UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA

FACULTAD DE INGENIERÍA DE MINAS, GEOLOGÍA Y CIVIL

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

"MODELO DE FRAMEWORK DE CÓDIGO ABIERTO JAVA PARA EL DESARROLLO RÁPIDO DE APLICACIONES WEB EMPRESARIALES BASADAS EN COMPONENTES DE NEGOCIO PARA ENTORNOS AGILES, 2019"

Tesis presentada por	:	Bach. Eliaquim Oriundo Mallcco
Para optar el título profesional de	:	Ingeniero de Sistemas
Tipo de Investigación	:	Descriptiva
Área de investigación	:	Ingeniería de software
Asesor	:	Mag. Ing. Hubner Janampa Patilla

Ayacucho, Junio del 2019

DEDICATORIA

Al Dios todo Poderoso por haberme dado la fortaleza y salud para cumplir uno de mis objetivos, y que cada día me permite mi crecimiento personal, profesional y espiritual.

A mis padres quienes siempre me apoyaron y quienes se sacrificaron para darme una educación, a mis maestros, quienes se empeñaron en lograr que entraran sus enseñanzas en mí, a mis familiares y amigos, quienes siempre me alentaron a la realización de este proyecto, además por todo el apoyo incondicional que siempre me han demostrado, y a todas las personas que colaboraron con sus valiosas opiniones, en el desarrollo de este proyecto.

AGRADECIMIENTO

A la Universidad Nacional De San Cristóbal De Huamanga, mi alma mater y a los docentes de la Escuela Profesional de Ingeniería de Sistemas, que con su ardua labor me ayudaron a superarme en esta etapa de mi vida universitaria.

Al Rev. Teódulo Ramírez Álamo, por su apoyo espiritual y moral durante toda mi vida además de las experiencias vividas y compartidas.

Al Mg. Ing. Hubner Janampa Patilla por su apoyo en la elaboración de este proyecto y por su asesoría pues me ha sido de mucha ayuda.

Sé que estas palabras no son suficientes para expresar mi agradecimiento, pero espero que, con ellas, se den a entender mis sentimientos de aprecio y cariño a todos ellos

CONTENIDO

Pág.

CAPITULO III

METODOLOGÍA DE LA INVESTIGACIÓN

CAPITULO IV

ANÁLISIS Y RESULTADOS DE LA INVESTIGACIÓN

RESUMEN

Existen en el mercado muchas herramientas de Framework de código abierto java conocidos como open source, que ayudan a la comunidad de programadores web a mejorar los procesos de desarrollo de las aplicaciones web.

En muchos proyectos de desarrollo de software, los desarrolladores se afrontan un dilema cotidiano, esto tiene relación con respecto al grado de satisfacción del producto final software por parte de los usuarios finales, y el tiempo que conlleva la culminación del proyecto.

También, las empresas y organizaciones de hoy en día buscan soluciones tecnológicas a medida y precisión, en el menor tiempo posible, buscando la misma calidad y el grado de satisfacción de las aplicaciones basadas en sus requisitos, con la finalidad de que sus sistemas sean mantenibles.

El modelo de Framework de código abierto java nos permitirá optimizar el tiempo de desarrollo de las aplicaciones web manteniendo los estándares de calidad. Esto enfocado ala aplicaciones web empresariales basadas en componentes de negocios para entornos agiles.

El objeto de esta investigación es estudiar la mejora de un producto software utilizando el modelo de Framework de código abierto java marcadas por ICONIX como metodología de desarrollo para las aplicaciones web empresariales. Además de ello se incluye estimaciones de la calidad del producto, la agilidad en el desarrollo. Además, dado que el factor humano es fundamental, se presenta un análisis cualitativo del desarrollo del proyecto.

PALABRAS CLAVE

Framework de código abierto java, aplicaciones web empresariales, componentes de negocio, entonos agiles, Plain Old Java Object (POJOs), Java Persistence API (JPA), Anotaciones, interfaz amigable, procesamiento de datos, Iconix.

INTRODUCCIÓN

Actualmente los Framework enfocados al desarrollo de aplicaciones web empresariales no estiman los tiempos y el grado de satisfacción de los usuarios, dentro de un ciclo de vida de un sistema de información empresarial.

En la búsqueda de las mejoras continuas y aprovechamientos de las tecnologías dominantes en temas de soluciones tecnológicas empresariales de software, se plantea esta investigación como modelo para las aplicaciones web basadas en componentes de negocio. Esto permitirá a los desarrolladores tener un alto grado de tasa de éxito en los proyectos de software, así como la aceptación y satisfacción del producto final por parte de los usuarios del sistema.

La motivación personal es investigar y dar a conocer una herramienta - modelo Framework dentro del universo java para lograr tasas altas de éxito en la producción y calidad de los servicios de software empresariales orientadas las empresas del ámbito local, nacional e internacional.

La investigación se centrará en el estudio del modelo de Framework de código abierto java para el desarrollo rápido de aplicaciones web empresariales basadas en componentes de negocio para entornos agiles, donde los objetivos específicos a lograr son a) Utilizar Plain Old Java Object (POJOs) como mecanismo para lograr la Funcionalidad del componente de negocio. b) Utilizar anotaciones. Como mecanismo para lograr interfaz amigable. C) Utilizar las Java Persistence API(JPA) como mecanismo de procesamiento de datos, a través de un caso de estudio: sistema de ventas de una librería.

CAPITULO I

PLANTEAMIENTO DE LA INVESTIGACIÓN

1.1. DIAGNÓSTICO Y ENUNCIADO DEL PROBLEMA

El manejo de la información a través del procesamiento de datos, es la tendencia actual en las organizaciones. El diccionario de la lengua española (RAE, RAE, 2011) define la palabra dato, dentro de la informática como: "Información dispuesta de manera adecuada para su tratamiento por una computadora", también como: "Información sobre algo concreto que permite su conocimiento exacto o sirve para deducir las consecuencias derivadas de un hecho". Los datos al ser analizadas y cuestionados se les proveen de un significado que produce información y conocimiento, a esto se le conoce como el procesamiento de datos, lo cual la (RAE, 2014) lo define como "La aplicación sistemática de una serie de operaciones sobre un conjunto de datos, generalmente por medio de máquinas, para explotar la información que estos datos representan".

Actualmente las soluciones tecnológicas basadas en software para las empresas, en su gran mayoría están desarrolladas por la comunidad de código abierto (open source), lo cual (OSI, 2007) define al código abierto Código abierto (en idioma inglés open source) como: "Término con el que se conoce al software distribuido y desarrollado libremente". El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código que a las cuestiones morales y/o filosóficas las cuales destacan en el llamado Software libre, para ello se debe de cumplir los criterios como: Redistribución gratuita, Código fuente, Trabajos derivados, Integridad del código fuente del autor, No discriminación contra personas o grupos, No hay discriminación contra los campos de esfuerzo, Distribución de la licencia ,La licencia no debe de ser específica para un producto, La licencia no debe restringir otro software, La licencia debe ser tecnológica neutral.

Dentro de este marco de muchas herramientas de código abierto (Framework), se generaron soluciones tecnológicas de software libre, que permitieron que las aplicaciones web, bajo estándares de calidad sean cada vez más flexibles y manejables, lo cual ha contribuido al intercambio y el procesamiento de datos de manera transparente para el usuario final y para las aplicaciones que se utilizan. Las empresas necesitan tener una gran flexibilidad para poder adaptarse ágilmente a la demanda exigente del entorno, por

ello las soluciones tecnológicas de software para las empresas u organizaciones experimentan grandes inconvenientes dentro de un marco de desarrollo, generando así muchas veces la insatisfacción por parte de los usuarios con respecto al producto final, uno de los factores se da en la definición de alcance del proyecto donde no se cumple lo solicitado por el usuario, también podemos mencionar las incongruencias del software que luego pasa a un estado de mantenimiento constante que no es rentable para las empresas y organizaciones.

Otro factor crítico que podemos mencionar, es el incumplimiento del tiempo establecido para el funcionamiento del software, esto genera un impacto sobre el costo de desarrollo el cual se incrementa, además de las necesidades cambiantes de las empresas que conlleva al desarrollo iterativo y cambios de line base. En muchos casos la falta de documento de plan de proyecto y la correcta definición de las especificaciones técnicas conlleva a un producto final poco exitoso.

Debido a estos factores surge incógnitas como: ¿Cuál es el modelo de Framework basado en código abierto java que nos permita la rápida creación de aplicaciones web?, ¿Cómo orientar las aplicaciones web a los componentes de negocio?, ¿Cómo reducir el trabajo físico y de procesamiento en el acceso a datos sin reducir la eficiencia?, Esto dificulta, a su vez, la rápida adaptación de los procesos para poder aprovechar nuevas oportunidades de negocio o responder a las amenazas externas.

En el presente trabajo de investigación emplearemos un modelo de Framework de código abierto java que nos permita la rápida creación de aplicaciones web enfocados a los componentes de Negocio para tener mejor tasa de éxito en los proyectos.

1.2. FORMULACIÓN DEL PROBLEMA DE LA INVESTIGACIÓN

1.2.1. PROBLEMA PRINCIPAL

¿Cómo el **Modelo de Framework de código abierto en java** agiliza el desarrollo de **aplicaciones web empresariales basados en componentes de negocio para entornos agiles, 2019**?

1.2.2. PROBLEMAS ESPECÍFICOS

A. ¿Cómo **Plain Old Java Object (POJOs)** mejora la **funcionalidad del componente de negocio, 2019**?
B. ¿Cómo las **Anotaciones** mejoran la **Interfaz Amigable, 2019**?
C. ¿Cómo el **Java Persistence Api (JPA)** mejoran el **procesamiento de datos, 2019**?

1.3. OBJETIVOS DE LA INVESTIGACIÓN

1.3.1. OBJETIVO GENERAL

Utilizar el **Modelo de Framework de código abierto en Java** para agilizar el desarrollo de **aplicaciones web empresariales basados en componentes de negocio para entornos agiles, 2019**.

1.3.2. OBJETIVOS ESPECÍFICOS

A. Utilizar **Plain Old Java Object (POJOs)** como mecanismo para lograr la **Funcionalidad del componente de negocio**.
B. Utilizar **anotaciones**, como mecanismo para lograr **interfaz amigable**.
C. Utilizar las **Java Persistence API(JPA)** como mecanismo de **procesamiento de datos**.

1.4. JUSTIFICACIÓN Y DELIMITACIÓN DE LA INVESTIGACIÓN

1.4.1. JUSTIFICACIÓN

El presente trabajo de investigación se centraliza en trabajar en un modelo de Framework de código abierto java para el desarrollo rápido de aplicaciones web empresariales basados en los componentes de negocio, esto dentro de los entornos agiles para los desarrolladores.

la implementación de este nuevo marco de trabajo implica la reducción de los tiempos de desarrollo y la optimización de las tareas, además de dar soporte a las mejoras continuas a los proyectos de software.

1.4.2. DELIMITACIÓN

El alcance del presente trabajo tiene un fin académico a nivel local, nacional e internacional, enfocado al mercado de desarrollo de aplicaciones web y la mejora continua.

CAPITULO II

REVISIÓN DE LITERATURA

2.1. ANTECEDENTES DE LA INVESTIGACIÓN

(BARROS, 2002) , en la revista de Ingeniería de Sistemas de la Universidad De Chile, resume lo siguiente: La tendencia actual, al usar software empaquetado, es hacer adaptaciones mínimas del mismo a los procesos de negocios de la empresa y ojalá usarlo tal como es, para acelerar y disminuir el costo de las implementaciones. Esto es evidentemente verdadero, con mayor razón, cuando se utilizan paquetes económicos –que son esencialmente rígidos- en las PYMES. Además, se cree que el enfoque tiene un buen potencial para realizar rediseño de procesos y construir soluciones adaptadas de apoyo a los mismos en las empresas. Esto debido a que la existencia de patrones de procesos de negocios prediseñados, adaptados a un cierto dominio y lógica de negocio con diversas opciones, hace factible que comunidades de empresas usuarias y desarrolladores de software se coordinen en la construcción de componentes genéricos –a partir de los Framework adaptables a situaciones particulares. Evidentemente, esto requiere un acuerdo dentro de un grupo de empresas, lo cual se puede conseguir de varias maneras.

(VILLALOBOS, 2010), en su investigación: *Diseño de Framework web para el desarrollo dinámico de Aplicaciones,* de la universidad Tecnológica de Pereira, concluye lo siguiente: La comunicación entre la base de datos y el usuario en cuanto a entradas y salidas es facilitada por medio de una interfaz web apoyada en una arquitectura Cliente/Servidor. También, Es necesario efectuar la revisión de ciertos detalles de compatibilidad entre las partes principales del sistema, las diferentes librerías que permiten realizar AJAX de una manera Cross-browser, manejo de gestores de plantillas, y acceso dinámico a la Metadata de las tablas en diferentes motores de Base de Datos.

(GÓMEZ, 2010), realizó la investigación: *Sistema De Información Para El Control, Seguimiento Y Mantenimiento Del Equipamiento Hospitalario,* en la Universidad Ricardo Palma, menciona en una de sus conclusiones que, La implementación de un sistema de información para la planificación de los trabajos de mantenimiento, así como para el control de los inventarios del equipamiento hospitalario, permitirá mejorar la gestión de inventarios con el consiguiente beneficio del aumento en

el cumplimiento de la programación de tareas de mantenimiento, lo que permitirá evitar reparaciones costosas y pérdidas de tiempo por la falta de disponibilidad de equipos. Además, recomienda mantener La exactitud de los datos, así como la actualización constante de los mismos por todas y cada una de las áreas, deberá ser una obligación para el éxito del sistema, también de Continuar con las actualizaciones del presente sistema de información una vez implementado, principalmente para optimizar aún más dichos procesos.

(Ojeda, 2008), en su investigación: *Análisis, Diseño e Implementación de un DataWarehouse de Soporte de Decisiones para un Hospital del Sistema de Salud Público*, de la Pontificia Universidad Católica Del Perú, donde menciona una de las conclusiones, que Es muy importante desarrollar una buena fase de análisis para evitar que a lo largo del proyecto surjan problemas que ameriten una reestructuración de los procesos, mapeos o de los reportes mismos. Algunos inconvenientes no saltan a la vista hasta que se tiene el reporte terminado, puesto que saltan incongruencias en los datos del informe o se identifica que los datos no eran agregables y por lo tanto se está presentando información incorrecta. En estos casos, se debe regresar a los procesos anteriores para resolver el problema.

2.2. MARCO TEÓRICO

2.2.1. FRAMEWORK DE CÓDIGO ABIERTO EN JAVA

(M. Degiovannini, 2007) define al Framework como una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

También (ECURED, 2015) define lo siguiente, un Framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, librerías y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

(OSI, 2007), define al código abierto como el Término con el que se conoce al software distribuido y desarrollado libremente, Además (SÁNCHEZ, 2007) menciona que el código abierto es el software que pudiendo obtenerse libremente, puede ser usado, copiado, analizado, modificado y redistribuido nuevamente de forma libre.

(REYES, 2013) define a Java como un lenguaje de programación de alto nivel orientado a objetos, con independencia de plataforma. El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática.

visto las definiciones anteriores se puede decir que el Framework de condigo abierto en java es un marco de trabajo que puede ser organizado y desarrollado bajo una estructura tecnológica que puede ser obtenido y usado con ellenguaje de alto nivel java.

Figura N.º 2.1: Marcos de trabajo de java, (CRBtech,2017).

A. PLAIN OLD JAVA OBJECT (POJOs)

(Johnson, 2005), Enlaza el desajuste de impedancia relacional del objeto con un marco responsable de la asignación de objetos persistentes a filas en un sistema de administración de base de datos relacional, generando todo el código del lenguaje de consulta estructurado necesario para recuperar y almacenar objetos. Es una instancia, objeto Java común de una clase que no extiende ni implementa nada en especial que no requiere ninguna ruta de clase, los pojos se utilizan para aumentar la legibilidad y la reutilización de un programa.

```java
public class Persona {

    private String nombre;
    private String apellido;

    public Persona(String nombre, String apellido) {
        this.nombre = nombre;
        this.apellido = apellido;
    }

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    @Override
    public String toString() {
        return "Persona->{nombre=" + nombre + ";apellido="+apellido+"}";
    }

}
```

Figura N.º 2.2: Implementación de una clase persona (POJO) en java, (CERITDUMBRE,2011).

B. ANOTACIONES

(Vivona, 2011), define lo siguiente: Las anotaciones son artefactos que permiten especificar información extra respecto de una clase, método o variable, directamente en el código y de forma tal que estas pueden ser accedidas como objetos por el sistema. A continuación, se detalla algunos ejemplos de anotaciones en el anexo A.

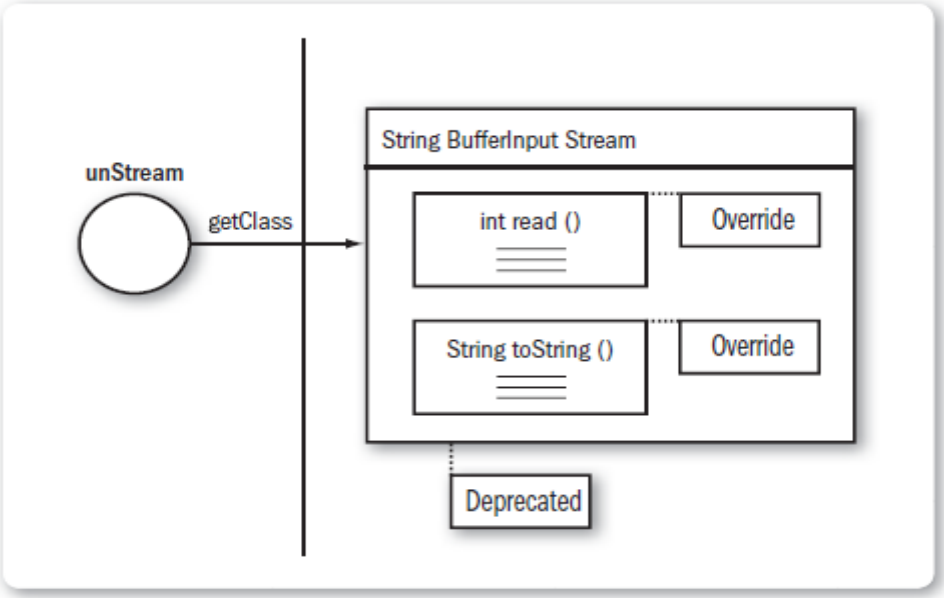

Figura N.º 2.3: Las anotaciones no afectan la semántica de una clase, sino que agregan información a sus elementos, (Vivona, 2011).

ALGUNAS ANOTACIONES CONOCIDAS

@Override

Esta anotación puede, en principio, parecer innecesaria, ya que el compilador sabe si estamos sobrescribiendo un método o no. Es conveniente utilizarla siempre por varios motivos. Por ejemplo, si queremos sobrescribir un método y nos equivocamos al momento de escribir el nombre, el compilador nos alertará de que no estamos sobrescribiendo un método conocido. Solamente está indicada para ser utilizada en tiempos de compilación. También es una forma de asegurarnos de que estamos implementando los métodos de una interfaz correctamente. Esta anotación solo puede ser aplicada sobre métodos.

@Test

Otra anotación que utilizamos frecuentemente es @Test, la cual es usada en nuestros **Test Cases** para indicar cuáles métodos representan pruebas unitarias. Esta anotación puede aceptar algunos argumentos en su uso. Por ejemplo, si deseamos indicar que el test arroja una excepción, lo hacemos con el argumento expected.

```
@Test(expected=IOException.class)
```

También es posible indicar que la prueba debe tardar menos de una determinada cantidad de tiempo o de lo contrario falla. Lo hacemos agregando el argumento timeout y especificando a continuación la cantidad de milisegundos que se debe esperar.

```
@Test(timeout=1000)
// esperamos un segundo o falla
```

Es necesario entender que ambos argumentos pueden ser utilizados al mismo tiempo. La anotación @Test solamente puede decorar métodos que sean públicos y no devuelvan nada (void).

@Deprecate

Esta anotación es utilizada para indicar que un elemento (clase, interfaz, método, etcétera) no debe utilizarse más y que es posible que en futuras versiones sea removido por completo. Generalmente se usa cuando un diseño es actualizado y se conservan los

elementos antiguos para mantener la compatibilidad hacia atrás. Anteriormente esta funcionalidad era ofrecida por un comentario que contenía el texto @deprecated. Tengamos en cuenta que see espera que el compilador alerte al programador de tales usos. Por otro lado, es importante señalar que @Deprecated puede ser utilizada para decorar cualquier elemento.

@SuppressWarnings

Sirve para apuntarle al compilador que debe dejar de indicar alarmas del tipo especificado sobre un elemento determinado. Los tipos de alarma los especificamos utilizando su nombre (en texto) y podemos indicar más de un tipo de alarma al mismo tiempo (con un array).

