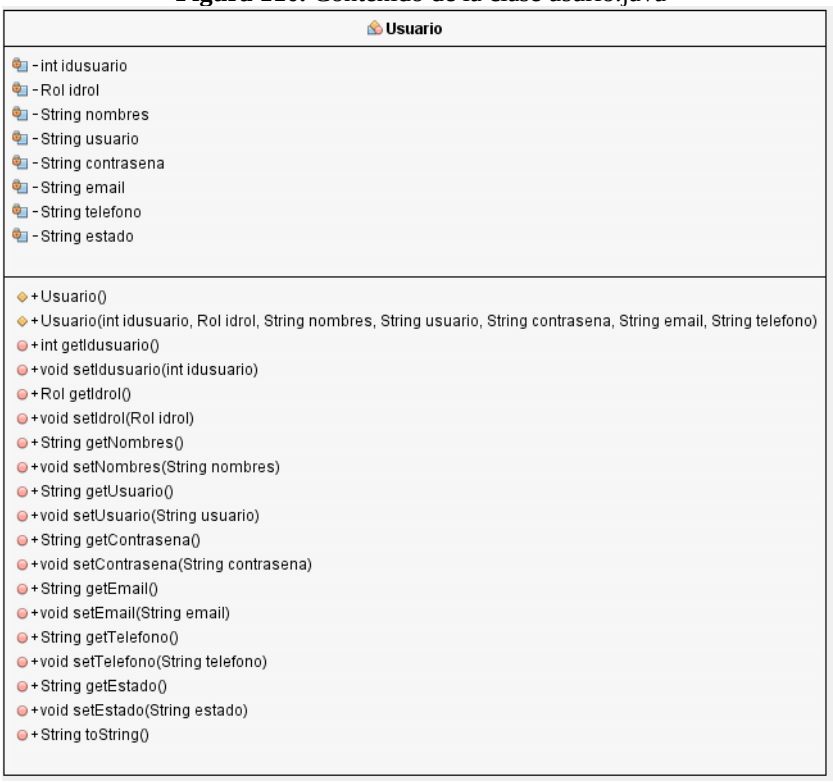
Fuente: Elaboración propia

Figura 109. Contenido de la clase proveedor.java



Fuente: Elaboración propia

Figura 110. Contenido de la clase usurio.java



Fuente: Elaboración propia

Figura 111. Contenido de la clase venta.java

Fuente: Elaboración propia

Anexo 6: Historias de usuario

Tabla 42. Historia de usuario 11

HISTORIA DE USUARIO 11	
ID: HU-11	Nombre Historia: Gestionar roles.
Usuario: Administrador	Sprint Asignado: 04
Fecha Inicio:	Fecha Fin:
Descripción: Como administrador del sistema necesito ingresar los diferentes roles del sistema.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación una vez ingresado el rol.	

Fuente: Elaboración propia

Tabla 43. Historia de usuario 12

HISTORIA DE USUARIO 12	
ID: HU-12	Nombre Historia: Asignar funcionalidad de acuerdo con el rol de usuario.
Usuario: Administrador	Sprint Asignado: 04
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito asignar funcionalidades a los usuarios de acuerdo con los diferentes roles.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 44. Historia de usuario 13

HISTORIA DE USUARIO 13	
ID: HU-13	Nombre Historia: Activar estado de vigencia de la funcionalidad asignada.
Usuario: Administrador	Sprint Asignado: 04
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito establecer un periodo de vigencia de las funcionalidades asignadas a los usuarios de acuerdo con su rol.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 45. Historia de usuario 14

HISTORIA DE USUARIO 14	
ID: HU-14	Nombre Historia: Buscar y visualizar usuarios empleados de la empresa.
Usuario: Administrador	Sprint Asignado: 04

Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito buscar y visualizar el listado de las personas que laboran en la empresa.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 46. Historia de usuario 15

HISTORIA DE USUARIO 15	
ID: HU-15	Nombre Historia: Gestionar proveedores.
Usuario: Administrador	Sprint Asignado: 05
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito gestionar (CRUD) los proveedores de la empresa.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 47. Historia de usuario 16

HISTORIA DE USUARIO 16	
ID: HU-16	Nombre Historia: Gestionar compras a proveedores.
Usuario: Administrador	Sprint Asignado: 05
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito un módulo en donde pueda realizar compras a los diferentes proveedores.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 48. Historia de usuario 17

HISTORIA DE USUARIO 17	
ID: HU-17	Nombre Historia: Consultar compras por fecha, mes
Usuario: Administrador	Sprint Asignado: 05
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito un módulo en donde pueda consultar las compras realizadas de acuerdo con un rango de fechas.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 49. Historia de usuario 18