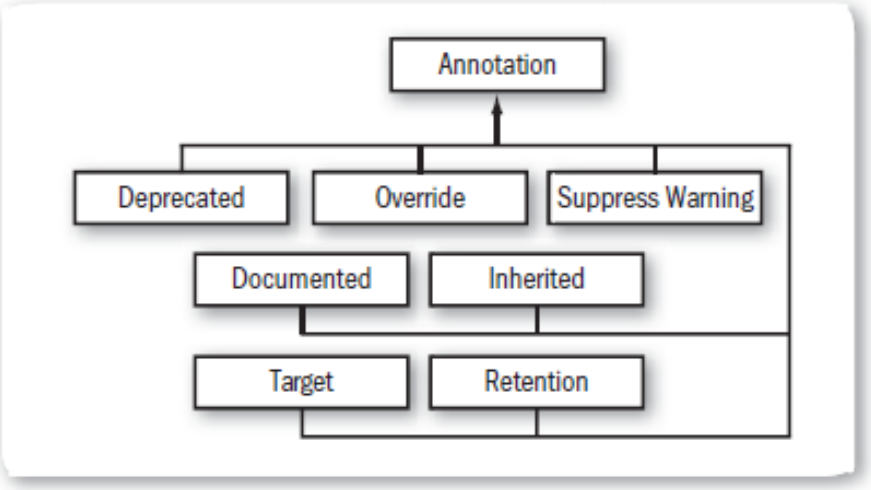

Figura N.º 2.4: Jerarquía de las anotaciones en Java. Podemos notar que está formada por interfaces, (Vivona, 2011).

C. JAVA PERSISTENCE API (JPA)

(IBM, n.d.), Medio por el cual Java puede recuperar información desde un sistema de almacenamiento no volátil, esto es vital para las aplicaciones empresariales debido al acceso necesario a las bases de datos relacionales. Las aplicaciones desarrolladas para este entorno deben gestionar por su cuenta la persistencia o utilizar soluciones de terceros para manejar las actualizaciones y recuperaciones de las bases de datos con persistencia. JPA (Java Persistence API) proporciona un mecanismo para gestionar la persistencia y la correlación relacional de objeto.

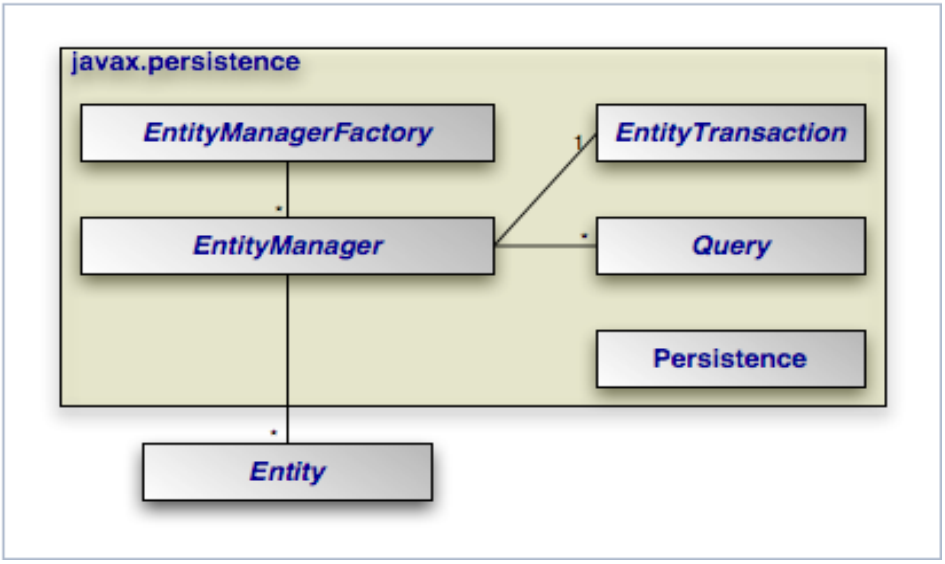

Figura N.º 2.5: Arquitectura de la API de persistencia de Java, (Apache OpenJPA 1.2, 2013).

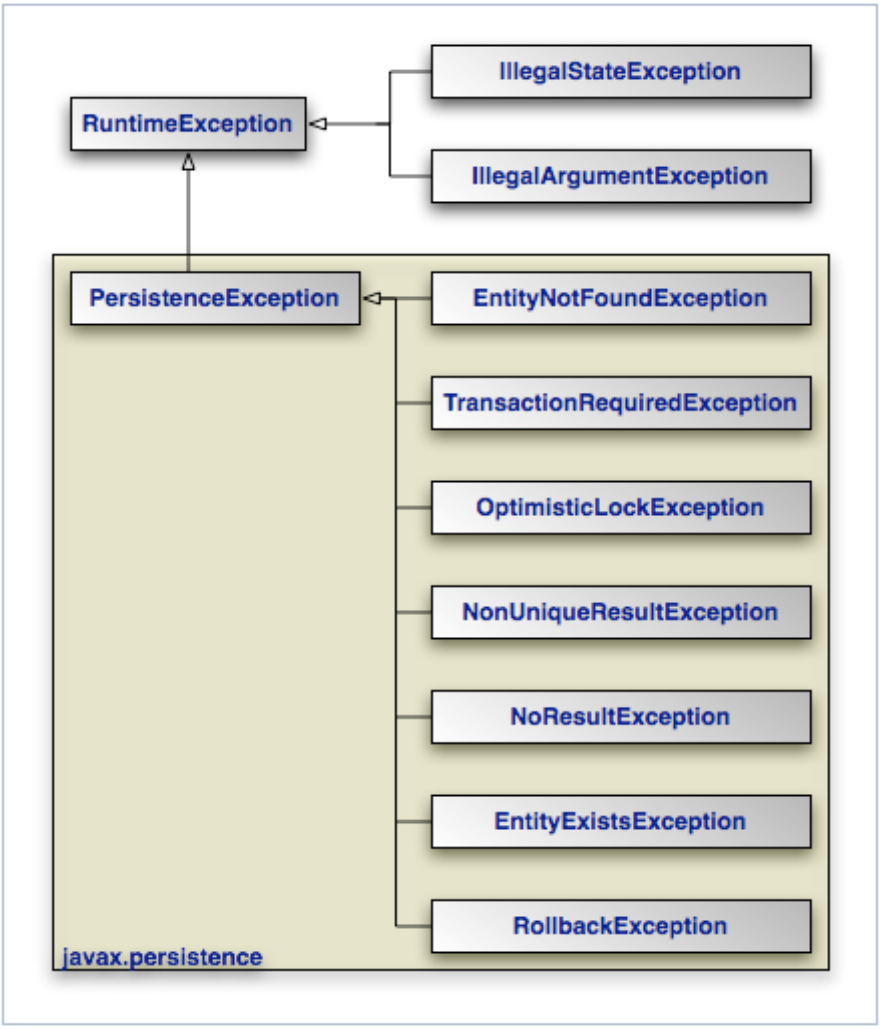

Figura N.º 2.6: Excepciones JPA, (Apache OpenJPA 1.2, 2013).

2.2.2. APLICACIONES WEB EMPRESARIALES BASADAS EN COMPONENTES DE NEGOCIOS PARA ENTORNOS AGILES

(Mora, 2006), las aplicaciones web permiten la generación automática de contenido, la creación de páginas personalizadas según el perfil del usuario o el desarrollo del comercio electrónico.; Además, una aplicación web permite interactuar con los sistemas informáticos de gestión de una empresa, como puede ser gestión de clientes, contabilidad o inventario, a través de una página web. Las aplicaciones web se encuentran dentro de las arquitecturas cliente servidor donde un ordenador solicita servicios (el cliente) y otro está a la espera de recibir solicitudes y las responde (el servidor).

(IBM, 2015), Una aplicación empresarial es una colección de componentes que proporciona una funcionalidad empresarial que se puede utilizar internamente, externamente o con otras aplicaciones empresariales. Puede crear aplicaciones empresariales de componentes individuales, que están relacionados entre sí. Por ejemplo, Gestión de pedidos, Gestión de inventario y Facturación son aplicaciones empresariales que pueden utilizar componentes individuales como un servidor de aplicaciones Java EE y una base de datos que se ejecuta en el servidor.

En los entornos ágiles (Agustin Yagüe y Juan Garbajosa, 2009), las pruebas son el centro de la metodología y, por lo tanto, son ellas las dirigen el proceso de desarrollo. los entornos ágiles plantean que el desarrollo no es un conjunto de fases en las que las pruebas son una fase más, sino que abogan por que las prácticas y el desarrollo estén completamente integradas, lo que puede llevar a modificar las estructuras organizativas de las empresas.

Entonces podemos definir a las aplicaciones web empresariales basadas en componentes de negocios para entornos agiles, como una aplicación web de contenido empresarial que interactúa con otros sistemas de datos empresariales basados en componentes empresariales personalizables de acuerdo a las necesidades y la gestión de datos donde los entornos agiles plantean un conjunto de faces integradas conforme ala estructura de las empresas.

Figura N.º 2.7: Desarrollo de aplicaciones empresariales para móviles y web,

(Universidad de Ciencias y Humanidades, 2013).

A. FUNCIONALIDAD DEL COMPONENTE NEGOCIO

(Oracle, 2013), define como, Componente de negocio, que es una lógica que resuelve o satisface las necesidades de un dominio de negocios en particular, como como banca, comercio minorista o finanzas, se maneja con beans (Clase destinada a almacenar datos) empresariales que se ejecutan en el nivel empresarial o el nivel web.

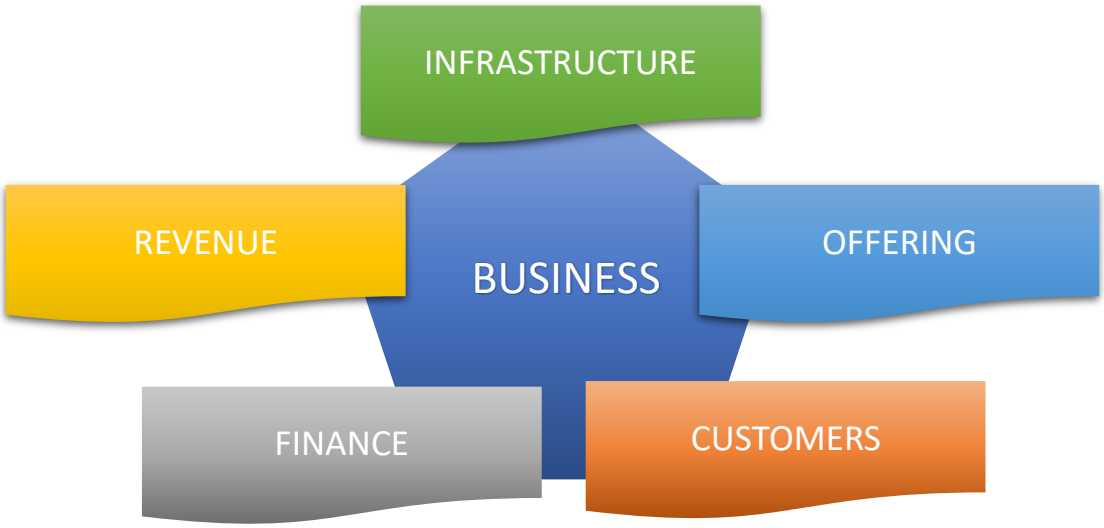

Figura N.º 2.8: Componentes de negocio, (elaboración propia).

B. INTERFAZ AMIGABLE

(Rober-Alonso, 2014) en su investigación titulada: *Estándar De Usabilidad Para La Interfaz Gráfica De Usuario En Los Proyectos De Desarrollo De Software*, de la Universidad Nacional De Loja, concluye que la aplicación del estándar de usabilidad en el desarrollo de interfaces gráficas para aplicaciones Web, permite a desarrolladores tener aplicaciones útiles y fáciles de usar por cualquier usuario.

También, (Rodríguez, 2015) lo define como el objetivo a lograr en este caso es un Diseño Universal, que pretende que nadie se vea limitado en el uso de algo por causa de esas diferencias. Puesto que una gran cantidad de los esfuerzos en interfaz actuales se apoyan en elementos gráficos, resulta lógico ofrecer a los usuarios con visión reducida la opción de utilizar esos elementos en la medida que sea posible. La reducción de la carga visual es el objetivo a lograr.

Figura N.º 2.9: Una interfaz amigable que simplifica la experiencia de usuarios , (SOFOS, 2017).

C. PROCESAMIENTO DE LOS DATOS

(PROJECT, 2018) lo define como la acumulación y manipulación de elementos de datos para producir información significativa. Entonces podemos mencionar que el procesamiento de datos es la Técnica que consiste en la recolección de los datos primarios

de entrada, que son evaluados y ordenados, para obtener información útil, que luego serán analizados por el usuario final, para que pueda tomar las decisiones o realizar las acciones que estime conveniente.

Figura N.º 2.10: Proceso de captura de información, (Fernando Salazar,2018).

2.2.3. METODOLOGÍA ICONIX.

(Rosenberg, 2007) "ICONIX proceso es un análisis de casos de uso basada en el diseño y la metodología. Su foco principal está en cómo llegar de forma fiable a partir de casos de uso de código en menor número de pasos posible".

Para (Porras, 2011) la metodología ICONIX está compuesta por; a) Primer paso.- Identificar el mundo real y los objetivos de dominio del negocio (modelo de dominio), b)Segundo paso.- Definir los requisitos de comportamiento (casos de uso), c)Tercer paso.- Realizar análisis de robustez para eliminar la ambigüedad de los casos de uso y determinar los defectos del modelo de dominio (diagrama de robustez), d) Cuarto paso.- Asignar comportamiento a los objetos (diagrama de secuencia), e) Quinto paso.- Finalizar el modelo estático (diagrama de clases), f) Sexto paso. Escribir y generar el código (código fuente), g) Séptimo paso. - Realizar pruebas de aceptación (prueba). El Proceso de Iconix se divide en flujos de trabajo dinámicos y estáticos, que son altamente repetitivo; usted puede ir a través de una repetición de todo el proceso para una pequeña cantidad de casos de uso (quizás un par de paquetes de valor, que no es una cantidad enorme teniendo en cuenta que cada caso de uso es sólo un par de párrafos), todo el camino a la fuente de código y pruebas unitarias. Por esta razón, el proceso Iconix es muy

adecuado para proyectos ágiles, donde se necesita información rápida sobre factores tales como los requisitos, el diseño, y las estimaciones (Rosenberg, 2007).

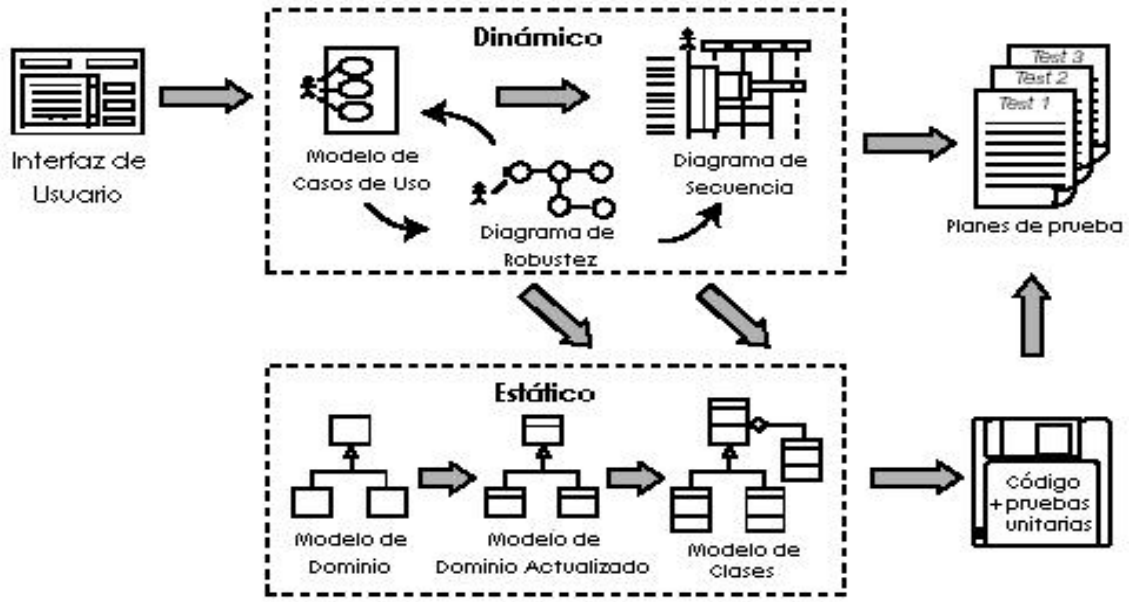

Figura N.º 2.11: Flujo de trabajo Iconix (Rosenberg, 2007)

(Rosenberg, 2007), en el análisis de requisitos se tiene; a) Requisitos funcionales. - Definir lo que el sistema debe ser capaz de hacer. Dependiendo de la forma en que su proyecto está organizado, ya sea que usted participe en la creación de los requisitos funcionales o que los requisitos serán "dictados desde lo alto" de un cliente o un equipo de analistas de negocios, b) Modelo de Dominio.- Entender el espacio del problema en términos inequívocos (sin ambigüedad), c) Requisitos de comportamiento.- Define la forma en que el usuario y el sistema interactúan (es decir, escribir el primer proyecto de casos de uso). Le recomendamos que empezar con un prototipo GUI (lo que llamamos “guion” GUI) e identificar todos los casos de uso que vamos a poner en práctica, o, al menos, llegar a una primera lista de casos de uso, que espera razonablemente cambiar a medida que se explora los requisitos con mayor profundidad. Inicia la etapa 1 con la revisión de requisitos, debe asegúrese de que la descripción de los casos de uso coincida con las expectativas de sus clientes. Tenga en cuenta que se deben revisar los casos de uso en pequeños lotes, justo antes de diseñarlos.

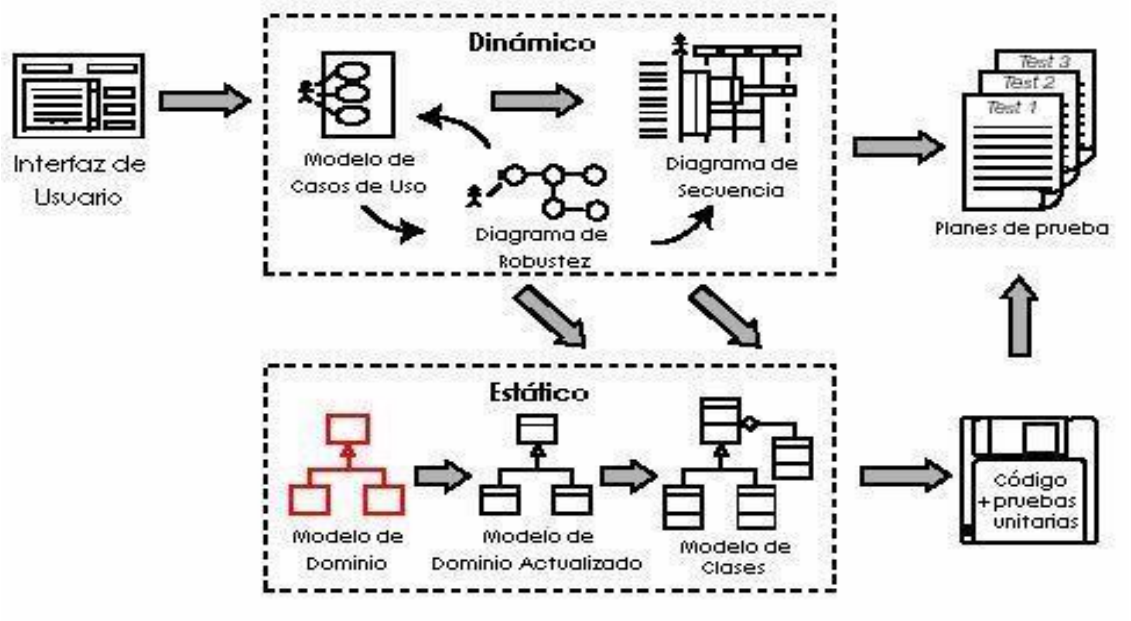

Figura N.º 2.12: Modelo de dominio (Rosenberg, 2007)

Para (Porras, 2011),los modelos de casos de uso son desarrollados en cooperación con el modelo de dominio; los casos de uso sirven de para realizar las pruebas de funcionalidades durante la fase de implementación; un caso de uso es una secuencia de acciones que un actor realiza en el sistema para alcanzar un objetivo.