HISTORIA DE USUARIO 18	
ID: HU-18	Nombre Historia: Buscar proveedores
Usuario: Administrador	Sprint Asignado: 05
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito una funcionalidad que permita buscar y visualizar a los diferentes proveedores de la empresa.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 50. Historia de usuario 19

HISTORIA DE USUARIO 19	
ID: HU-19	Nombre Historia: Historial de precios.
Usuario: Administrador	Sprint Asignado: 05
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito un reporte en donde se pueda ver a detalle el historial de precios de un determinado producto de acuerdo con su proveedor.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 51. Historia de usuario 20

HISTORIA DE USUARIO 20	
ID: HU-20	Nombre Historia: Reportes para el módulo proveedores.
Usuario: Administrador	Sprint Asignado: 05
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito generar reportes sobre los proveedores de la empresa.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 52. Historia de usuario 21

HISTORIA DE USUARIO 21	
ID: HU-21	Nombre Historia: Gestionar clientes.
Usuario: Administrador	Sprint Asignado: 06
Fecha Inicio:	Fecha Fin:

Descripción: como administrador del sistema necesito gestionar (CRUD) los clientes de la empresa.
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.

Fuente: Elaboración propia

Tabla 53. Historia de usuario 22

HISTORIA DE USUARIO 22	
ID: HU-22	Nombre Historia: Gestionar venta.
Usuario: Administrador	Sprint Asignado: 06
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito gestionar (CRUD) las ventas efectuadas.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 54. Historia de usuario 23

HISTORIA DE USUARIO 23	
ID: HU-23	Nombre Historia: Consultar ventas por día, fecha, mes.
Usuario: Administrador	Sprint Asignado: 06
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito consultar las ventas efectuadas de acuerdo con diferentes criterios de búsqueda por ejemplo día, fechas específicas o meses.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Tabla 55. Historia de usuario 24

HISTORIA DE USUARIO 24	
ID: HU-24	Nombre Historia: Reportes para el módulo ventas.
Usuario: Administrador	Sprint Asignado: 06
Fecha Inicio:	Fecha Fin:
Descripción: como administrador del sistema necesito un módulo de reportes para el módulo ventas. Estos reportes pueden ser mediante gráficas o tablas.	
Pruebas de Aceptación: ✓ Mostrar un mensaje de confirmación cuando la acción se realiza exitosamente.	