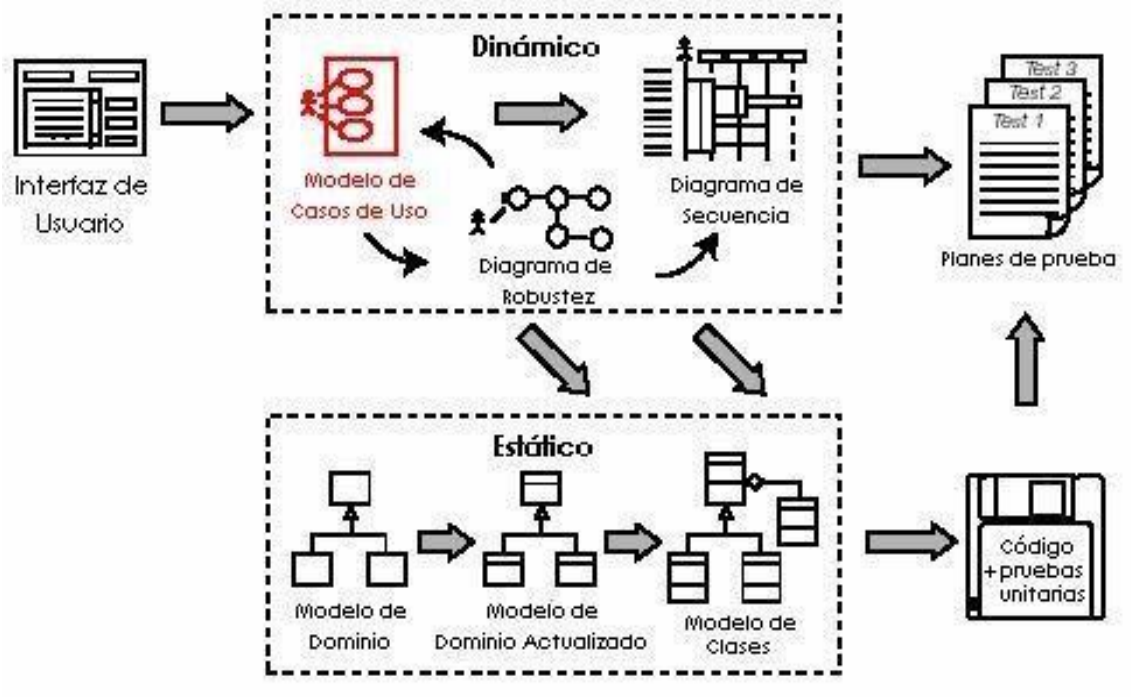

Figura N.º 2.13: Modelo de casos de uso (Rosenberg, 2007)

(Rosenberg, 2007), la revisión de requisitos garantiza que el sistema tal y como se describe coincide con los requisitos. Se trata de un período de sesiones de colaboración que impliquen al representante(s) del cliente, los usuarios finales (es

decir, las personas que realmente van a utilizar el sistema, o quien está usando el sistema actual que se sustituirá), y las personas de marketing-básicamente, todos los stakeholders que tienen un interés en asegurar que los requisitos encajen con su punto de vista del sistema.

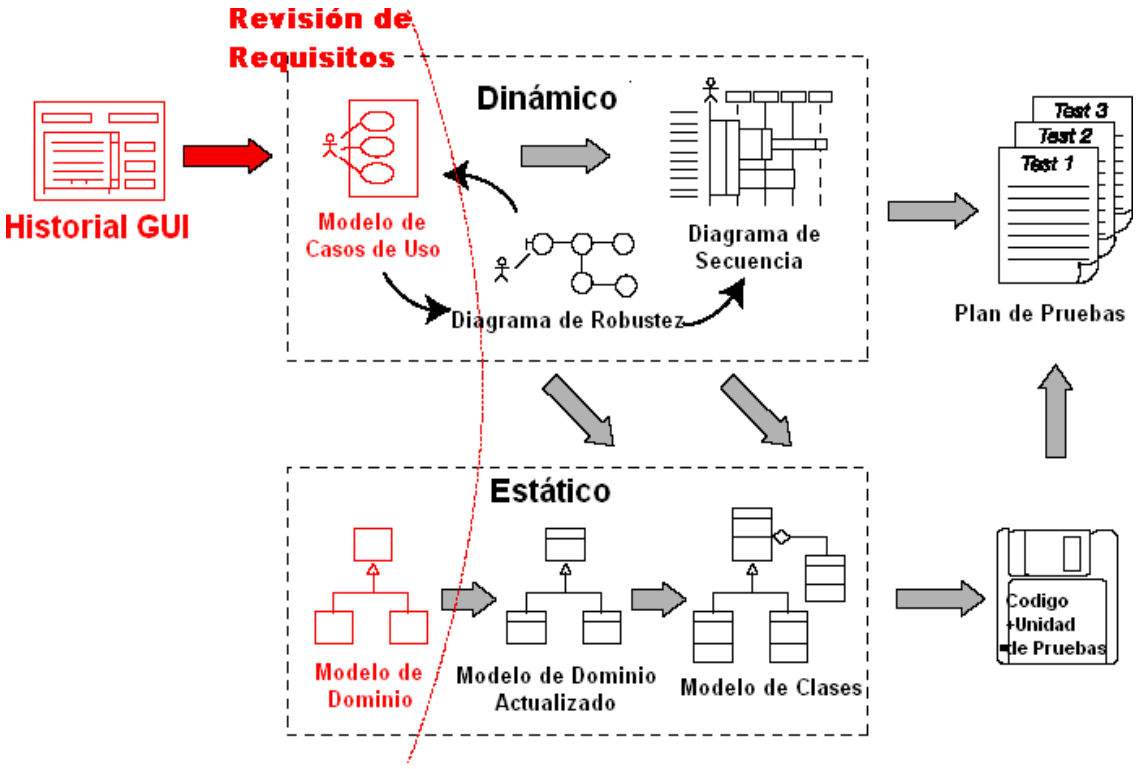

Figura N.º 2.14: Revisión de requisitos (Rosenberg, 2007)

Porras (2011), en el diseño preliminar se tiene; a) Dibujar diagrama de robustez. - Es una "imagen del objeto" descripción paso por paso de cada uno de los casos de uso, reescribir los casos de uso a medida que avanza, b) Actualizar modelo de dominio. - Mientras escribe los casos de uso y dibuja el diagrama de robustez, descubrirá algunas clases "pérdidas", corregir las ambigüedades y, añadir atributos a los objetos de dominio, c) Nombrar controladores. - Nombre todas las funciones lógicas del software, necesarios para que los casos de uso funcionen, d) Escribir. - Reescribir el borrador de los casos de uso. Inicia la etapa 2 con la revisión del diseño preliminar.

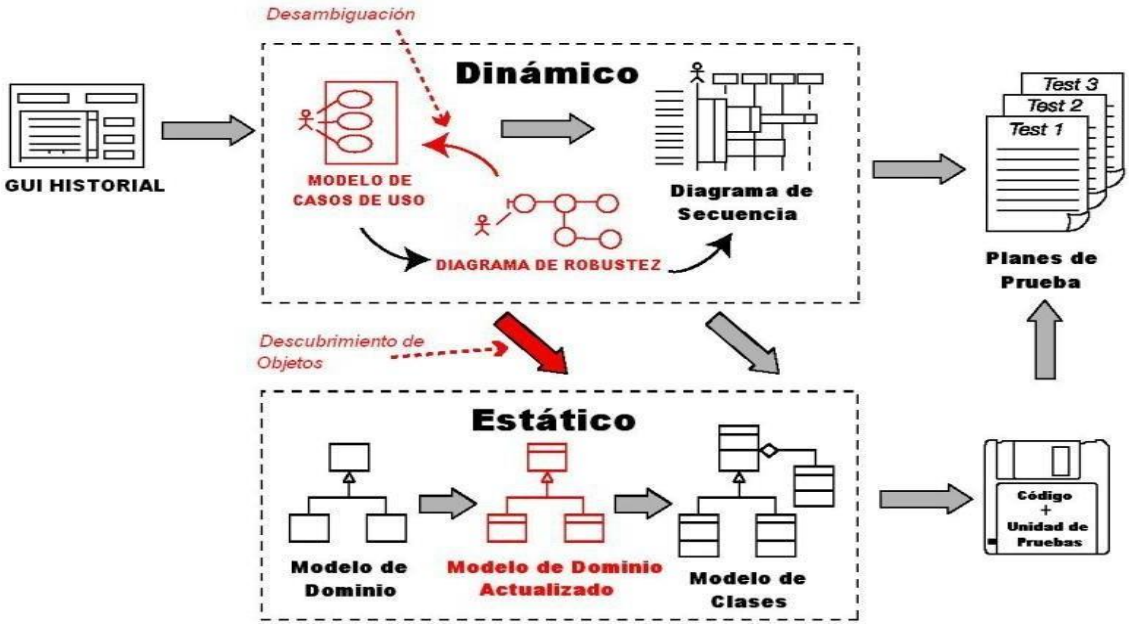

Figura N.º 2.15: Análisis de robustez (Rosenberg, 2007)

Para (Rosenberg, 2007)la revisión de diseño preliminar le ayuda a asegurarse de que los diagramas de robustez, el modelo de dominio, y los casos de uso concuerden mutuamente. Esta revisión es el "paso" entre las fases del Diseño Preliminar y el Diseño Detallado, para cada paquete de casos de uso.

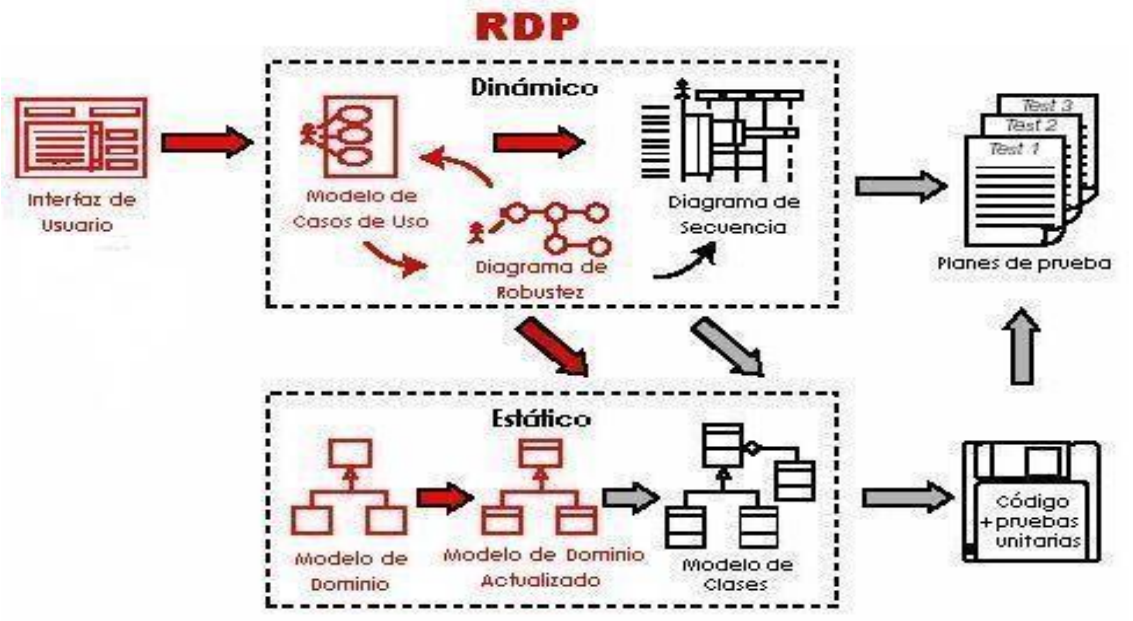

Figura N.º 2.16: Revisión de diseño preliminar (Rosenberg, 2007)

(Rosenberg, 2007), en el diseño detallado se tiene; a) Diagrama de Secuencia. - Dibuje un diagrama de secuencia (un diagrama de secuencia por cada caso de uso) para mostrar en detalle cómo va a implementar cada caso de uso. La función fundamental de los diagramas de secuencia es asignar comportamiento para sus clases, b) Actualice

el modelo de dominio mientras se dibujan los diagramas de secuencia, y añada operaciones a los objetos de dominio. En esta fase, los objetos de dominio son realmente clases de dominio, o entidades, el modelo de dominio debe convertirse rápidamente en un modelo estático, o diagrama de clases-una parte crucial de su diseño detallado, c) Limpiar el modelo estático. Inicia la etapa 3 con la revisión del diseño crítico(RDC).

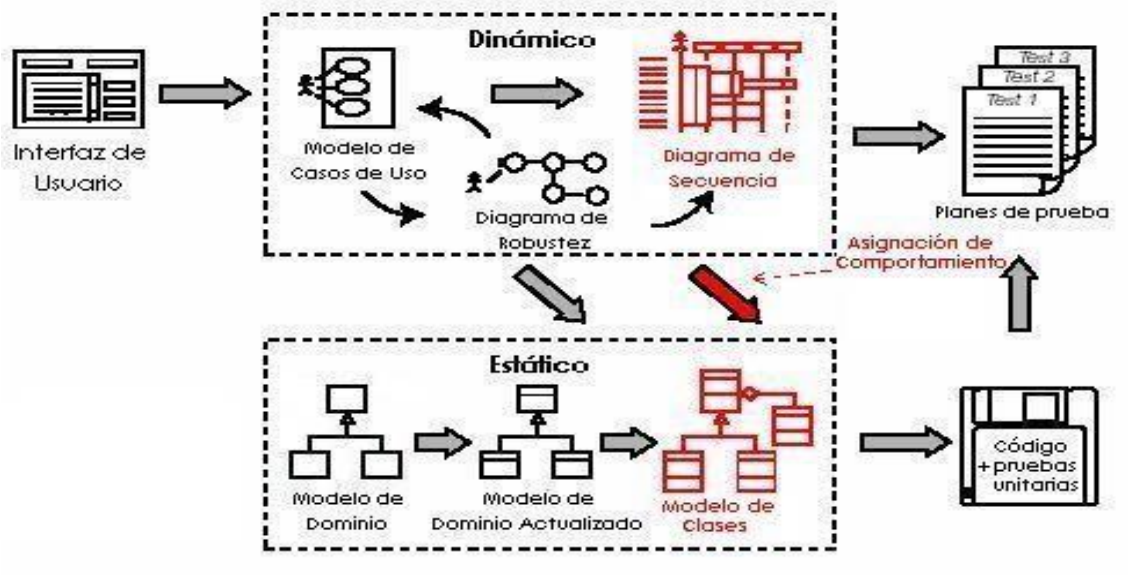

Figura N.º 2.17: Diagrama de secuencia (Rosenberg, 2007)

Para (Porras, 2011),la revisión crítica del diseño es un paso entre el diseño y la implementación, usamos tres criterios; hacer coincidir la descripción de los casos de uso con los diagramas de secuencia, verificar la continuidad de los mensajes y revisar para un buen diseño.

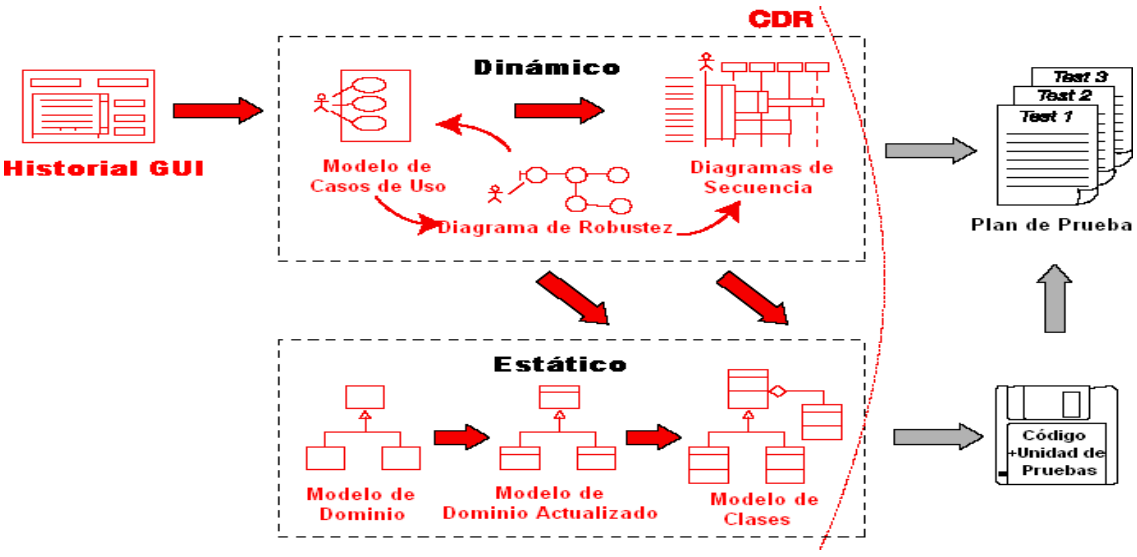

Figura N.º 2.18: Revisión del diseño crítico (Rosenberg, 2007)

(Porras, 2011), en la implementación se tiene; a) Código y pruebas unitarias. - Escribir el código fuente y formular las pruebas unitarias o, escriba las pruebas unitarias y luego el código, b) Integración y pruebas de escenario. - Base las pruebas de integración en los casos de uso, para aprobar los cursos básico y alterno, c) Revisar código y actualizar el modelo. - Luego prepararse para la siguiente iteración del proceso ICONIX, con otro pequeño grupo de casos de uso.

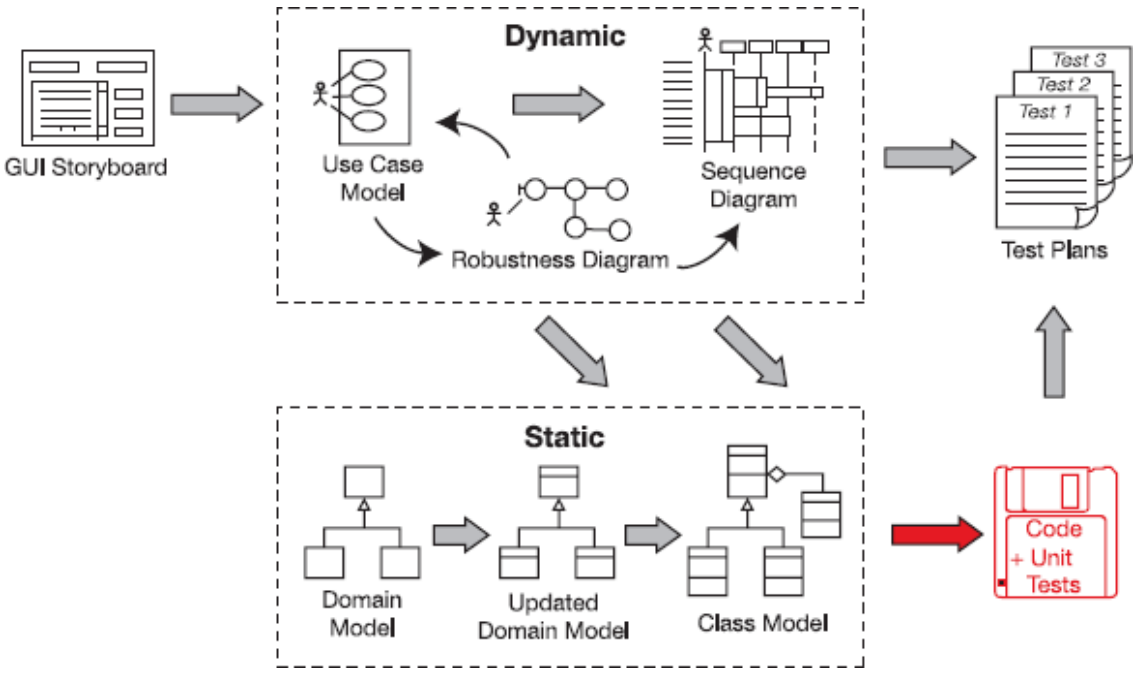

Figura N.º 2.19: Implementación (Rosenberg, 2007)

2.2.4. HIBERNATE

Hibernate es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones. Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL. (Wikipedia, 2017).

Hibernate es un framework que agiliza la relación entre la aplicación y la base de datos. (Hibernate, 2011).

Figura N.º 2.20: Logo de Hibernate ORM (HIBERNATE, 2018).

2.2.5. ECLIPSE IDE

(Scuse, 2018), Eclipse es un IDE (en realidad es más que un IDE, que inicialmente estaba controlado por IBM, pero luego IBM lanzó el código fuente para Eclipse, Haciendo el proyecto de código abierto. hay una gran comunidad de Programadores que proporcionan extensiones a Eclipse para mejorar el entorno de Eclipse.

Figura N.º 2.21: Logo de Eclipse IDE (ECLIPSE FOUNDATION, 2014).

2.2.6. AJAX

(Pérez, 2008), “Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes.” AJAX permite mejorar completamente la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.

2.2.7. APACHE TOMCAT SERVER

Servidor gratuito de código abierto, producto de software que implementa todas las especificaciones Java EE, de manera que al desplegar o instalar una aplicación Java EE en el servidor, sabemos seguro que va a encontrarse con todos los contenedores y servicios definidos por la especificación y que seguramente utiliza y necesita la aplicación, (Jose Miguel oedax Casaa, 2012)

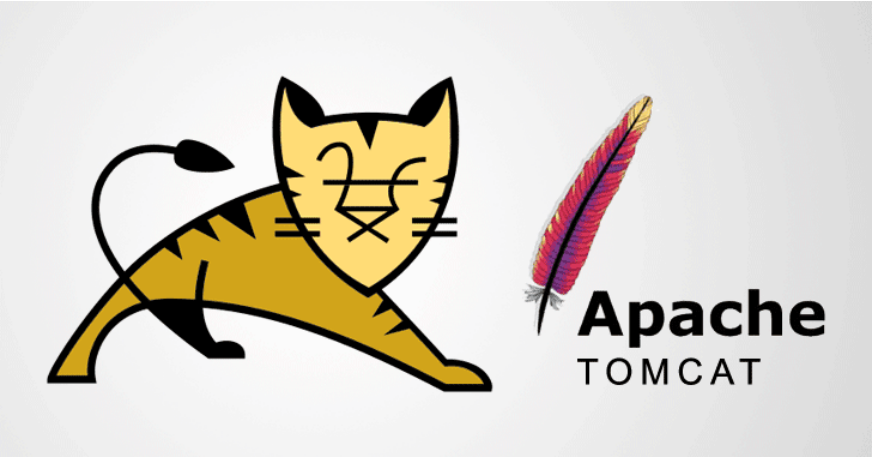

Figura N.º 2.22: Logo de Apache Tomcat IDE (APACHE FOUNDATION, 2019).

2.2.8. MYSQL

(Thibaud, 2006), Es un sistema de administración de bases de datos relacionales (SGBDR) en un entorno de red, especialmente en arquitecturas cliente/servidor. Se proporciona con muchas herramientas y es compatible con muchos lenguajes de programación. Es el más célebre SGBDR del mundo Open Source, gracias a su compatibilidad con el servidor de páginas web Apache y el lenguaje de páginas web dinámicas. Este servidor de bases de datos es interrogable por SQL(Structured Query Languaje), el lenguaje estándar más popular para interrogar bases de datos. Sql permite manipular los datos fácilmente.

Figura N.º 2.23: Logo de Mysql (ORACLE, 2018).

2.2.9. TECNOLOGÍAS DE INTERNET

Internet, también llamado autopista de información, designa un conjunto de redes informáticas relacionadas entre sí, para permitir que los usuarios puedan comunicarse

entre sí; es una red abierta. Su principio básico es la trasmisión de datos de manera fiable entre ordenadores (Esencial C. , 2011).

(Luján, 2001) asevera que “al contrario a otros servicios online, que se controlan de forma centralizada, la Internet posee un diseño descentralizado. Dado que cada ordenador (host) en la Internet es independiente. Los operadores pueden elegir qué servicio usar y qué servicios locales proporcionar” (p.12).

(CAFASSI, 1998) asevera que Internet es una tecnología. Sin embargo, no es la acepción más correcta y su consecuencia se ven a la hora de analizar las implicaciones sociales que conlleva su uso.

A. APLICACIÓN WEB

Para (Luján, 2001), “una aplicación web […] es un tipo especial de aplicación cliente servidor, donde el cliente (el navegador, explorador o visualizador) como el servidor (el servidor web) y el protocolo mediante el que se comunican (HTTP) están estandarizados y no han de ser creados por el programador de aplicaciones” (p.8).

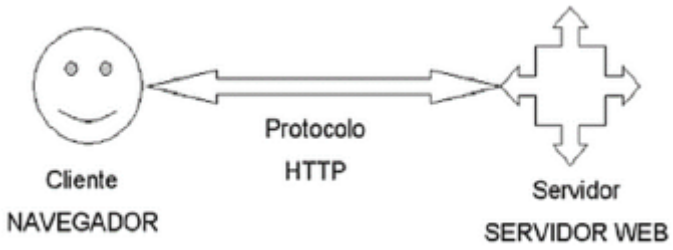

Figura N.º 2.24. Esquema básico de una aplicación web (Luján, 2001)

Según (Suh, 2004), una aplicación web es definido como cualquier programa de aplicación que corra en internet o intranet y extranet corporativa. El usuario de una aplicación web usa un buscador web en una computadora cliente para correr el programa que reside en el servidor.

Una aplicación web generalmente se compone de los siguientes elementos: a) Recursos estáticos: paginas HTML, imágenes, sonidos, hojas de estilo, etc., b) Recursos dinámicos: servlets, JSP, Java Bean, c) Librerías de clases y d) Descriptor de despliegue para definir los parámetros de funcionamiento de la aplicación en el servidor (Groussard, 2010).

TIPOS DE APLICACIÓN WEB

Según (Suh, 2004), hay tres tipos de aplicaciones web: documentos web estáticos, aplicaciones web interactivas simples y sistemas de bases de datos basados en web complejos. Las aplicaciones web estáticas no interactúan o cambian información con los observadores; su propósito es compartir y distribuir información al público. En las aplicaciones web interactivas, los visitantes de los sitios intercambian información con los dueños de la web, muchos de estos sitios usan formularios de respuesta para recoger retroalimentación o evaluación del cliente en sus productos o servicios. Las aplicaciones web complejas manejan transacciones comerciales sofisticadas en línea.

B. SERVIDOR WEB

Un servidor web puede ser descrito por la fórmula servidor web = plataforma + software + información. […] el hardware de la computadora, su sistema operativo y software de la red son la plataforma informática; el segundo componente es el software mismo y el más importante de todos, el servidor web necesita algo que ser servir, sin información que servir, un servidor web tiene pocas razones de existir (Yeager, 1996).

Para (Gourley D. y Totty B., 2002), un servidor web procesa solicitudes HTTP y envía respuestas. El término "servidor web" puede referirse a un software de servidor web o a un dispositivo o computadora en particular dedicado a servir las páginas web. […] la lógica del servidor web implementa el protocolo HTTP, maneja los recursos web y provee las capacidades administrativas del servidor web; y […] comparte responsabilidades para administrar las conexiones TCP con el sistema operativo.

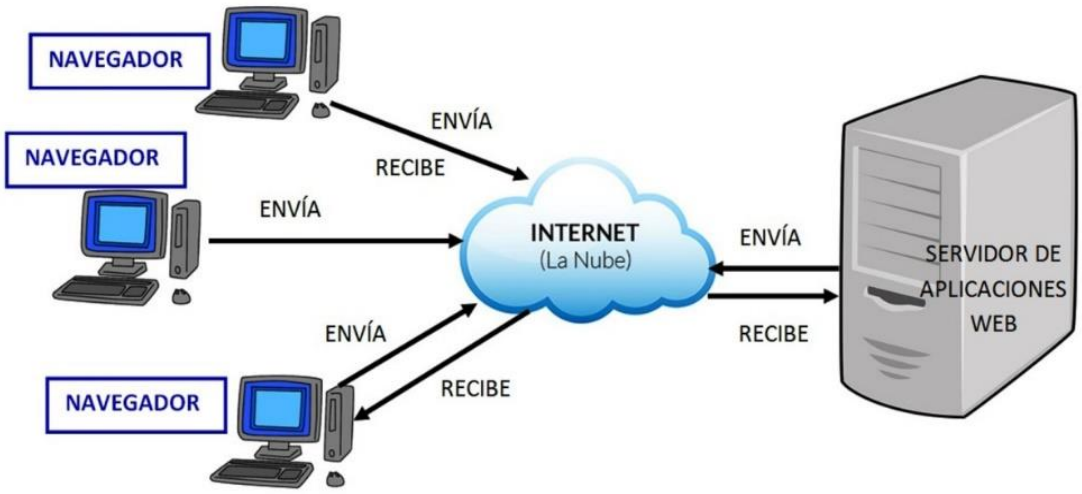

Figura N.º 2.25. Funcionamiento de un servidor web (NICONEWMAN,2016).

C. PROTOCOLO

Proviene del griego “protókolon”, que significa “la primera hoja o tapa, encolada, de un manuscrito importante con notas sobre su contenido” (Antonio, 2001).

HTTP

(Luján, 2001), “el protocolo HTTP forma parte de la familia de protocolos de comunicaciones […] TCP / IP, que son los empleados en internet. Permiten la conexión de sistemas heterogéneos, facilitando el intercambio de información ente distintos ordenadores” (p.8). también “El protocolo de transferencia de hipertexto es un sencillo protocolo cliente–servidor que articula los intercambios de información entre los clientes Web y los servidores HTTP” (Romero, 1997, p. 203).

Para (Kroenke, 2003) , hay dos características importantes en ese protocolo: a) Orientado a peticiones, pues los servidores HTTP esperan peticiones para realizar alguna acción y generar una respuesta y b) No mantienen un estado, ya que reciben una pregunta, la 35 procesan y luego olvidan cuál cliente la formuló.

TCP

Según (Esencial C. , 2011), este protocolo, descompone el mensaje en paquetes y asegura la fiabilidad de la transmisión. Cuando los paquetes llegan a su destino, se agrupan automáticamente para formar un único mensaje, idéntico al original. TCP (Transmission Control Protocol), “utiliza mensajes IP para lograr una transferencia de datos libre de errores. Ambos establecen un diálogo con otro sistema a base de enviar servicios de mensajes IP”

(Romero, 1997). “TCP es un protocolo completo, orientado a la conexión, de transporte fiable para las aplicaciones de TCP/IP. Provee el direccionamiento de una capa de transportes que permite que múltiples aplicaciones de software usen simultáneamente una única dirección IP, y que permite que un par de dispositivos establezcan una conexión virtual y luego pasen datos bidireccionalmente.”

IP

En el protocolo de internet, cada equipo posee una sola dirección única en la red a la que pertenece y cada red posee una dirección única en internet. Por lo tanto, cada

ordenador tiene asignado un nombre y un número IP, y en algunos casos un nombre de dominio (Esencial C. , 2011).

"IP (Internet Protocol) es capaz de enviar mensajes de pequeño tamaño (denominado datagrama) entre dos ordenadores conectados en red. No ofrece garantías de que los mensajes alcancen su destino, debido a los posibles fallos de las redes de comunicaciones." (Romero, 1997).

2.2.10. POBLACION.

Según (Tamayo, 2004)4), la población es la "totalidad de un fenómeno de estudio, que incluye la totalidad de unidades de análisis o entidades de población que integran dicho fenómeno y que debe cuantificarse para un determinado estudio integrando un conjunto N de entidades que participan de una determinada característica, y se le denomina población por constituir la totalidad del fenómeno adscrito a un estudio o investigación" (p.176).

Para (Tomás, J., 2009)la población es el conjunto de todos los individuos que cumplen ciertas propiedades y de quienes deseamos estudiar ciertos datos. Se puede entender que una población abarca todo el conjunto de elementos de los cuales se puede obtener información, entendiendo que todos ellos han de poder ser identificados.

2.2.11. MUESTRA

Para (TAMAYO, 2004), "a partir de la población cuantificada para una investigación se determina la muestra, cuando no es posible medir cada una de las entidades de la población; esta muestra, se considera, es representativa de la población" (p.176).

Para (Tomás, J., 2009)la muestra es una parte o un subconjunto de la población en el que se observa el fenómeno a estudiar y de donde sacaremos unas conclusiones generalizables a toda la población. Se considera que una muestra es grande cuando el número de individuos seleccionados es igual o superior a 30, y una muestra es pequeña cuando los individuos son menos de 30. Además, este autor, también afirma que para que la muestra sea representativa debe cumplir con los siguientes puntos: a) Han de delimitarse y definirse claramente las características que conforman la totalidad de la población, b) Ha de haber garantías de que cada elemento de la población tiene las mismas posibilidades de figurar en la muestra. En consecuencia, debería utilizarse el

procedimiento de muestreo adecuado, y c) La muestra deberá tener el tamaño adecuado para poder extrapolar los resultados obtenidos al conjunto de la población con garantías de fiabilidad.

CAPITULO III

METODOLOGÍA DE LA INVESTIGACIÓN

3.1. TIPO Y NIVEL DE LA INVESTIGACIÓN

3.1.1. TIPO DE INVESTIGACIÓN

Para (TAMAYO, 2004), comprende la descripción, registro, análisis e interpretación de la naturaleza actual, y la composición o procesos de los fenómenos. El enfoque se hace sobre conclusiones dominantes o sobre como una persona, grupo o cosa se conduce o funciona en el presente. La investigación descriptiva trabaja sobre realidades de hecho, y su característica fundamental es la de presentarnos una interpretación correcta.

También (Grajales, 2000) dice lo siguiente, los estudios descriptivos buscan desarrollar una imagen o fiel representación (descripción) del fenómeno estudiado a partir de sus características. Describir en este caso es sinónimo de medir. Miden variables o conceptos con el fin de especificar las propiedades importantes de comunidades, personas, grupos o fenómeno bajo análisis. El énfasis está en el estudio independiente de cada característica, es posible que de alguna manera se integren la medición de dos o más características con el fin de determinar cómo es o cómo se manifiesta el fenómeno. Pero en ningún momento se pretende establecer la forma de relación entre estas características. En algunos casos los resultados pueden ser usados para predecir.

En la presente investigación no pretendemos manipular las variables, por el contrario, pretendemos generar registros análisis e interpretar la naturaleza del modelo Framework de código abierto Java para tratar de producir resultados y una interpretación correcta. De acuerdo a estas consideraciones la investigación es de tipo descriptiva.

3.1.2. NIVEL DE INVESTIGACIÓN

La investigación explicativa según (ROBERTO, 2012), Se encarga de buscar el porqué de los hechos mediante el establecimiento de relaciones causa-efecto. En este sentido, los estudios explicativos pueden ocuparse tanto de la determinación de las causas (investigación postfacto), como de los efectos (investigación experimental), mediante la prueba de hipótesis. Sus resultados y conclusiones constituyen el nivel más profundo de conocimientos.

También (Supo, 2012)menciona que la investigación explicativa son estudios que plantean relaciones de causalidad, donde la estadística no es suficiente para completar sus objetivos, de manera que se tendrá que completan otros criterios de causalidad. El experimento es uno de criterios para demostrar causalidad.

3.2. POBLACIÓN Y MUESTRA

3.2.1. POBLACIÓN

La población de estudio para la presente investigación estuvo compuesta por el Conjunto de librerías del marco de trabajo de código abierto en java.

3.2.2. MUESTRA

La selección de muestra ha sido por conveniencia y estuvo compuesta por el Framework de código abierto en java.

3.3. VARIABLES E INDICADORES

3.3.1. DEFINICIÓN CONCEPTUAL DE LAS VARIABLES

VARIABLE INDEPENDIENTE

Framework de código abierto en java, es un marco de trabajo que puede ser organizado y desarrollado bajo una estructura tecnológica que puede ser obtenido y usado con el lenguaje de alto nivel java.

VARIABLE DEPENDIENTE

Aplicaciones web empresariales basadas en componentes de negocios para entornos agiles, aplicación web que se encuentran dentro de las arquitecturas cliente servidor donde un ordenador solicita servicios (el cliente) y otro está a la espera de recibir solicitudes y las responde (el servidor)., esto orientado a hacia los componentes de negocio para plataformas agiles.

3.3.2. DEFINICIÓN OPERACIONAL DE LAS VARIABLES

VARIABLE INDEPENDIENTE

X: **Framework de código abierto en java**.

INDICADORES

X1 Plain Old Java Object (POJOs)

X2 Anotaciones

X3 Java Persistence API (JPA)

VARIABLE DEPENDIENTE

Y: **Aplicaciones web empresariales basadas en componentes de negocios para entornos agiles.**

INDICADORES

Y1: Funcionalidad del componente negocio

Y2: Interfaz amigable

Y3: Procesamiento de los datos

3.4. TÉCNICAS E INSTRUMENTOS PARA RECOLECTAR INFORMACIÓN

3.4.1. TÉCNICAS

Según (Bernal, C. , 2006), la investigación descriptiva se soporta principalmente en técnicas como la encuesta, la entrevista, la observación y la revisión documental. Por lo tanto, en esta investigación se utilizó la técnica de observación a las herramientas Framework java existentes para recolectar información sobre el procesamiento de datos, las interfaces amigables y sobre todo el tiempo de creación de las aplicaciones web empresariales. y también se utilizó el análisis documental de algunos Framework existentes en el marcado de desarrollo de software para obtener un logro comparativo acerca del estudio.

3.4.2. INSTRUMENTO

Según el apartado anterior, el instrumento guía de observación usado con Frameworks del lenguaje java con el fin de generar información sobre procesamiento de datos, las interfaces amigables y sobre todo el tiempo de creación de las aplicaciones web empresariales se encuentra en el anexo B, la guía de análisis documental de estas herramientas Frameworks usada con fin de generar información sobre procesamiento de datos, las interfaces amigables y sobre todo el tiempo de creación de las aplicaciones web empresariales se encuentra en el anexo C.

3.5. HERRAMIENTAS PARA EL TRATAMIENTO DE DATOS E INFORMACIÓN

Las herramientas tecnológicas que se utilizan son seleccionadas tomando en cuenta la facilidad y rapidez en su desarrollo, que facilite administrar con facilidad la complejidad de la aplicación, desarrollo de aplicaciones aplicando estándares y nos permita utilizar técnicas para asegurar la información crítica. Se ha seleccionado las tecnologías según la tabla.

SOFTWARE	FABRICANTE	SERVICIO
Windows 10 pro	Microsoft corporation	Sistema operativo para escritorio, da la facilidad de instalar herramientas de desarrollo.
Eclipse ide	IBM	Plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar "Aplicaciones de Cliente Enriquecido"
Mysql	MySQL AB, Sun Microsystems y Oracle Corporation	Sistema de gestión de bases de datos relacional desarrollado bajo licencia dual Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base datos de código abierto más popular del mundo,
Java	Sun Microsystems	Lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible
Sparx Systems Enterprise architect	Sparx Systems	Herramienta de diseño y modelado visual basada en OMG UML. La plataforma soporta: el diseño y construcción de sistemas de software; modelado de procesos de negocio; y modelado de dominios basados en la industria.
MySQL Workbench	Oracle Corporation	Es una herramienta de diseño de base de datos visual que integra el desarrollo de SQL, la administración, el diseño de la base de datos, la creación y el mantenimiento en un único entorno de desarrollo integrado para el sistema de base de datos MySQL.

Tabla N.º 3.1. Herramientas tecnológicas para tratamiento de datos.

3.6. TÉCNICAS PARA APLICAR ICONIX

Revisando el marco teórico de la metodología Iconix desarrollado en el capítulo II, se formula las faces para desarrollar la aplicación web usando la metodología.

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Identificar requisitos	Requisito funcionales y no funcionales Casos de prueba	Entrevistas Definir lo que el sistema debe hacer Escribir al menos un caso de prueba para cada requisito	Usuario Cliente Analista
Identificar objetos del mundo real y dibujar modelo de dominio	Modelo de dominio	Identificar clases clave del negocio Identificar sustantivos y depurar Identificar objetos en requisitos funcionales y asignar al modelo de dominio Utilizar agregación y generalización	Analista
Realizar prototipo de interfaz gráfica	Prototipo GUI	Utilizar historia de eventos del usuario Utilizar los requisitos funcionales Diseñar interfaz gráfica básica	Programador analista
Descubrir casos de uso	Lista de casos de uso	Utilizar requisitos funcionales Entrevistas	Usuario Cliente Analista
Dibujar y empaquetar casos de uso	Diagrama de casos de uso Paquete entre requisitos funcionales y casos de uso	Identificar roles y responsabilidades de actores Asociar actores con casos de uso Relacionar casos de uso Agrupar lógicamente casos de uso	Analista

Asignar requisitos funcionales a los casos de uso	Relación entre requisitos funcionales y casos de uso	Asignar requisitos funcionales a los casos de uso	Analista
Escribir el primer borrador de casos de uso	Primer borrador de casos de uso	Utilizar glosario de objetos del modelo de dominio. Utilizar la regla de dos párrafos Escribir el caso de uso como flujos de evento/respuesta Escribir el caso de uso con estructura sustantivo-verbo-sustantivo Escribir caso de uso en voz activa Referenciar por su nombre a las pantallas	Analista

Tabla N.º 3.2. Análisis de requisitos (Porras, 2011).

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Revisar el modelo de dominio	Modelo de dominio	Identificar al menos el 80% de clases clave de dominio del problema	Usuario Cliente Analista Programador
Revisar el prototipo GUI	Prototipo GUI	Diseñar con precisión la GUI relacionada al caso de uso	Usuario Cliente Analista Programador
Revisar el modelo de	Casos de uso revisado	Eliminar clases fuera del dominio del problema	Usuario Cliente Analista

casos de uso		Cambiar descripción de voz pasiva a activa Describir todos los cursos alternos Asociar todos los requisitos a los casos de uso Describir que intenta hacer el usuario para cada caso de uso.	Programador

Tabla N.º 3.3. Revisión de requisitos (Porras, 2011).