Fuente: Elaboración propia

Anexo 7: Pruebas unitarias

[1] Pruebas al Módulo Proveedor

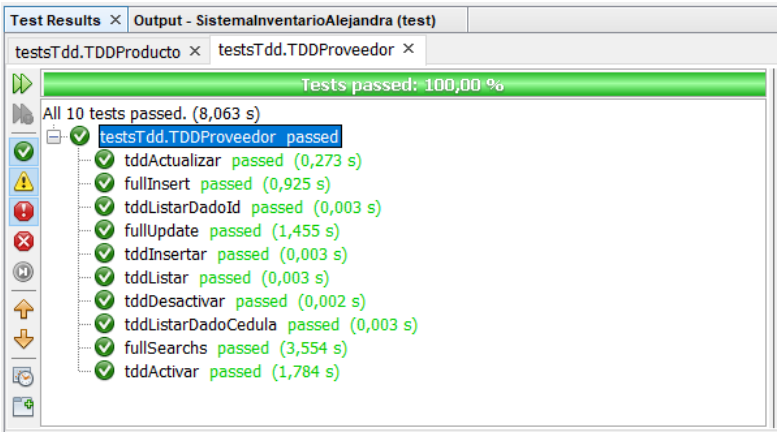
```
package testsTdd;

import java.util.ArrayList;
import java.util.List;
import modelo.dao.ProveedorDAO;
import modelo.entidades.Proveedor;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class TDDProveedor {

    Proveedor objProveedor;
    ProveedorDAO cdao;
    List<Proveedor> proveedores;
    int result;

    /*
    inicilaizar los objetos
     */
    @Before
    public void init() {
        cdao = new ProveedorDAO();
        objProveedor = new Proveedor();
        result = 0;
    }

    /*
    insertar un nuevo proveedor
     */
    @Test
    public void tddInsertar() throws Exception {
        try {
            objProveedor.setRuc("0123456789");
            objProveedor.setNombres("test nombre insertar");
            objProveedor.setDireccion("test direccion insertar");
            objProveedor.setTelefono("0987654321");
            objProveedor.setEmail("testinsertar@test.com");
            result = cdao.agregar(objProveedor);
        } catch (Exception e) {
            System.out.println("public void tddInsertar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD insertar dice: ", 1, result);
    }

    /*
    actualizar un proveedor
     */
    @Test
    public void tddActualizar() throws Exception {
        try {
            objProveedor.setRuc("9876543210");
            objProveedor.setNombres("test nombre actualizar");
            objProveedor.setDireccion("test direccion actualizar");
            objProveedor.setTelefono("0912345678");
            objProveedor.setEmail("testactualizar@test.com");
            objProveedor.setIdproveedor(6);
            result = cdao.actualizar(objProveedor);
        } catch (Exception e) {
            System.out.println("public void tddActualizar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD actualizar dice: ", 1, result);
    }

    /*
    Listar todos los proveedors
```

```
     */
    @Test
    public void tddListar() {
        List<Proveedor> lstproveedors = new ArrayList<>();
        try {
            lstproveedors = cdao.listar();
        } catch (Exception e) {
            System.out.println("Tdd listar dice: " + e.getMessage());
        }
        Assert.assertNotNull("Tdd listar proveedores", lstproveedors);
    }

    /*
    buscar proveedor por id
     */
    @Test
    public void tddListarDadoId() {
        objProveedor = cdao.buscarProveedor(6);
        Assert.assertNotNull("Tdd listar proveedores dado Id: ", objProveedor.getIdproveedor());
    }

    /*
    buscar proveedor por cedula
     */
    @Test
    public void tddListarDadoCedula() {
        objProveedor = cdao.listarID("0987654321");
        Assert.assertNotNull("Tdd listar proveedores dado Cedula: ",
objProveedor.getIdproveedor());
    }

    /*
    desactivar un proveedor
     */
    @Test
    public void tddDesactivar() throws Exception {
        try {
            int idProveedor = 6;
            result = cdao.desactivar(idProveedor);
        } catch (Exception e) {
            System.out.println("public void tddDesactivar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD desactivar dice: ", 1, result);
    }

    /*
    activar un proveedor
     */
    @Test
    public void tddActivar() throws Exception {
        try {
            int idProveedor = 6;
            result = cdao.activar(idProveedor);
        } catch (Exception e) {
            System.out.println("public void tddActivar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD activar dice: ", 1, result);
    }

    /*
    creación de un método que inserta 200 usuarios de prueba
     */
    @Test
    public void fullInsert() {
        boolean cont = false;
        String numberId = "0603977";
        String names = "test ";
        String address = "test ";
        String phone = "0996824";
        try {
            for (int i = 0; i < 200; i++) {
                objProveedor = new Proveedor();
                objProveedor.setRuc(numberId + transformNumber(i));
```

```
            objProveedor.setNombres(names + transformNumber(i));
            objProveedor.setDireccion(address + transformNumber(i));
            objProveedor.setTelefono(phone + transformNumber(i));
            objProveedor.setEmail("test" + transformNumber(i) + "@test.com");
            result = cdao.agregar(objProveedor);
            cont = true;
        }
    } catch (Exception e) {
        System.out.println("ERROR  public void fullInsert() dice: " + e.getMessage());
    }
    Assert.assertTrue(cont);
}

/*
actualización en masa
 */
@Test
public void fullUpdate() {
    boolean resultu = false;
    int upd = 0;
    String names = "update test ";
    String address = "update test ";

    try {
        proveedores = cdao.listar();
        for (int i = 0; i < proveedores.size(); i++) {
            objProveedor = new Proveedor();
            objProveedor = cdao.listarID(proveedores.get(i).getRuc());
            objProveedor.setNombres(names + transformNumber(i + 1));
            objProveedor.setDireccion(address + transformNumber(i + 1));
            objProveedor.setEmail("update_test" + transformNumber(i + 1) + "@test.com");
            upd = cdao.actualizar(objProveedor);
            if (upd == 1) {
                resultu = true;
            } else {
                resultu = false;
            }
        }
    } catch (Exception e) {
        System.out.println("ERROR public void fullUpdate() dice: " + e.getMessage());
    }
    Assert.assertTrue(resultu);
}

/*
consulta masiva de datos
 */
@Test
public void fullSearchs() {
    boolean p = false;
    try {
        for (int i = 0; i < 999; i++) {
            proveedores = cdao.listar();
            System.out.println("proveedores almacenados: " + proveedores.size());
            if (proveedores.size() > 0) {
                p = false;
            } else {
                p = true;
            }
        }
    } catch (Exception e) {
        System.out.println("ERROR public void fullSearchs() dice: " + e.getMessage());
    }
    Assert.assertFalse(p);
}

public String transformNumber(int num) {
    String n = null;
    if (num <= 9) {
        n = "00" + num;
    }
    if (num > 9 && num < 100) {
        n = "0" + num;
```

```
        }
        if (num >= 100) {
            n = String.valueOf(num);
        }

        return n;
    }
}
```