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Reescribir el primer borrador para cada caso de uso	Casos de uso desambiguado	Rescribir el caso de uso durante el análisis de robustez	Analista
Identificar el primer corte de objetos que completan escenarios para cas caso de uso	Diagrama de robustez	Copiar la descripción del caso de uso en el diagrama de robustez Usar clases del modelo de dominio Crear un objeto interfaz por cada GUI y nombrarlo Transformar verbos del caso de uso en controladores Relacionar un caso de uso al diagrama de robustez cuando es invocado Utilizar las reglas para construir el diagrama de robustez	Analista

Actualizar el modelo de dominio	Modelo de dominio actualizado	Actualizar el modelo de dominio con nuevas clases atributos durante el análisis de robustez	Analista
Actualiza el diagrama de clases de análisis	Modelo de dominio actualizado	Actualizar el diagrama de clases de análisis al finalizar el análisis de robustez Asignar atributos a las clases entidad	Analista

Tabla N.º 3.4. Diseño preliminar (Porras, 2011).

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Revisar descripción de casos de uso	Caso de uso	Coincidir la descripción del caso de uso con el diagrama de robustez.	Usuario Cliente Analista Programador
Revisar diagrama de robustez	Diagrama de robustez	Coincidir el diagrama de robustez con descripción de caso de uso. Verificar que el diagrama de robustez cumple con las reglas. Verificar que el diagrama de robustez tener todos los cursos alternos.	Usuario Cliente Analista Programador
Revisar modelo de dominio actualizado	Modelo de dominio actualizado	Coincidir los objetos entidad del diagrama de robustez con el modelo de dominio actualizado.	Usuario Cliente Analista Programador

Tabla N.º 3.5. Revisión de diseño preliminar (Porras, 2011).

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLE
Dividir modelo de dominio actualizado para cada caso de uso	Parte de modelo de dominio actualizado	Coincidir las clases entidad del diagrama de robustez con parte del modelo de dominio actualizado y dibujado	Diseñador
Dibujar un diagrama de secuencia para cada caso de uso	Diagrama de secuencia	Copiar la descripción del caso de uso Copiar objetos entidad, interfaz y actores del diagrama de robustez Verificar que un mensaje del diagrama de secuencia es verbo en el caso de uso Hacer refactoring al diagrama de secuencia antes de codificar.	Programador Diseñador
Actualizar el diagrama de un caso de uso	Diagrama de clase	Asignar operaciones a las clases a partir de los mensajes del diagrama de secuencia Establecer multiplicidad en las clases Depurar la clases, operaciones y atributos del diagrama de clases	Programador Diseñador
Extraer controladores para pruebas unitarias	Lista de controladores.	Identificar controladores para la lógica del negocio desde un diagrama de robustez.	Programador Diseñador

Tabla N.º 3.6. Diseño (Porras, 2011).

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLE
Revisar diagrama de secuencia	Diagrama de secuencia.	Verificar que el diagrama de secuencia coincide con la descripción del caso de uso Verificar que el diagrama de secuencia representa los cursos básicos y alterno Verificar en los mensajes que los atributos y valores de retorno son correctos	
Revisar diagrama de clases	Diagrama de clases	Verificar que el nombre, atributos y operaciones que asignaron correctamente a las clases. Asignar requisitos no funcionales a los casos de uso y clases	Diseñador Programador
Revisar modelo de dominio actualizado	Modelo de dominio actualizado	Verificar nombres y atributos del modelo dominio actualizado	
Revisar lista de pruebas unitarias	Lista de controladores	Actualizar la lista de controladores	

Tabla N.º 3.7. Revisión crítica de diseño (Porras, 2011).

CAPITULO IV

ANÁLISIS Y RESULTADOS DE LA INVESTIGACIÓN

4.1. ANÁLISIS DE REQUISITOS

Son las características que un software debe tener para poder soportar y/o ejecutar una aplicación. Por lo cual a continuación mostramos los resultados de los artefactos de acuerdo a la metodología Iconix, aplicado al caso de estudio, sistema de facturación de una librería.

4.1.1. REQUERIMIENTOS FUNCIONALES

N.º REQ.	REQUISITOS FUNCIONALES
RF01	El sistema debe permitir la autenticación de los **usuarios**.
RF02	El sistema debe permitir agregar **clientes** para agilizar el proceso de **venta** (facturación, boleta).
RF03	El sistema debe permitir eliminar, actualizar la lista de **clientes** del sistema.
RF04	El sistema debe de guardar, actualizar, editar, eliminar la **dirección** de los **clientes** de manera **detallada**.
RF05	El sistema debe permitir visualizar **reportes** de los **clientes** del sistema.
RF06	El sistema debe permitir eliminar **autores** del sistema.
RF07	El sistema debe permitir modificar datos de los **autores**.
RF08	El sistema debe permitir consultar ciertos datos de los **autores**.
RF09	el sistema debe permitir mantener **categorías** de **productos**
RF10	El sistema ordenara los productos mediante **categorías** (libro, revista, manual, separata).
RF11	El sistema debe permitir eliminar **categorías** de **productos**
RF12	El sistema debe permitir actualizar **categoría** de **productos**
RF13	El sistema debe permitir realizar ventas con la emisión de **comprobantes** (factura, boleta, etc.).
RF14	El sistema debe permitir guardar un comprobante.
RF15	El sistema debe permitir reversar un comprobante previamente emitida.
RF16	El sistema debe permitir agregar **productos**.

RF17	El sistema debe permitir eliminar **productos** del sistema.
RF18	El sistema debe permitir modificar ciertos datos de los **productos**.
RF19	El sistema debe permitir consultar ciertos datos de los **productos**.
RF20	El sistema debe permitir agregar detalles del **producto**
RF21	El sistema debe permitir modificar ciertos **detalles de los productos**.
RF22	El sistema debe permitir visualizar reportes de los **productos**.
RF23	El sistema debe permitir guardar, editar, eliminar **imagen** de los **productos**.
RF24	El sistema debe permitir guardar, editar, eliminar una galería de imágenes de los **productos**.

Tabla N.º 4.1. Tabla Requisitos funcionales.

4.1.2. REQUERIMIENTOS NO FUNCIONALES

N.º REQ.	REQUISITOS NO FUNCIONALES
RNF01	El lenguaje de programación debe ser java
RNF03	Procesamiento de datos muy rápido y eficaz.
RNF04	Interfaz gráfica amigable para el usuario
RNF05	El sistema operativo debe de Windows 7 en adelante
RNF06	El sistema debe presentar una arquitectura y codificación usando estándares que permita su operación y mantenimiento adecuado
RNF07	La base datos no representará ningún gasto por el licenciamiento
RNF08	El sistema debe utilizar el idioma castellano para los mensajes y textos en la interfaz.
RNF09	La resolución mínima para una buena visualización y ejecución del sistema será un tamaño de pantalla de 800x600 pixel.
RNF10	Los reportes serán exportables en formato PDF.

Tabla N.º 4.2. Tabla Requisitos no funcionales.

N.º	Glosario de Términos
1	usuario
2	autor
3	categoría
4	factura
5	boleta
6	Cliente
7	Detalle Venta
8	Producto
9	Revista
10	Libro
11	Reporte
12	Manual
13	Venta
14	Emitir comprobante

Tabla N.º 4.3. Tabla Glosario de Terminos.

4.1.3. MODELADO DE DOMINIO

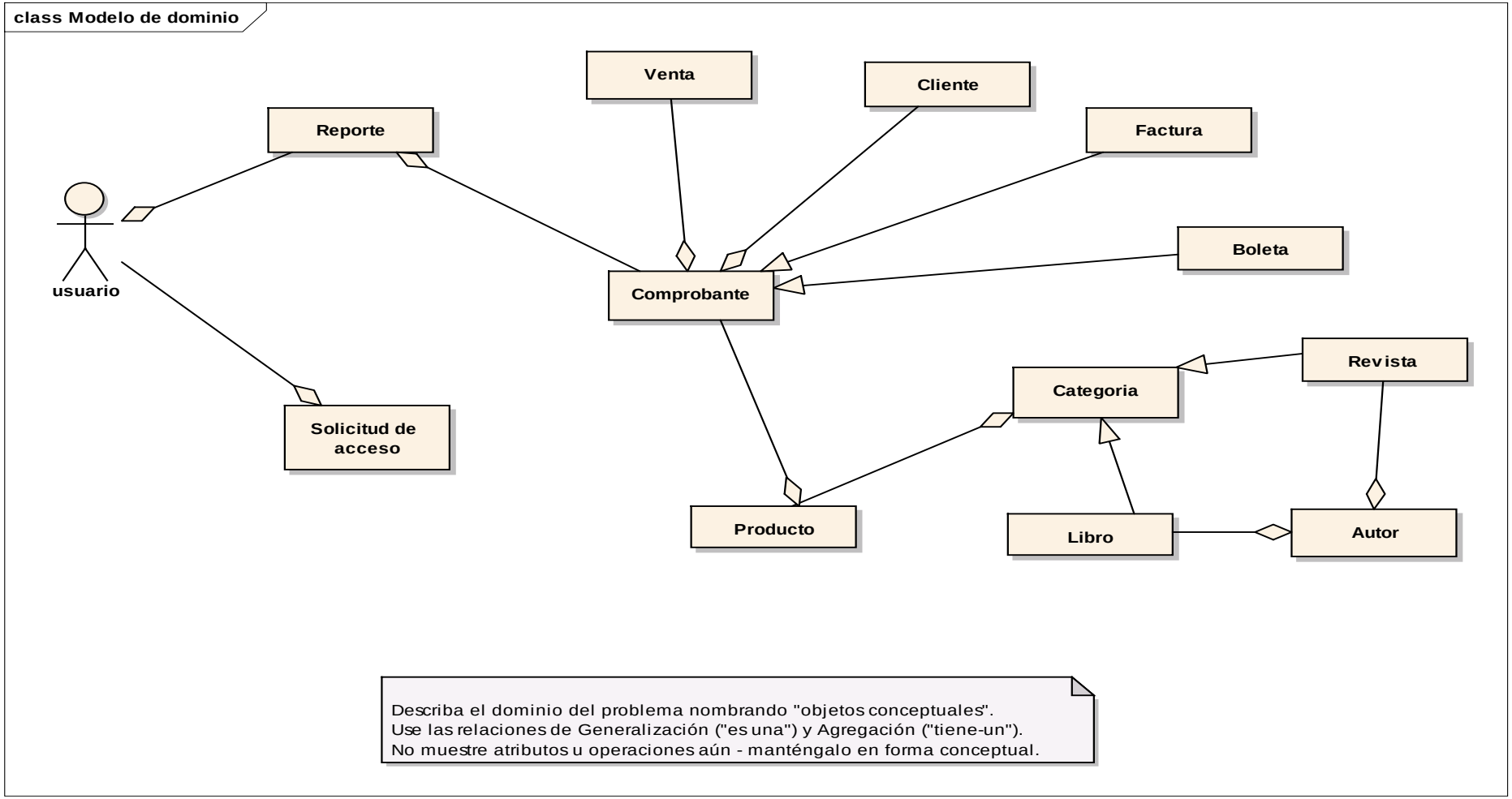

Figura N.º 4.1. Modelo de dominio inicial.

4.1.4. RELACIÓN ENTRE REQUISITOS FUNCIONALES Y CASOS DE USO

REQUISITOS FUNCIONALES		CASOS DE USO
RF01	El sistema debe permitir la autenticación de los **usuarios**.	CU01: Autenticar Usuario
RF02	El sistema debe permitir agregar **clientes** para agilizar el proceso de **venta** (facturación, boleta).	CU02: Mantener Clientes
RF03	El sistema debe permitir eliminar, actualizar la lista de **clientes** del sistema.	
RF04	El sistema debe de guardar, actualizar, editar, eliminar la **dirección** de los **clientes** de manera **detallada**.	CU03: Generar Reportes
RF05	El sistema debe permitir visualizar **reportes** de los **clientes** del sistema.	
RF06	El sistema debe permitir eliminar **autores** del sistema.	CU05: Mantener Autores
RF07	El sistema debe permitir modificar datos de los **autores**.	
RF08	El sistema debe permitir consultar ciertos datos de los **autores**.	
RF09	el sistema debe permitir mantener **categorías** de **productos**	CU06: Mantener Categoría
RF10	El sistema ordenara los productos mediante **categorías** (libro, revista, manual, separata).	
RF11	El sistema debe permitir eliminar **categorías** de **productos**	
RF12	El sistema debe permitir actualizar **categoría** de **productos**	
RF13	El sistema debe permitir realizar ventas con la emisión de **comprobantes** (factura, boleta, etc.).	CU07: Realizar Venta
RF14	El sistema debe permitir guardar un comprobante.	
RF15	El sistema debe permitir reversar un comprobante previamente emitida.	CU08: Mantener Comprobante

RF16	El sistema debe permitir agregar **productos**.	
RF17	El sistema debe permitir eliminar **productos** del sistema.	
RF18	El sistema debe permitir modificar ciertos datos de los **productos**.	
RF19	El sistema debe permitir consultar ciertos datos de los **productos**.	
RF20	El sistema debe permitir agregar detalles del **producto**	CU09: Mantener
RF21	El sistema debe permitir modificar ciertos **detalles de los productos**.	Producto
RF22	El sistema debe permitir visualizar reportes de los **productos**.	
RF23	El sistema debe permitir guardar, editar, eliminar imagen de los **productos**.	
RF24	El sistema debe permitir guardar, editar, eliminar una galería de imágenes de los **productos**.	

Tabla N.º 4.4. Relación de requisitos y casos de uso.

4.1.5. LISTA DE CASOS DE USO

NUMERO	CASOS DE USO
CU01	Autenticar Usuario
CU02	Mantener Clientes
CU03	Generar Reportes
CU04	Mantener Producto
CU05	Mantener Autores
CU06	Mantener Categoría
CU07	Realizar Venta
CU08	Mantener Comprobante

Tabla N.º 4.5. Listado de casos de uso.

4.1.6. DIAGRAMA DE CASOS DE USO

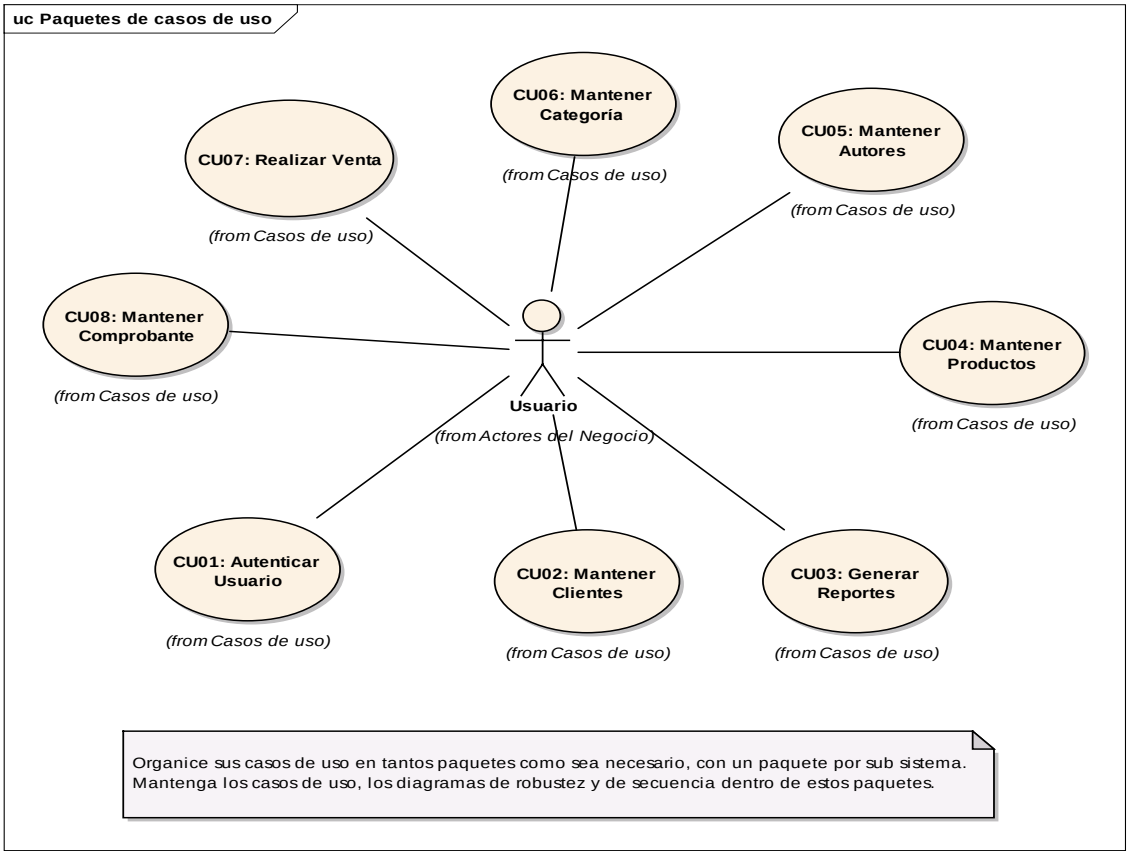

Figura N.º 4.2. Diagrama de Casos de Uso.

4.1.7. PROTOTIPO DE INTERFAZ GUI.

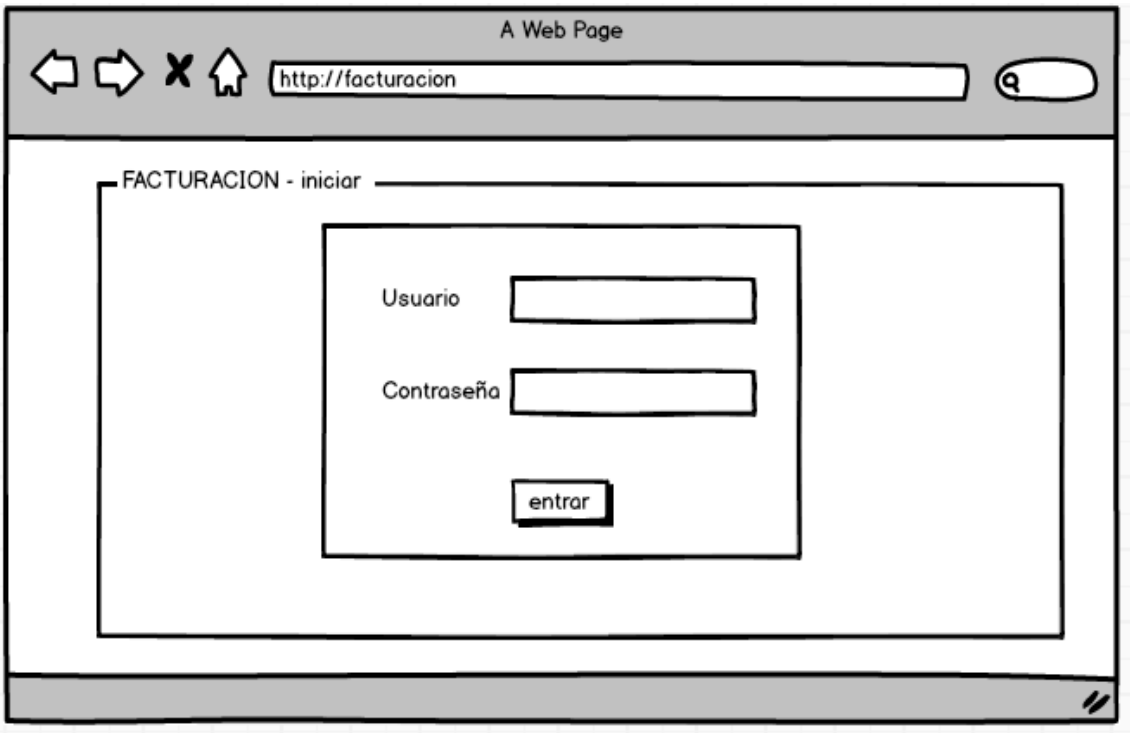

Figura N.º 4.3. CU01 Autenticar Usuario

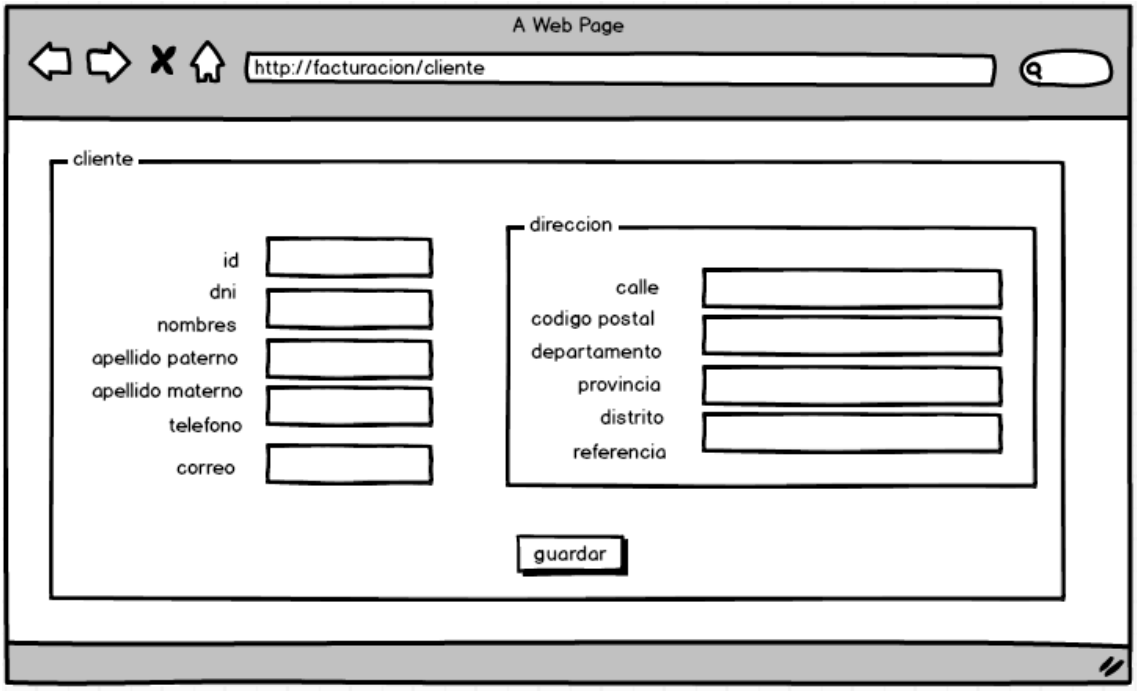

Figura N.º 4.4. CU02 Mantener Clientes

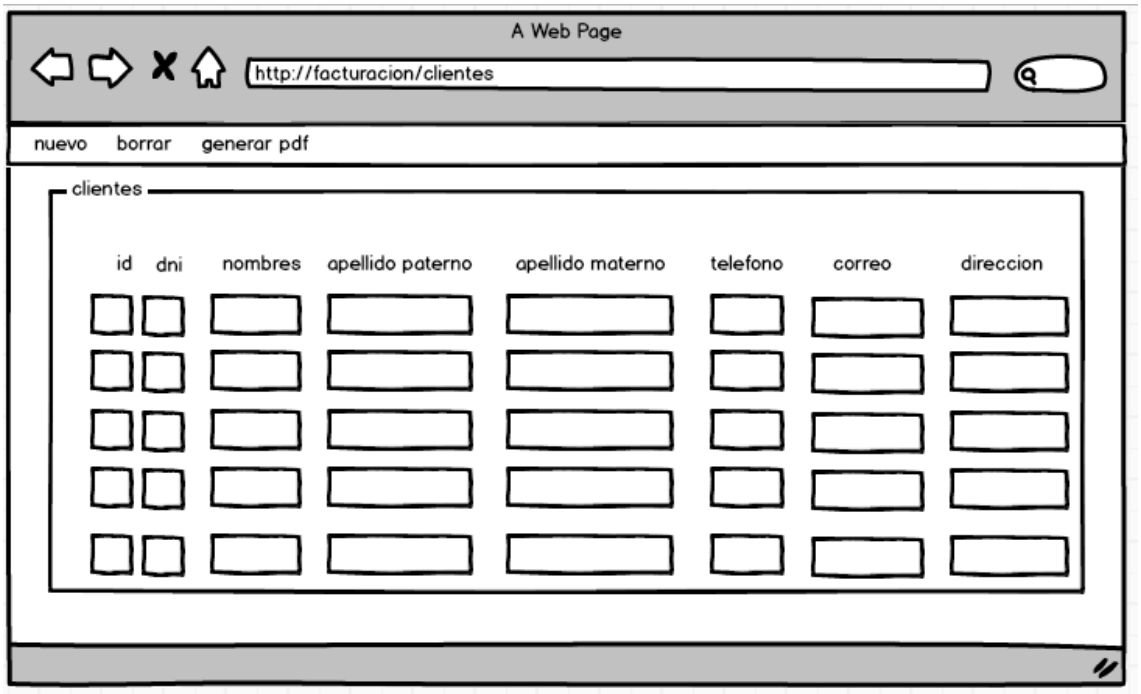

Figura N.º 4.5. CU02 Mantener Clientes.

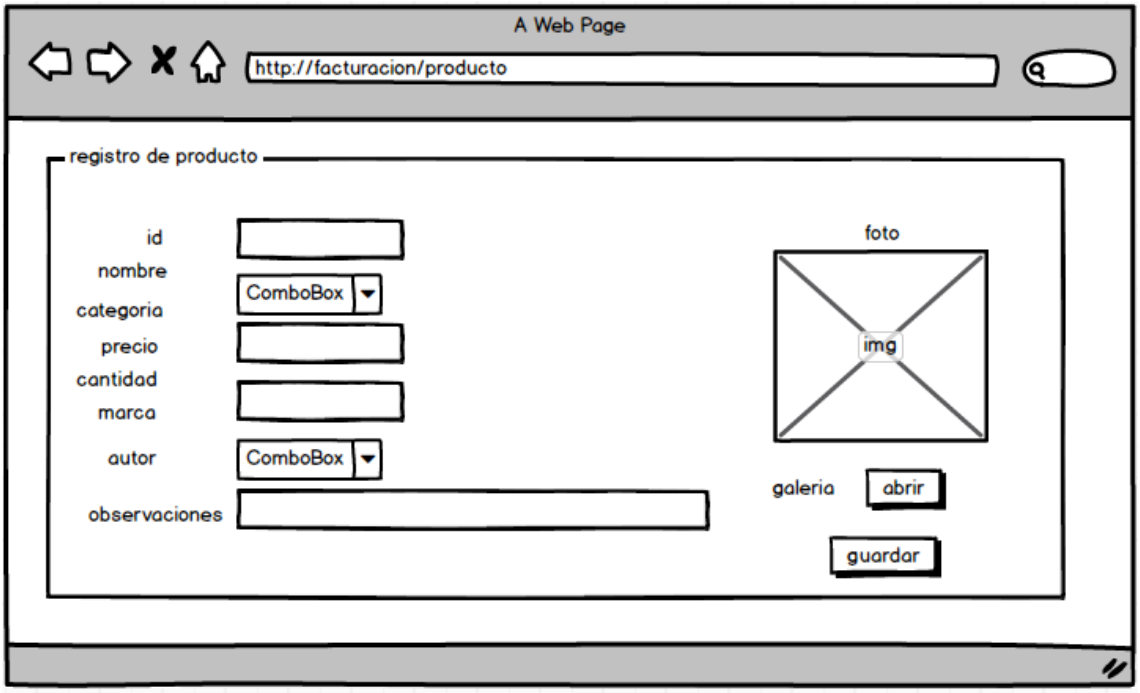

Figura N.º 4.6. CU04. Mantener Producto

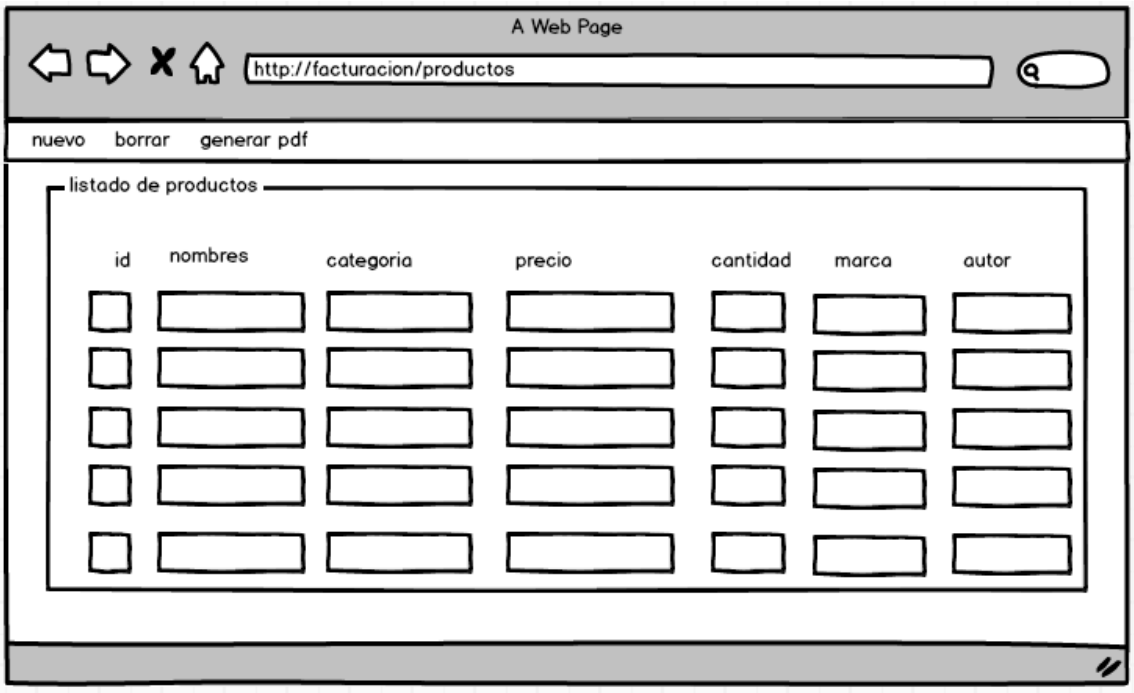

Figura N.º 4.7. CU04. Mantener Producto.

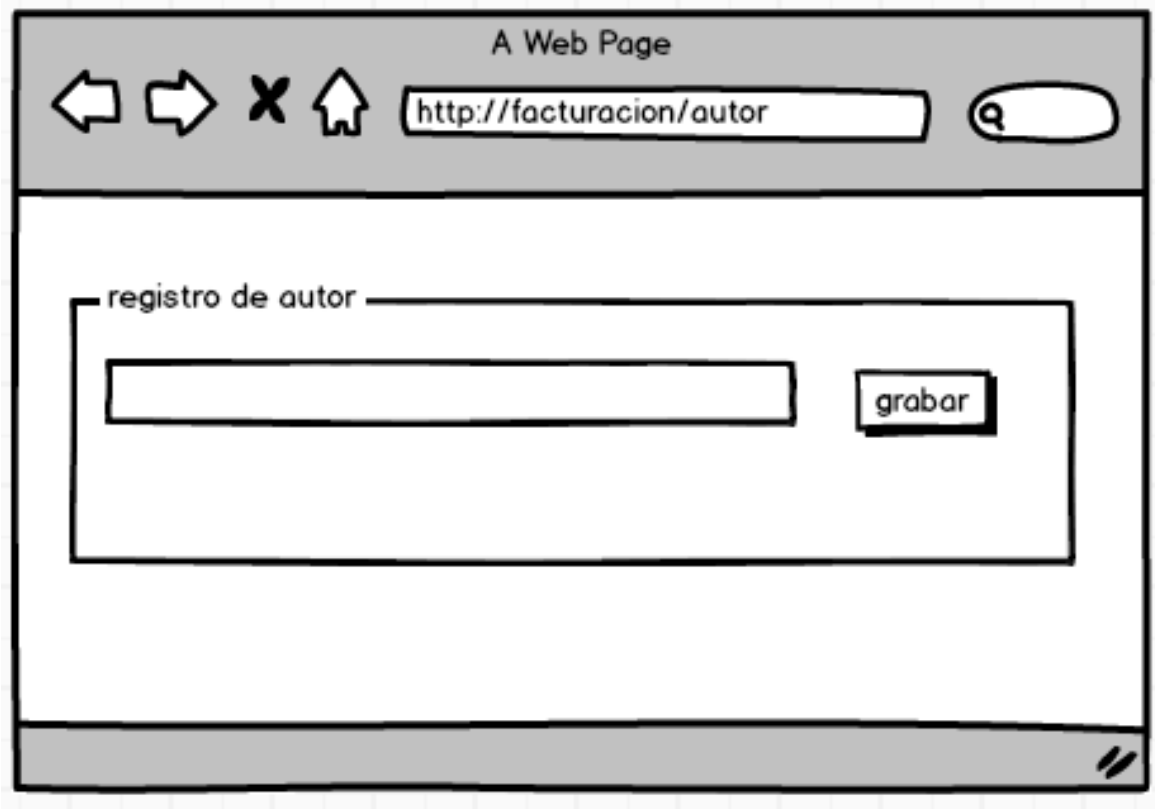

Figura N.º 4.8. CU05. Mantener Autores.

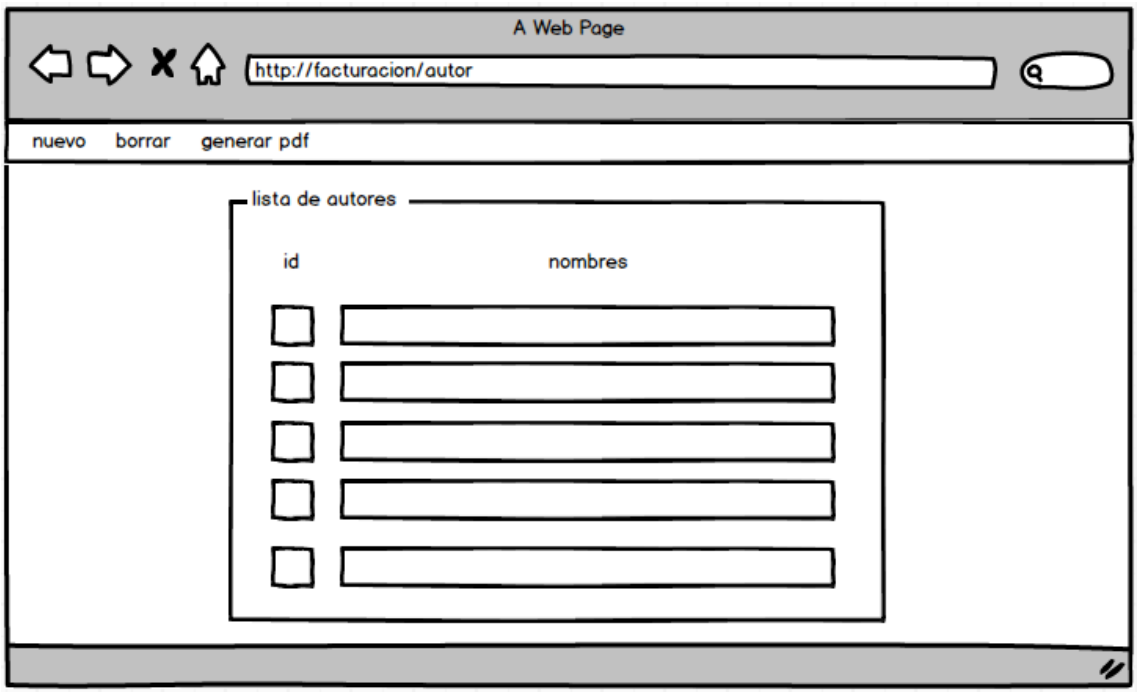

Figura N.º 4.9. CU05. Mantener autores.

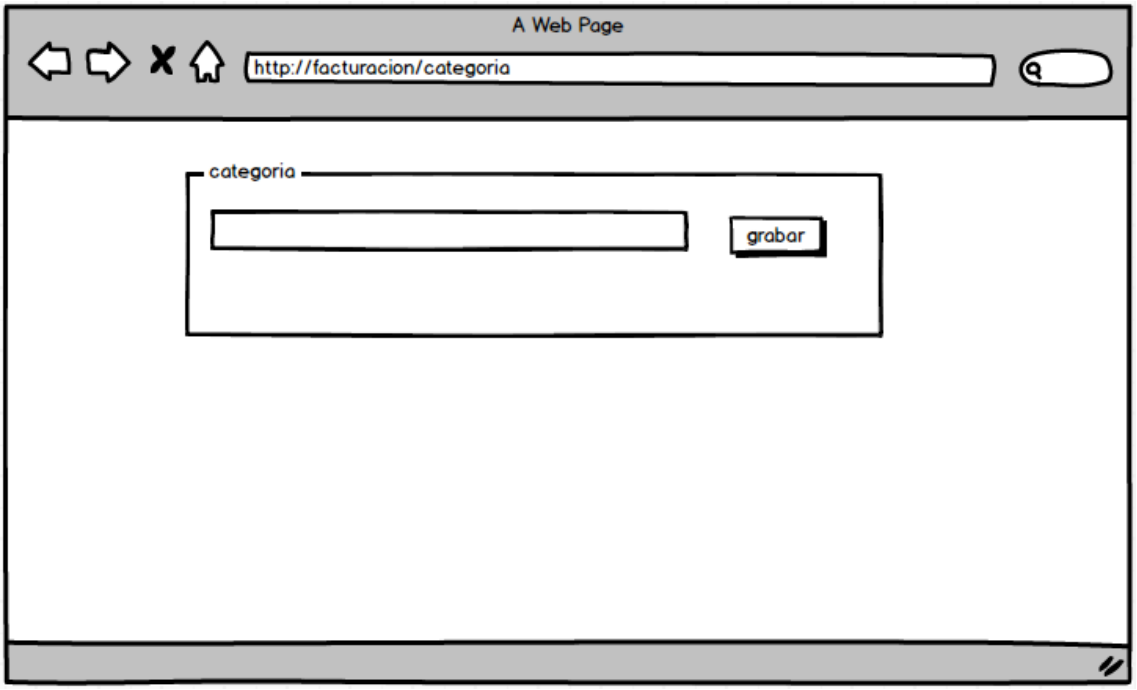

Figura N.º 4.10. CU06. Mantener Categoría.

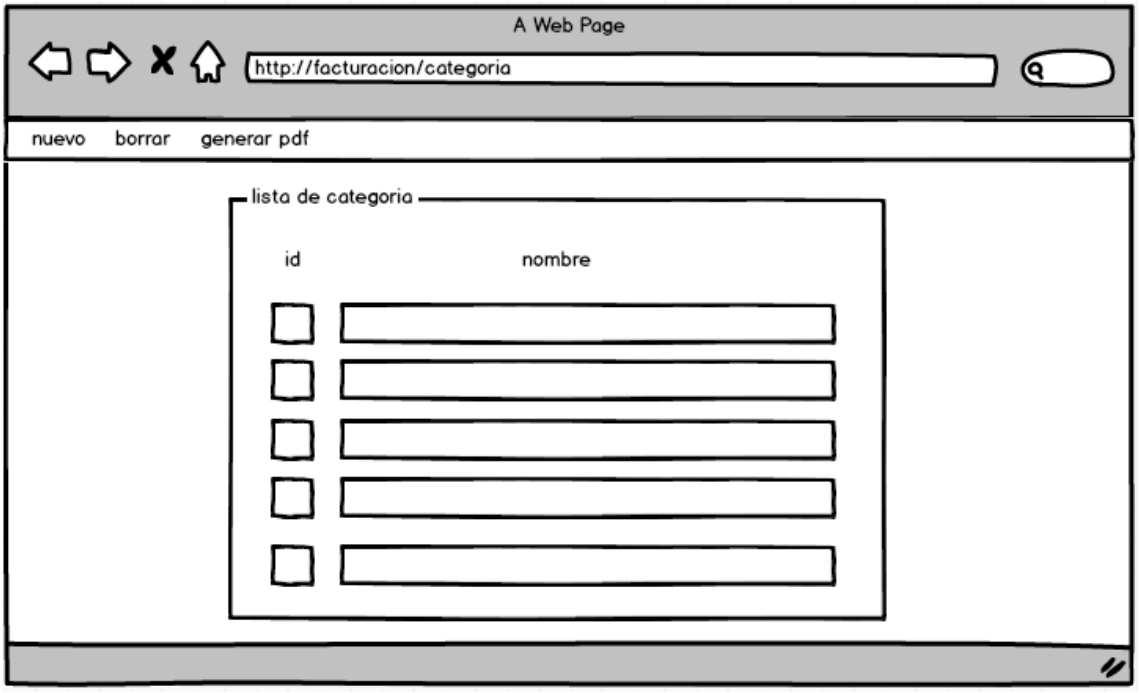

Figura N.º 4.11. CU06. Mantener Categoría.