Resultados de la ejecución de las pruebas



[2] Pruebas al módulo categoría

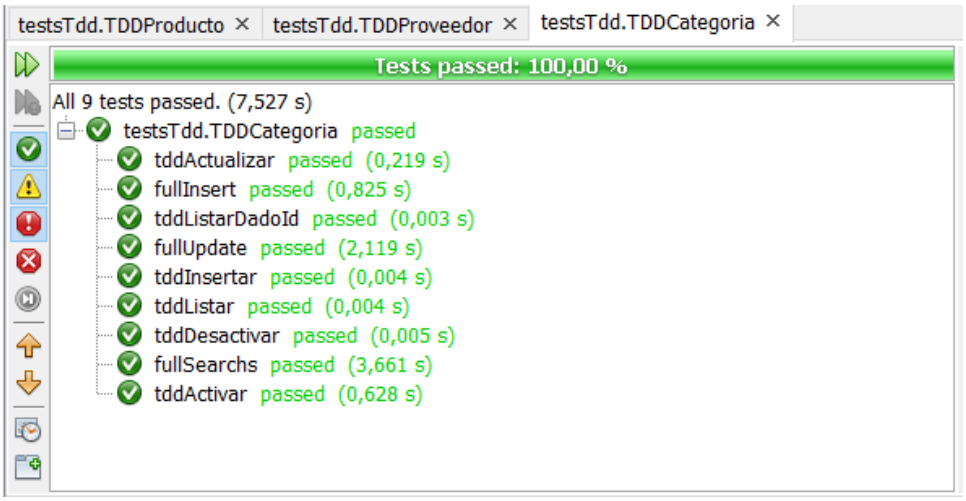
```
package testsTdd;

import java.util.ArrayList;
import java.util.List;
import modelo.dao.CategoriaDAO;
import modelo.entidades.Categoria;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class TDDCategoria {

    Categoria objCategoria;
    CategoriaDAO cdao;
    List<Categoria> categorias;

    /*
    inicilaizar los objetos
     */
    @Before
    public void init() {
        cdao = new CategoriaDAO();
        objCategoria = new Categoria();
    }

    /*
    insertar un nuevo categoria
     */
    @Test
    public void tddInsertar() throws Exception {
        int result = 0;
        try {
            objCategoria.setNombre("test nombre insertar");
            objCategoria.setDescripcion("test descripcion insertar");
            result = cdao.agregar(objCategoria);
        } catch (Exception e) {
            System.out.println("public void tddInsertar() dice: " + e.getMessage());
        }
```

```
    Assert.assertEquals("TDD insertar dice: ", 1, result);
}

/*
actualizar un categoria
 */
@Test
public void tddActualizar() throws Exception {
    int result = 0;
    try {
        objCategoria.setNombre("test nombre actualizar");
        objCategoria.setDescripcion("test descripcion actualizar");
        objCategoria.setIdcategoria(5);
        result = cdao.actualizar(objCategoria);
    } catch (Exception e) {
        System.out.println("public void tddActualizar() dice: " + e.getMessage());
    }
    Assert.assertEquals("TDD actualizar dice: ", 1, result);
}

/*
Listar todos los categorias
 */
@Test
public void tddListar() {
    List<Categoria> lstcategorias = new ArrayList<>();
    try {
        lstcategorias = cdao.listar();
    } catch (Exception e) {
        System.out.println("Tdd listar dice: " + e.getMessage());
    }
    Assert.assertNotNull("Tdd listar categorias", lstcategorias);
}

/*
buscar categoria por id
 */
@Test
public void tddListarDadoId() {
    objCategoria = cdao.buscarCategoria(5);
    Assert.assertNotNull("Tdd listar categorias dado Id: ", objCategoria.getIdcategoria());
}

/*
desactivar un categoria
 */
@Test
public void tddDesactivar() throws Exception {
    int result = 0;
    try {
        int idCategoria = 5;
        result = cdao.desactivar(idCategoria);
    } catch (Exception e) {
        System.out.println("public void tddDesactivar() dice: " + e.getMessage());
    }
    Assert.assertEquals("TDD desactivar dice: ", 1, result);
}

/*
activar un categoria
 */
@Test
public void tddActivar() throws Exception {
    int result = 0;
    try {
        int idCategoria = 5;
        result = cdao.activar(idCategoria);
    } catch (Exception e) {
        System.out.println("public void tddActivar() dice: " + e.getMessage());
    }
    Assert.assertEquals("TDD activar dice: ", 1, result);
}
```

```
/*
creación de un método que inserta 200 usuarios de prueba
 */
@Test
public void fullInsert() {
    int result = 0;
    boolean cont = false;
    String names = "test ";
    String descripcion = "test ";
    try {
        for (int i = 0; i < 200; i++) {
            objCategoria = new Categoria();
            objCategoria.setNombre(names + transformNumber(i));
            objCategoria.setDescripcion(descripcion + transformNumber(i));
            result = cdao.agregar(objCategoria);
            cont = true;
        }
    } catch (Exception e) {
        System.out.println("ERROR  public void fullInsert() dice: " + e.getMessage());
    }
    Assert.assertTrue(cont);
}

/*
actualización en masa
 */
@Test
public void fullUpdate() {
    boolean resultu = false;
    int upd = 0;
    String names = "update test ";
    String descripcion = "update test ";

    try {
        categorias = cdao.listar();
        for (int i = 0; i < categorias.size(); i++) {
            objCategoria = new Categoria();
            objCategoria.setNombre(names + transformNumber(i + 1));
            objCategoria.setDescripcion(descripcion + transformNumber(i + 1));
            objCategoria.setIdcategoria(Integer.parseInt(transformNumber(i + 1)));
            upd = cdao.actualizar(objCategoria);
            if (upd == 1) {
                resultu = true;
            } else {
                resultu = false;
            }
        }
    } catch (Exception e) {
        System.out.println("ERROR public void fullUpdate() dice: " + e.getMessage());
    }
    Assert.assertTrue(resultu);
}

/*
consulta masiva de datos
 */
@Test
public void fullSearchs() {
    boolean p = false;
    try {
        for (int i = 0; i < 999; i++) {
            categorias = cdao.listar();
            System.out.println("categorias almacenados: " + categorias.size());
            if (categorias.size() > 0) {
                p = false;
            } else {
                p = true;
            }
        }
    } catch (Exception e) {
        System.out.println("ERROR public void fullSearchs() dice: " + e.getMessage());
    }
    Assert.assertFalse(p);
```

```
    }

    public String transformNumber(int num) {
        String n = null;
        if (num <= 9) {
            n = "00" + num;
        }
        if (num > 9 && num < 100) {
            n = "0" + num;
        }
        if (num >= 100) {
            n = String.valueOf(num);
        }

        return n;
    }
}
```

Resultados de la ejecución de las pruebas



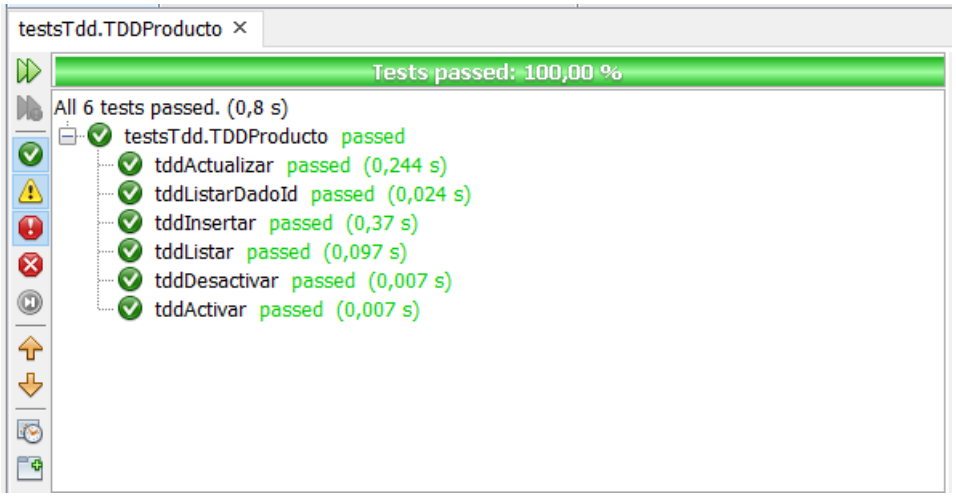
[3] Pruebas al módulo producto

```
package testsTdd;

import java.util.ArrayList;
import java.util.List;
import modelo.dao.*;
import modelo.entidades.*;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class TDDProducto {

    ProductoDAO pdao;
    Producto objProducto;
    CategoriaDAO cdao;
    Categoria objCategoria;
    ProveedorDAO provdao;
    Proveedor objProveedor;

    /*
    inicilaizar los objetos
     */
    @Before
    public void init() {
        pdao = new ProductoDAO();
        objProducto = new Producto();
        cdao = new CategoriaDAO();
        objCategoria = new Categoria();
```

```
        provdao = new ProveedorDAO();
        objProveedor = new Proveedor();
    }

    /*
    insertar un nuevo producto
     */
    @Test
    public void tddInsertar() throws Exception {
        int result = 0;
        try {
            objProducto.setNombre("test nombre insertar");
            objCategoria =  cdao.buscarCategoria(5);
            objProducto.setIdcategoria(objCategoria);
            objProveedor =  provdao.buscarProveedor(7);
            objProducto.setIdproveedor(objProveedor);
            objProducto.setPrecio(10);
            objProducto.setStock(10);
            result = pdao.agregar(objProducto);
        } catch (Exception e) {
            System.out.println("public void tddInsertar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD insertar dice: ", 1, result);
    }

    /*
    actualizar un producto
     */
    @Test
    public void tddActualizar() throws Exception {
        int result = 0;
        try {
            objProducto.setNombre("test nombre actualizar");
            objCategoria =  cdao.buscarCategoria(5);
            objProducto.setIdcategoria(objCategoria);
            objProveedor =  provdao.buscarProveedor(7);
            objProducto.setIdproveedor(objProveedor);
            objProducto.setPrecio(10);
            objProducto.setStock(10);
            objProducto.setIdproducto(8);
            result = pdao.actualizar(objProducto);
        } catch (Exception e) {
            System.out.println("public void tddInsertar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD actualizar dice: ", 1, result);
    }


    /*
    Listar todos los productos
    */
    @Test
    public void tddListar() {
        List<Producto> lstproductos = new ArrayList<>();
        try {
            lstproductos = pdao.listar();
        } catch (Exception e) {
            System.out.println("Tdd listar dice: " + e.getMessage());
        }
        Assert.assertNotNull("Tdd listar productos", lstproductos);
    }


    /*
    buscar producto por id
    */
    @Test
    public void tddListarDadoId() {
        objProducto = pdao.buscarProducto(8);
        Assert.assertNotNull("Tdd listar productos dado Id: ", objProducto.getIdproducto());
    }


    /*
```

```
    desactivar un producto
     */
    @Test
    public void tddDesactivar() throws Exception {
        int result = 0;
        try {
            int idProducto = 8;
            result = cdao.desactivar(idProducto);
        } catch (Exception e) {
            System.out.println("public void tddDesactivar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD desactivar dice: ", 1, result);
    }

    /*
    activar un producto
     */
    @Test
    public void tddActivar() throws Exception {
        int result = 0;
        try {
            int idProducto = 8;
            result = cdao.activar(idProducto);
        } catch (Exception e) {
            System.out.println("public void tddActivar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD activar dice: ", 1, result);
    }
}
```