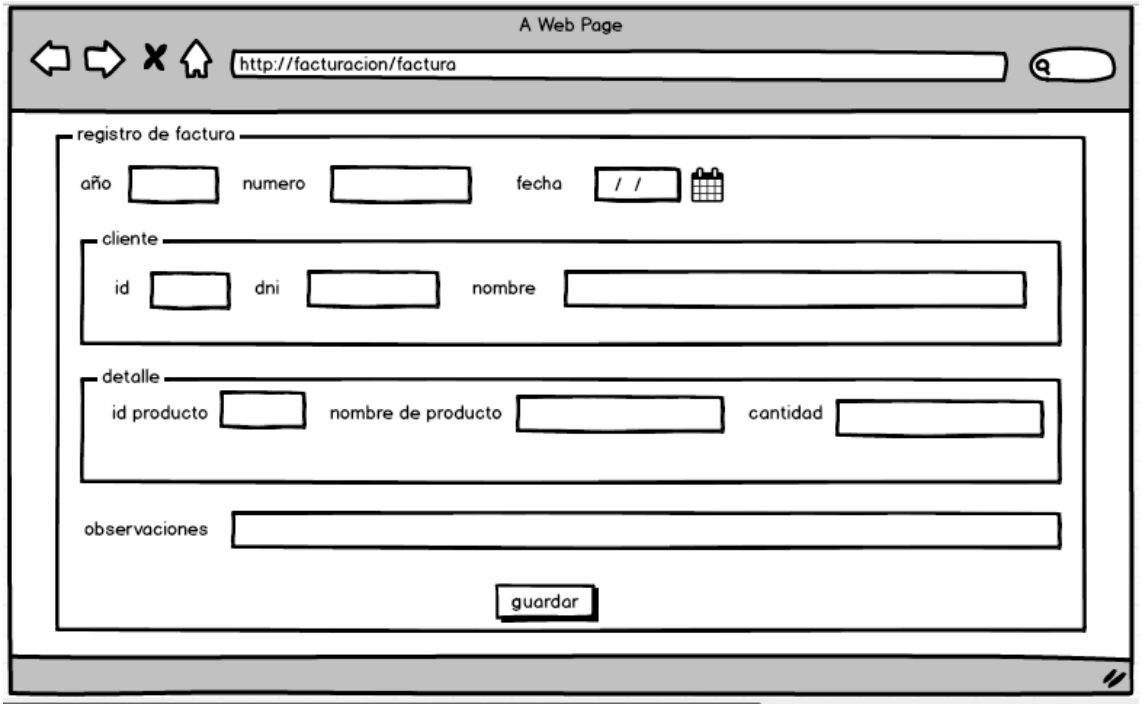

Figura N.º 4.12. CU07 Realizar venta.

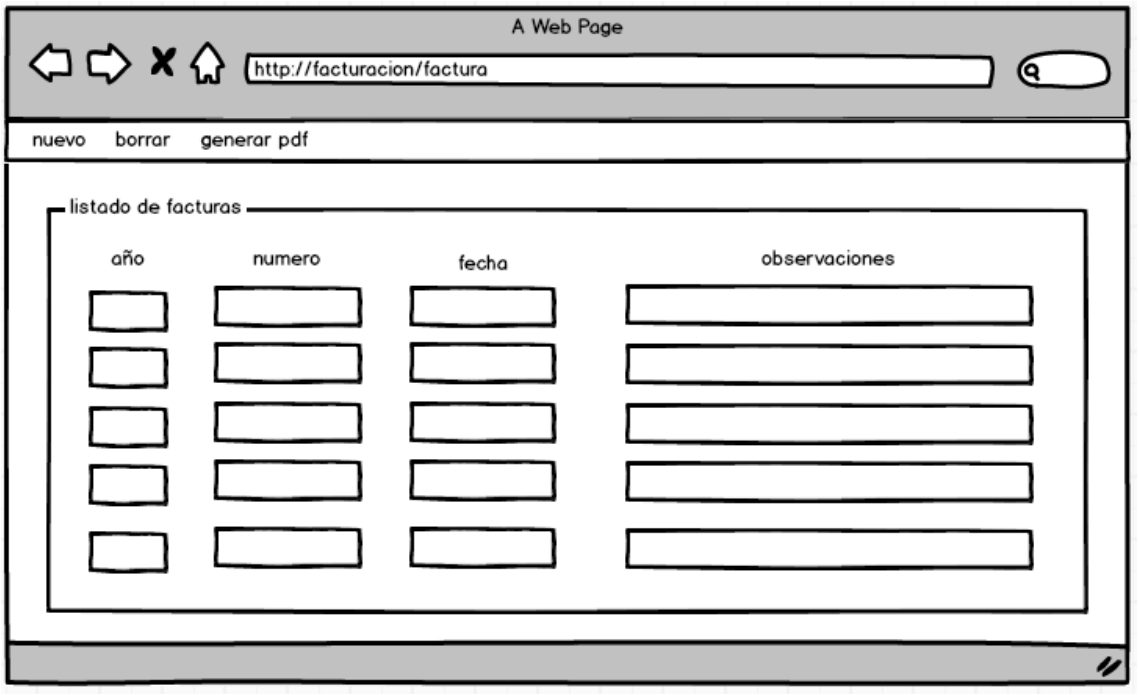

Figura N.º 4.13. CU07 Realizar venta.

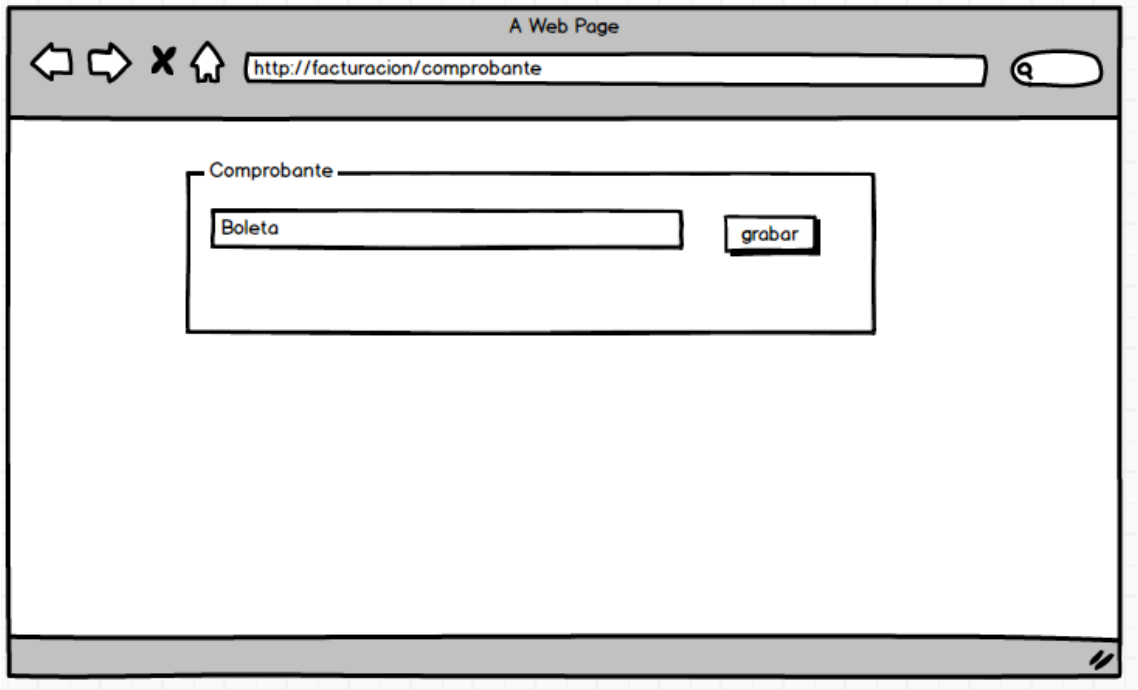

Figura N.º 4.14. CU08 Mantener Comprobante.

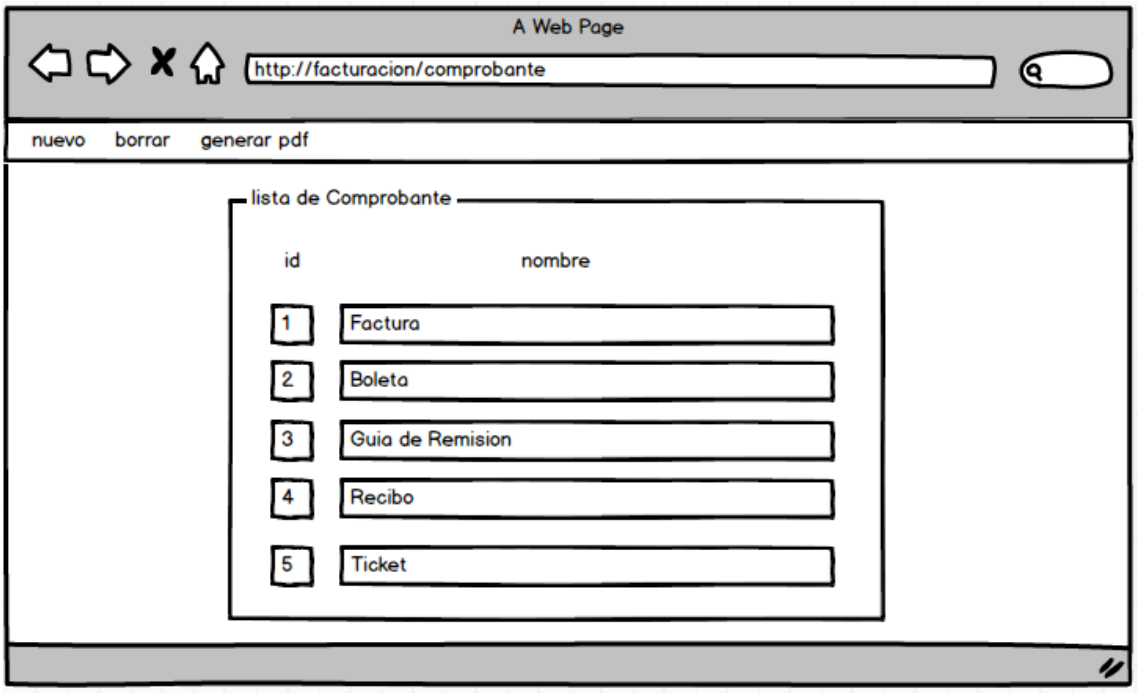

Figura N.º 4.15. CU08 Mantener Comprobante.

4.1.8. PRIMER BORRADOR DE CASOS DE USO

C U	DESCRIPCIÓN
CU01: autenticar usuario	**Curso básico:** el **usuario** del sistema se encuentra en la parte inicio del sistema, el sistema le muestra la pantalla de Loguin. El **usuario** del sistema, para iniciar sesión en la aplicación, introduce el usuario en el campo usuario, además de introducir la contraseña en el campo password. El usuario del sistema hace click en entrar al sistema, si los datos son correctos este ingresa al menú de facturación, **Curso alterno:** en el caso de que no se compruebe los datos del **usuario** de sistema, el sistema arrojara un mensaje de tipo alerta "usuario no autorizado". El **usuario** hace click en cerrar ventada y se cierra el módulo.
CU02: Mantener Clientes.	**Curso básico:** El usuario del sistema se encuentra en el menú del sistema, hace clic en el ítem de cliente, el sistema le muestra la pantalla de "registro de clientes". El **usuario** del sistema, para registrar un nuevo cliente, hace clic en el la pestaña nueva, el sistema abre un campo con la lista de datos a registrar como: id, DNI, nombre, apellido paterno, apellido materno, teléfono, correo, dirección a través de un abigeo., hace clic en el botón guardar. El **usuario** del sistema hace clic en el ítem lista para buscar, modificar los datos, el sistema le muestra una lista de los clientes al cual se le puede consultar por los campos de texto como DNI, Nombres, apellidos paterno y materno, teléfono, correo., luego el **usuario** del sistema selecciona al cliente en cuestión y el sistema le vuelve a mostrar la pantalla de registro de cliente, con los datos ya cargados del cliente. El usuario del sistema modifica los datos y hace clic en guardar. El **usuario** de sistema hace clic en lista de clientes para buscar al cliente a eliminar, el sistema le muestra una lista de clientes donde

	selecciona el cliente adecuado u activa el botón de eliminar, el sistema elimina al cliente. **Curso alterno:** El **usuario** del sistema cancela sus operaciones cerrando el módulo juntamente con el navegador El **usuario** hace clic en cerrar ventada y se cierra el módulo. El **usuario** del sistema ingresa datos no válidos, el sistema le muestra un mensaje de validación de campos.
CU03: Generar Reporte	**Curso básico:** El **usuario** del sistema se encuentra en el menú principal del sistema de facturación. Hace clic en los módulos de, autor, categoría, cliente, productos, de los que desee. El **usuario** del sistema para generar un reporte de los módulos que desee, hace clic en lista de tarea superior con el título generar pdf, el sistema le genera una nueva ventana con el pdf listo para su impresión. **Curso alterno:** El usuario del sistema cuando genera una lista de datos inexistentes el sistema le muestra un mensaje de no se puede generar reporte de datos vacíos.
CU04: mantener productos	**Curso básico:** El usuario del sistema se encuentra en el menú del sistema, hace clic en el ítem de producto, el sistema le muestra la pantalla de "registro de producto". El **usuario** del sistema, para registrar un nuevo producto, hace clic en el la pestaña nueva, el sistema abre un campo con la lista de datos a registrar como: id, nombre, categoría, precio, cantidad, marca, foto, autor, observaciones, hace clic en el botón guardar El **usuario** del sistema hace clic en el ítem lista para buscar, modificar los datos, el sistema le muestra una lista de los productos al cual se le puede consultar por los campos de texto como nombre, categoría, precio, cantidad, marca., luego el **usuario** del sistema selecciona al producto en cuestión y el sistema le vuelve a mostrar la pantalla de

	registro de producto, con los datos ya cargados del producto. El usuario del sistema modifica los datos y hace clic en guardar. El **usuario** de sistema hace clic en lista de productos para buscar al producto a eliminar, el sistema le muestra una lista de productos donde selecciona el producto adecuado y u activa el botón de eliminar, el sistema elimina al producto. **Curso alterno:** El **usuario** del sistema cancela sus operaciones cerrando el módulo juntamente con el navegador El **usuario** hace clic en cerrar ventada y se cierra el módulo. El **usuario** del sistema ingresa datos no válidos, el sistema le muestra un mensaje de validación de campos.
CU05: mantener autores	**Curso básico:** El usuario del sistema se encuentra en el menú del sistema, hace clic en el ítem de autor, el sistema le muestra la pantalla de "registro de autor". El **usuario** del sistema, para registrar un nuevo autor, hace clic en el la pestaña nueva, el sistema abre un campo con el ítem de nombre de autor, hace clic en el botón guardar El **usuario** del sistema hace clic en el ítem lista para buscar, modificar los datos, el sistema le muestra una lista de los actores al cual se le puede consultar por los campos de texto como nombre, luego el **usuario** del sistema selecciona al autor en cuestión y el sistema le vuelve a mostrar la pantalla de registro de autor, con los datos ya cargados del autor. El usuario del sistema modifica los datos y hace clic en guardar. El **usuario** de sistema hace clic en lista de autores para buscar al autor a eliminar, el sistema le muestra una lista de autores donde selecciona el autor adecuado y activa el botón de eliminar, el sistema elimina al autor.

	Curso alterno: El **usuario** del sistema cancela sus operaciones cerrando el módulo juntamente con el navegador El **usuario** hace clic en cerrar ventada y se cierra el módulo. El **usuario** del sistema ingresa datos no válidos, el sistema le muestra un mensaje de validación de campos.
CU06: mantener categoría	**Curso básico:** El usuario del sistema se encuentra en el menú del sistema, hace clic en el ítem de categoría, el sistema le muestra la pantalla de "registro de categoría". El **usuario** del sistema, para registrar una nueva categoría, hace clic en el la pestaña nueva, el sistema abre un campo con el ítem de nombre de categoría, hace clic en el botón guardar El **usuario** del sistema hace clic en el ítem lista para buscar, modificar los datos, el sistema le muestra una lista de las categorías al cual se le puede consultar por los campos de texto como nombre de categoría, luego el **usuario** del sistema selecciona la categoría en cuestión y el sistema le vuelve a mostrar la pantalla de registro de categoría, con los datos ya cargados. El usuario del sistema modifica los datos y hace clic en guardar. El **usuario** de sistema hace clic en lista de categoría para buscar categoría a eliminar, el sistema le muestra una lista de categorías donde selecciona la categoría adecuado y activa el botón de eliminar, el sistema elimina la categoría. **Curso alterno:** El **usuario** del sistema cancela sus operaciones cerrando el módulo juntamente con el navegador El **usuario** hace clic en cerrar ventada y se cierra el módulo. El **usuario** del sistema ingresa datos no válidos, el sistema le muestra un mensaje de validación de campos.

CU07: realizar venta	**Curso básico:** El usuario del sistema se encuentra en el menú del sistema, hace clic en el ítem de ventas, el sistema le muestra la pantalla de "registro de venta". El **usuario** del sistema, para registrar una nueva venta, hace clic en el la pestaña nueva, el sistema abre un campo con los ítems de año, el número de factura, y la fecha, también el sistema proporciona la lista de los clientes a los cuales se le asignara la venta, este campo que corresponde a los clientes contiene el DNI, nombre. Además, el sistema le muestra una vista de detalles de los productos los cuales van a ser vendidos, esta vista contiene el nombre de los productos y la cantidad de ellos, al final de la vista el sistema ofrece un campo de observaciones de la venta, hace clic en el botón guardar. El **usuario** del sistema hace clic en el ítem lista para buscar, modificar los datos, el sistema le muestra una lista de las ventas al cual se le puede consultar por los campos de texto como año, número, fecha, observaciones luego el **usuario** del sistema selecciona la venta en cuestión y el sistema le vuelve a mostrar la pantalla de registro de venta, con los datos ya cargados. El usuario del sistema modifica los datos y hace clic en guardar. El **usuario** de sistema hace clic en lista de venta para buscar y eliminar, el sistema le muestra una lista de ventas donde selecciona la venta adecuado y activa el botón de eliminar, el sistema elimina la venta. **Curso alterno:** El **usuario** del sistema cancela sus operaciones cerrando el módulo juntamente con el navegador El **usuario** hace clic en cerrar ventada y se cierra el módulo. El **usuario** del sistema ingresa datos no válidos, el sistema le muestra un mensaje de validación de campos.

CU08: mantener comprobante	**Curso básico:** El usuario del sistema se encuentra en el menú del sistema, hace clic en el ítem de comprobante, el sistema le muestra la pantalla de "registro de comprobante". El **usuario** del sistema, para registrar un nuevo tipo de comprobante, hace clic en el la pestaña nueva, el sistema abre un campo con el ítem de nombre de comprobante, hace clic en el botón guardar El **usuario** del sistema hace clic en el ítem lista para buscar, modificar los datos, el sistema le muestra una lista de los comprobantes al cual se le puede consultar por los campos de texto como nombre de comprobante, luego el **usuario** del sistema selecciona el comprobante en cuestión y el sistema le vuelve a mostrar la pantalla de registro de comprobante, con los datos ya cargados. El usuario del sistema modifica los datos y hace clic en guardar. El **usuario** de sistema hace clic en lista de comprobante para buscar el tipo de comprobante a eliminar, el sistema le muestra una lista de comprobantes donde selecciona el adecuado y activa el botón de eliminar, el sistema elimina el comprobante. **Curso alterno:** El **usuario** del sistema cancela sus operaciones cerrando el módulo juntamente con el navegador El **usuario** hace clic en cerrar ventada y se cierra el módulo. El **usuario** del sistema ingresa datos no válidos, el sistema le muestra un mensaje de validación de campos.

Tabla N.º 4.6. Primer borrador de casos de uso.