Resultados de la ejecución de las pruebas



[4] Pruebas al módulo Cliente

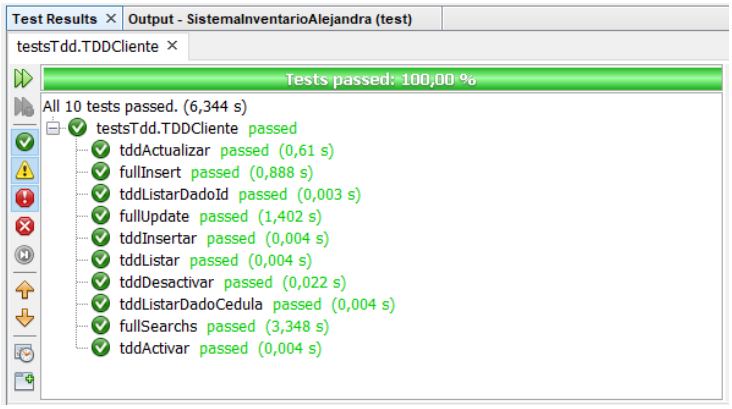
```
package testsTdd;

import java.util.ArrayList;
import java.util.List;
import modelo.dao.ClienteDAO;
import modelo.entidades.Cliente;
import org.junit.*;

public class TDDCliente {

    Cliente objCliente;
    ClienteDAO cdao;
    List<Cliente> clientes;

    /*
```

```
inicilaizar los objetos
 */
@Before
public void init() {
    cdao = new ClienteDAO();
    objCliente = new Cliente();
}

/*
insertar un nuevo cliente
 */
@Test
public void tddInsertar() throws Exception {
    int result = 0;
    try {
        objCliente.setNro_documento("0123456789");
        objCliente.setNombres("test nombre insertar");
        objCliente.setDireccion("test direccion insertar");
        objCliente.setTelefono("0987654321");
        objCliente.setEmail("testinsertar@test.com");
        result = cdao.agregar(objCliente);
    } catch (Exception e) {
        System.out.println("public void tddInsertar() dice: " + e.getMessage());
    }
    Assert.assertEquals("TDD insertar dice: ", 1, result);
}

/*
actualizar un cliente
 */
@Test
public void tddActualizar() throws Exception {
    int result = 0;
    try {
        objCliente.setNro_documento("9876543210");
        objCliente.setNombres("test nombre actualizar");
        objCliente.setDireccion("test direccion actualizar");
        objCliente.setTelefono("0912345678");
        objCliente.setEmail("testactualizar@test.com");
        objCliente.setIdcliente(3);
        result = cdao.actualizar(objCliente);
    } catch (Exception e) {
        System.out.println("public void tddActualizar() dice: " + e.getMessage());
    }
    Assert.assertEquals("TDD actualizar dice: ", 1, result);
}

/*
Listar todos los clientes
 */
@Test
public void tddListar() {
    List<Cliente> lstclientes = new ArrayList<>();
    try {
        lstclientes = cdao.listar();
    } catch (Exception e) {
        System.out.println("Tdd listar dice: " + e.getMessage());
    }
    Assert.assertNotNull("Tdd listar clientes", lstclientes);
}

/*
buscar cliente por id
 */
@Test
public void tddListarDadoId() {
    objCliente = cdao.buscarCliente(3);
    Assert.assertNotNull("Tdd listar clientes dado Id: ", objCliente.getIdcliente());
}

/*
buscar cliente por cedula
 */
```

```
@Test
public void tddListarDadoCedula() {
    objCliente = cdao.listarID("0605316843");
    Assert.assertNotNull("Tdd listar clientes dado Cedula: ", objCliente.getIdcliente());
}

/*
desactivar un cliente
 */
@Test
public void tddDesactivar() throws Exception {
    int result = 0;
    try {
        int idCliente = 3;
        result = cdao.desactivar(idCliente);
    } catch (Exception e) {
        System.out.println("public void tddDesactivar() dice: " + e.getMessage());
    }
    Assert.assertEquals("TDD desactivar dice: ", 1, result);
}

/*
activar un cliente
 */
@Test
public void tddActivar() throws Exception {
    int result = 0;
    try {
        int idCliente = 3;
        result = cdao.activar(idCliente);
    } catch (Exception e) {
        System.out.println("public void tddActivar() dice: " + e.getMessage());
    }
    Assert.assertEquals("TDD activar dice: ", 1, result);
}

/*
creación de un método que inserta 200 usuarios de prueba
 */
@Test
public void fullInsert() {
    boolean cont = false;
    int band = 0;
    String numberId = "0603977";
    String names = "test ";
    String address = "test ";
    String phone = "0996824";
    try {
        for (int i = 0; i < 200; i++) {
            objCliente = new Cliente();
            objCliente.setNro_documento(numberId + transformNumber(i));
            objCliente.setNombres(names + transformNumber(i));
            objCliente.setDireccion(address + transformNumber(i));
            objCliente.setTelefono(phone + transformNumber(i));
            objCliente.setEmail("test" + transformNumber(i) + "@test.com");
            band = cdao.agregar(objCliente);
            cont = true;
        }
    } catch (Exception e) {
        System.out.println("ERROR  public void fullInsert() dice: " + e.getMessage());
    }
    Assert.assertTrue(cont);
}

/*
actualización en masa
 */
@Test
public void fullUpdate() {
    boolean band = false;
    int upd = 0;
    String names = "update test ";
    String address = "update test ";
```

```
        try {
            clientes = cdao.listar();
            for (int i = 0; i < clientes.size(); i++) {
                objCliente = new Cliente();
                objCliente = cdao.listarID(clientes.get(i).getNro_documento());
                objCliente.setNombres(names + transformNumber(i + 1));
                objCliente.setDireccion(address + transformNumber(i + 1));
                objCliente.setEmail("update_test" + transformNumber(i + 1) + "@test.com");
                upd = cdao.actualizar(objCliente);
                if (upd == 1) {
                    band = true;
                } else {
                    band = false;
                }
            }
        } catch (Exception e) {
            System.out.println("ERROR public void fullUpdate() dice: " + e.getMessage());
        }
        Assert.assertTrue(band);
    }

    /*
    consulta masiva de datos
     */
    @Test
    public void fullSearchs() {
        boolean p = false;
        try {
            for (int i = 0; i < 999; i++) {
                clientes = cdao.listar();
                System.out.println("clientes almacenados: " + clientes.size());
                if (clientes.size() > 0) {
                    p = false;
                } else {
                    p = true;
                }
            }
        } catch (Exception e) {
            System.out.println("ERROR public void fullSearchs() dice: " + e.getMessage());
        }
        Assert.assertFalse(p);
    }

    public String transformNumber(int num) {
        String n = null;
        if (num <= 9) {
            n = "00" + num;
        }
        if (num > 9 && num < 100) {
            n = "0" + num;
        }
        if (num >= 100) {
            n = String.valueOf(num);
        }

        return n;
    }
}
```

Resultados de la ejecución de las pruebas



[5] Pruebas a la entidad Ventas

```
package testsTdd;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import modelo.dao.*;
import modelo.entidades.*;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class TDDVenta {

    Venta objVenta;
    VentaDAO cdao;
    Cliente objCliente;
    ClienteDAO pdao;
    Usuario objUsuario;
    UsuarioDAO udao;
    DetalleVenta objDVenta;
    Producto objProducto;
    ProductoDAO prodao;

    /*
    inicilaizar los objetos
     */
    @Before
    public void init() {
        cdao = new VentaDAO();
        objVenta = new Venta();
        pdao = new ClienteDAO();
        objCliente = new Cliente();
        udao = new UsuarioDAO();
        objUsuario = new Usuario();
        objDVenta = new DetalleVenta();
        prodao = new ProductoDAO();
        objProducto = new Producto();
    }

    /*
    insertar un nuevo venta
     */
    @Test
    public void tddInsertarVenta() throws Exception {
        int result = 0;
        try {
            objCliente = pdao.buscarCliente(3);
            objVenta.setIdcliente(objCliente);
            objUsuario = udao.buscarUsuario(7);
            objVenta.setIdusuario(objUsuario);
```

```
            objVenta.setNumeroserie("00000018");
            objVenta.setFecha(new Date());
            objVenta.setTotal(10);
            result = cdao.guardarVentas(objVenta);
        } catch (Exception e) {
            System.out.println("public void tddInsertar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD insertar venta dice: ", 1, result);
    }

    /*
    insertar un nuevo detalle venta
     */
    @Test
    public void tddInsertarDetalleVenta() throws Exception {
        int result = 0;
        try {
            objVenta = cdao.idVentas();
            objDVenta.setIdventa(objVenta);
            objProducto = prodao.buscarProducto(8);
            objDVenta.setIdproducto(objProducto);
            objDVenta.setCantidad(1);
            objDVenta.setPrecio(10);
            result = cdao.guardarDetalleVentas(objDVenta);
        } catch (Exception e) {
            System.out.println("public void tddInsertar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD insertar detalle venta dice: ", 1, result);
    }

    /*
    Listar todos las ventas
     */
    @Test
    public void tddListar() {
        List<Venta> lstventas = new ArrayList<>();
        try {
            lstventas = cdao.listar();
        } catch (Exception e) {
            System.out.println("Tdd listar dice: " + e.getMessage());
        }
        Assert.assertNotNull("Tdd listar ventas", lstventas);
    }

    /*
    Listar ventas por rango de fecha
     */
    @Test
    public void tddListarFecha() {
        List<Venta> lstventas = new ArrayList<>();
        try {
            lstventas = cdao.listarVentaFecha(java.sql.Date.valueOf("2021-09-16"),
java.sql.Date.valueOf("2021-09-16"));
            System.out.println(lstventas);
        } catch (Exception e) {
            System.out.println("Tdd listar dice: " + e.getMessage());
        }
        Assert.assertNotNull("Tdd listar ventas fecha", lstventas);
    }

    /*
    Listar id maximo de la venta
     */
    @Test
    public void tddListarIdMaxVenta() {
        objVenta = cdao.idVentas();
        Assert.assertNotNull("Tdd listar id maxixo venta: ", objVenta.getIdventa());
    }

    /*
    Listar id maximo de la venta
     */
    @Test
```

```
    public void tddListarNumMaxFactura() {
        String numFactura = cdao.numeroFactura();
        Assert.assertNotNull("Tdd listar numero factura venta: ", numFactura);
    }

    /*
    cancelar un venta
     */
    @Test
    public void tddCancelar() throws Exception {
        int result = 0;
        try {
            int idVenta = 69;
            result = cdao.cancelarVenta(idVenta);
        } catch (Exception e) {
            System.out.println("public void tddDesactivar() dice: " + e.getMessage());
        }
        Assert.assertEquals("TDD cancelar venta dice: ", 1, result);
    }
}
```

Resultados de la ejecución de las pruebas