4.2. REVISIÓN DE REQUISITOS

4.2.1. MODELADO DE DOMINIO REVISADO

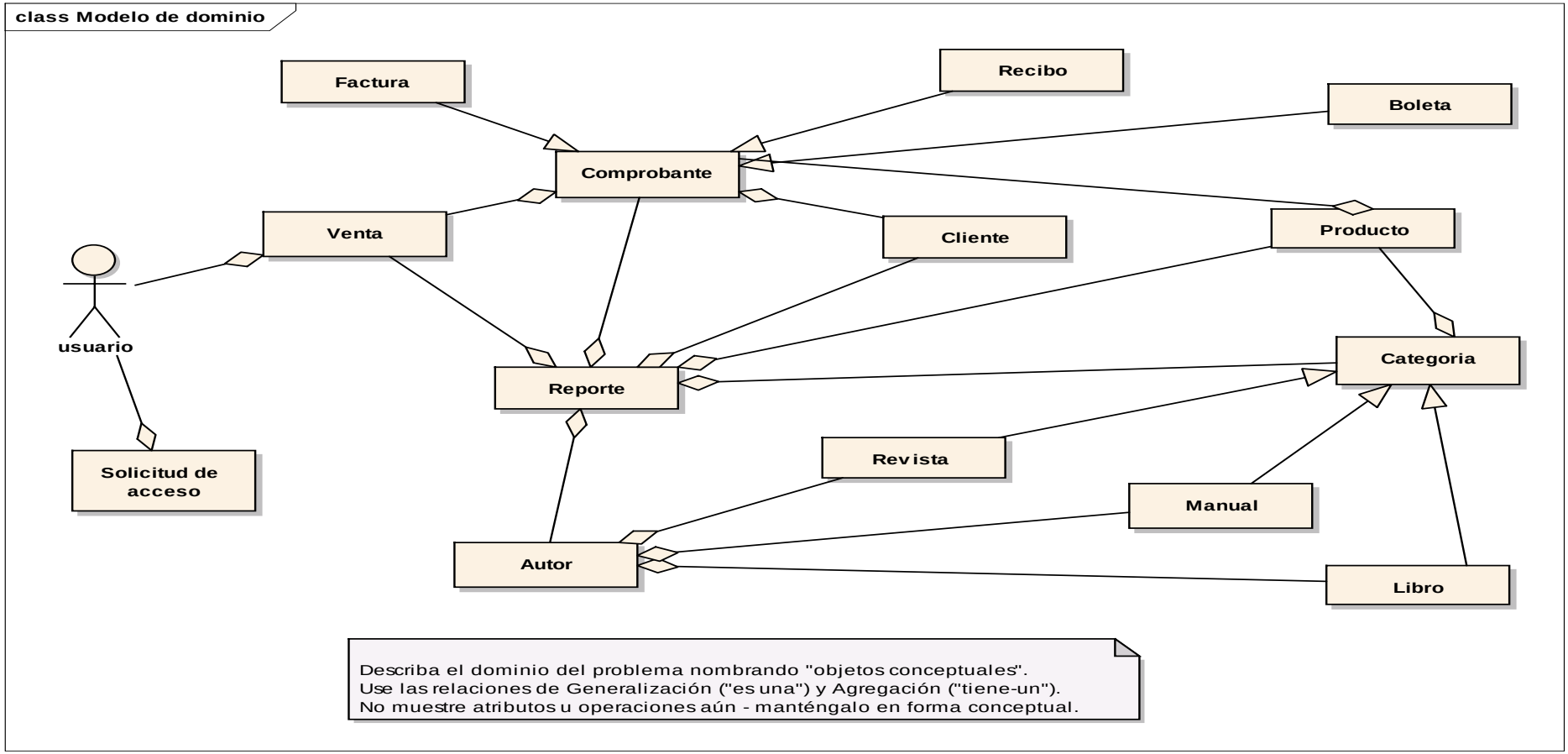

Figura N.º 4.16. Modelo de dominio revisado

4.2.2. PROTOTIPO DE GUI MEJORADO

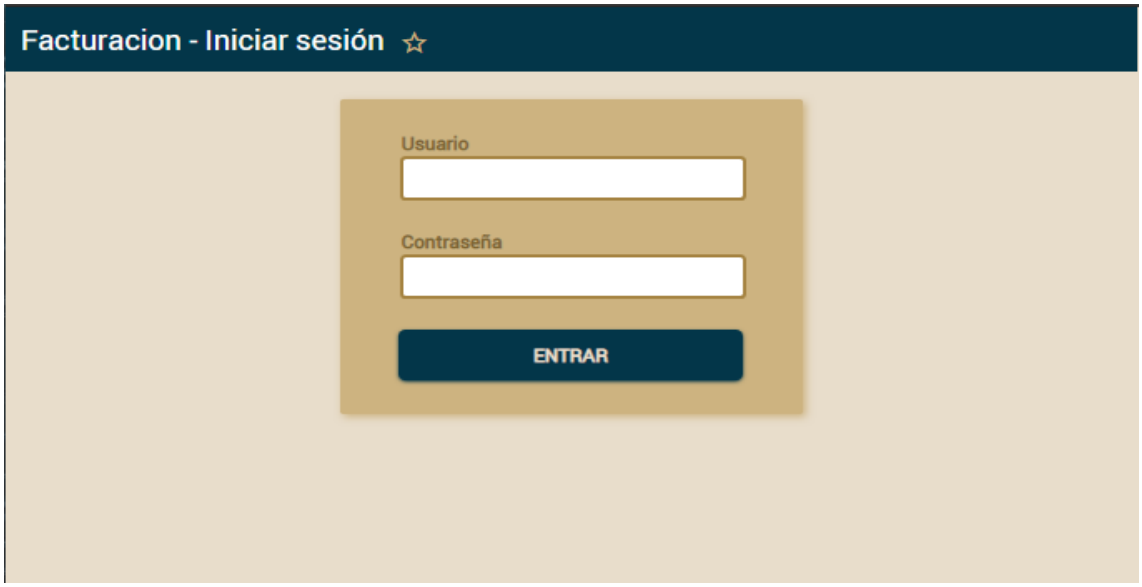

Figura N.º 4.17. CU01 Autenticar Usuario

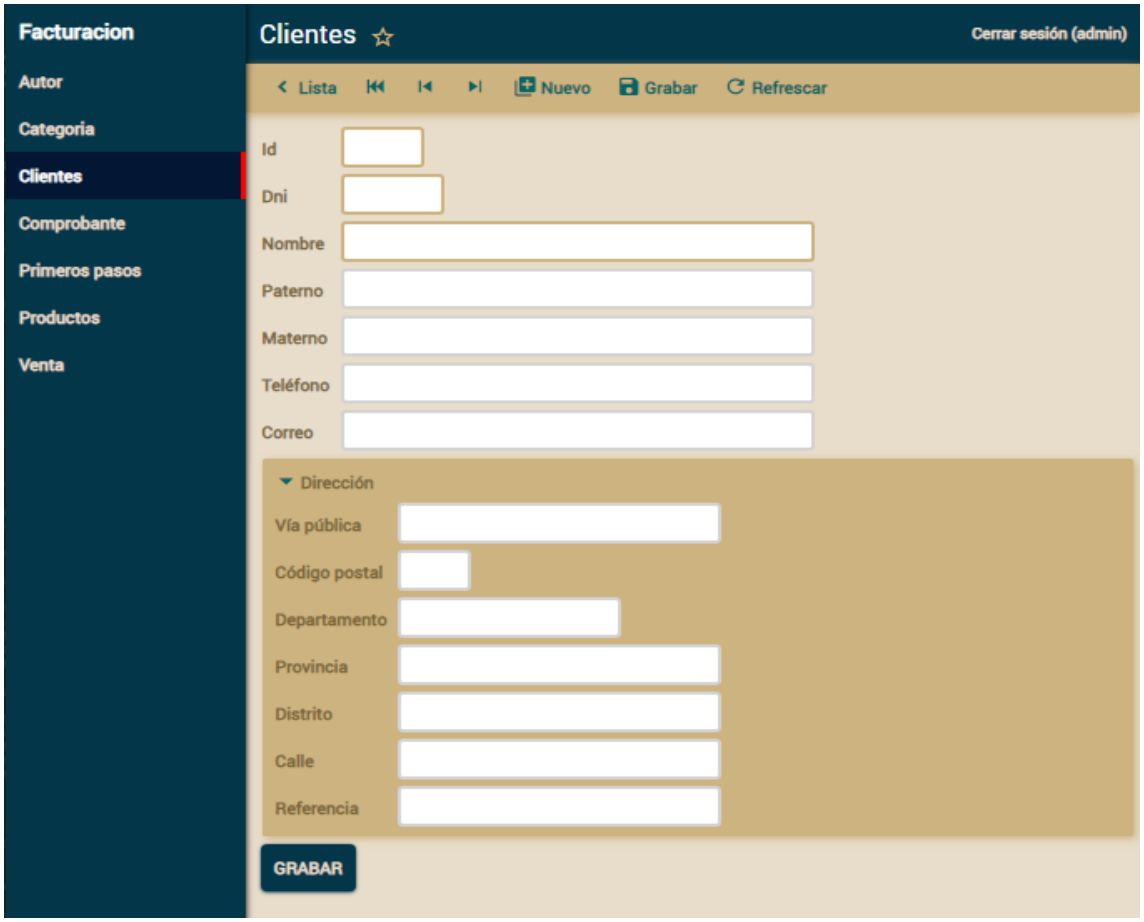

Figura N.º 4.18. CU02 Mantener Clientes

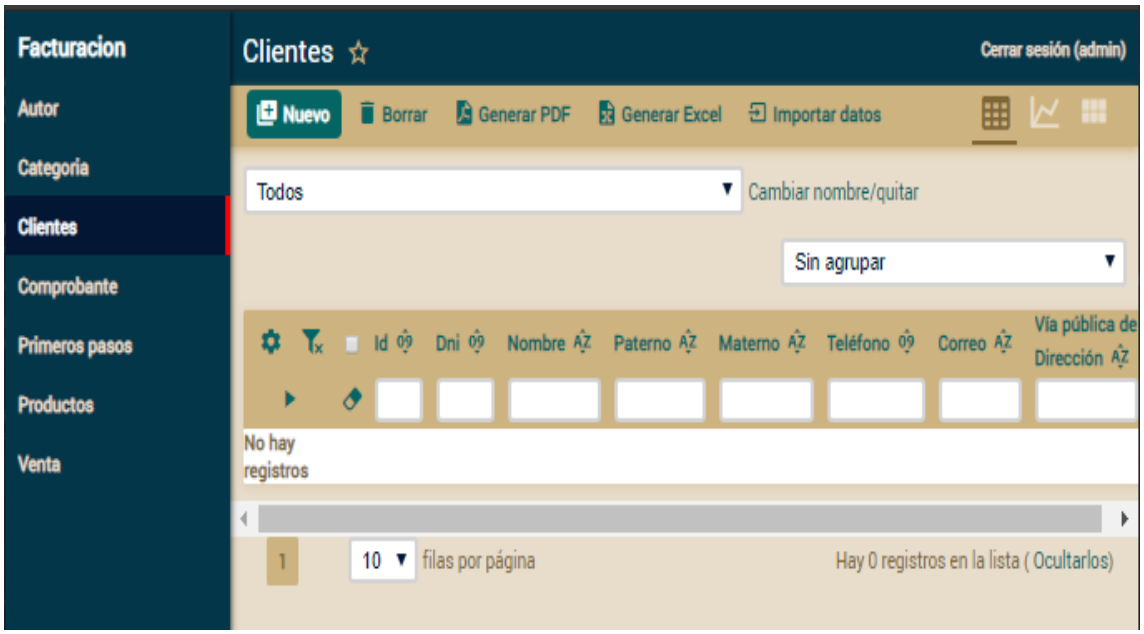

Figura N.º 4.19. CU02 Mantener Clientes.

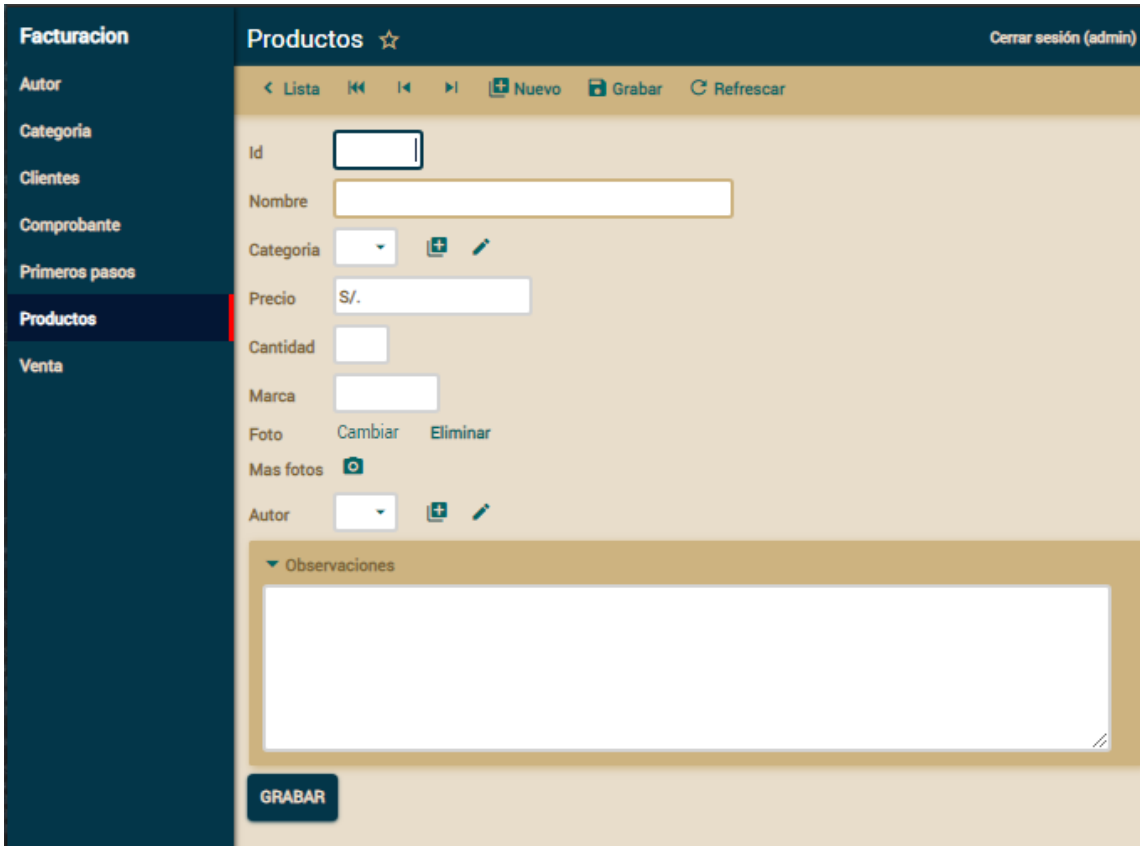

Figura N.º 4.20. CU04. Mantener Producto

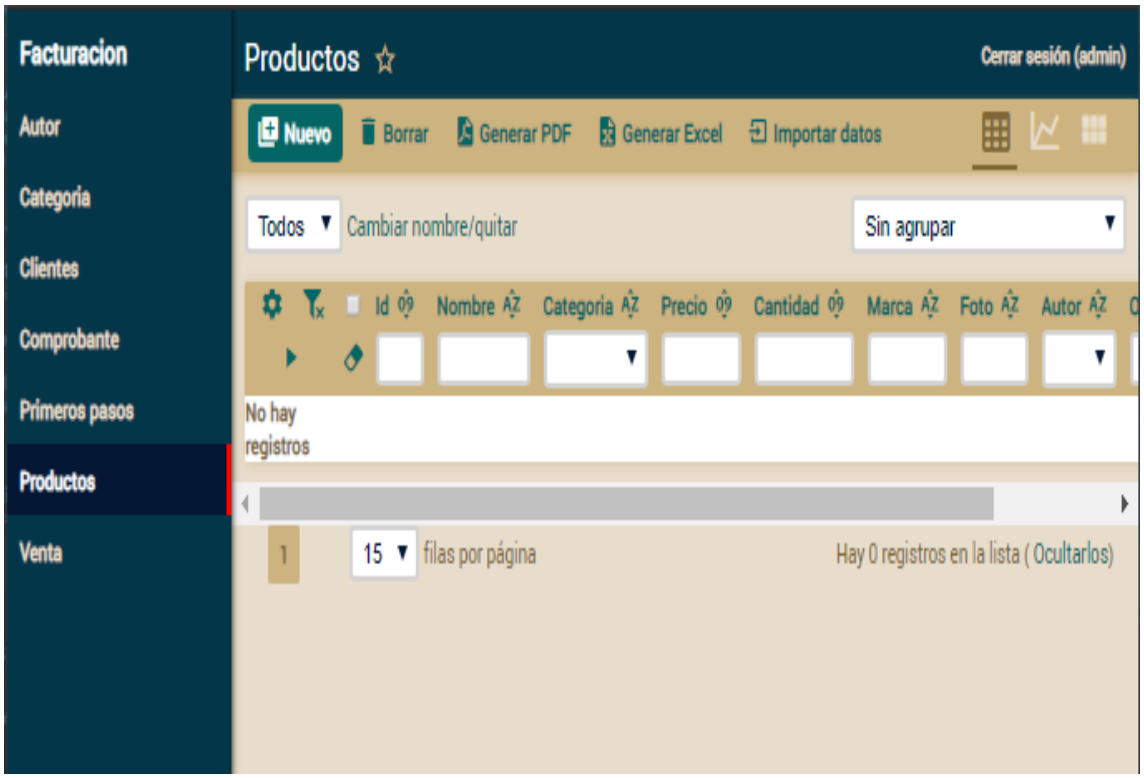

Figura N.º 4.21. CU04. Mantener Producto.

Figura N.º 4.22. CU05. Mantener Autores.

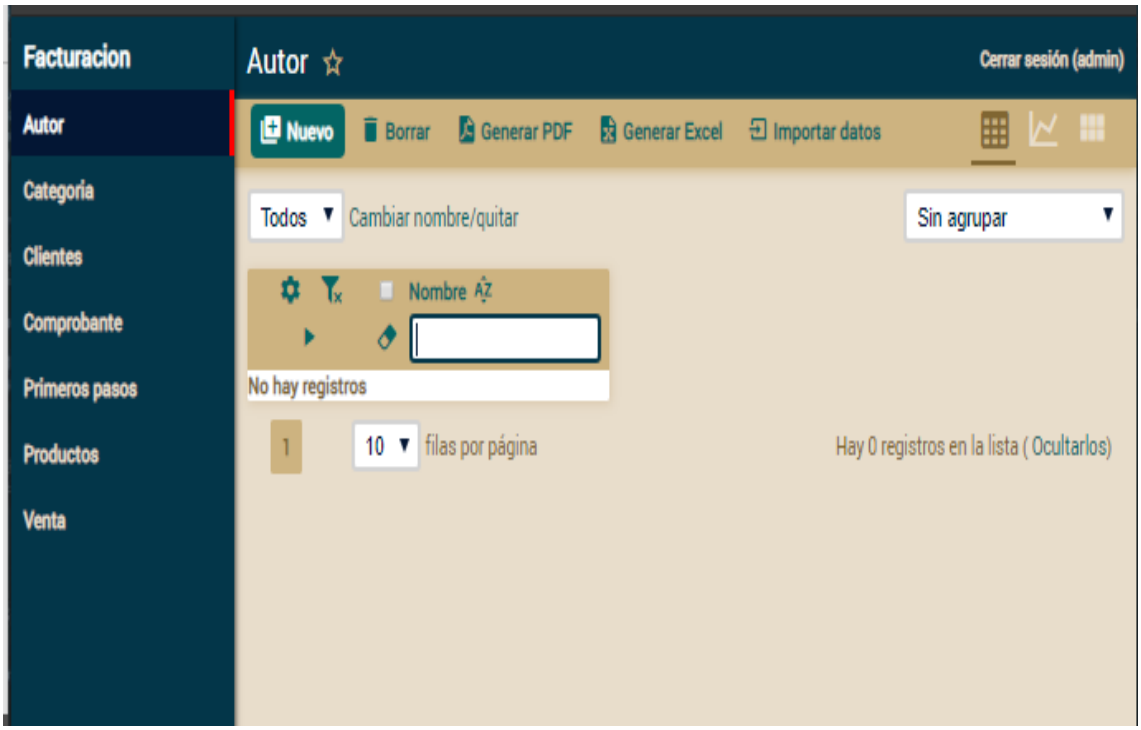

Figura N.º 4.23. CU05. Mantener autores.

Figura N.º 4.24. CU06. Mantener Categoría.

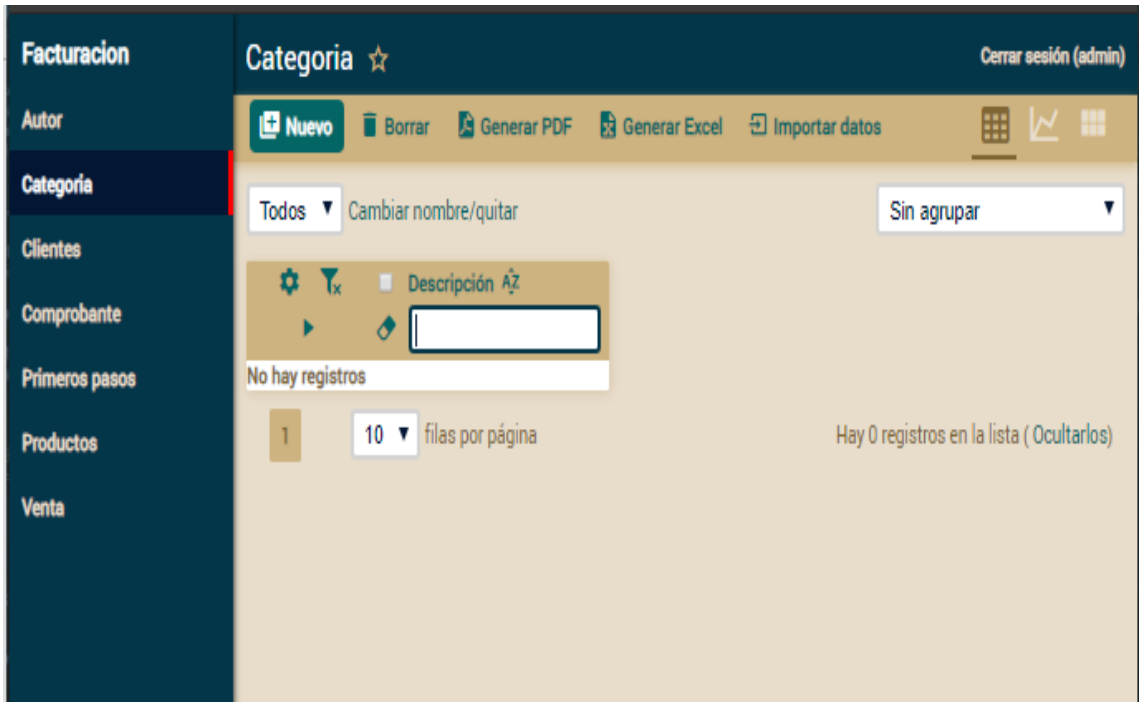

Figura N.º 4.25. CU06. Mantener Categoría.

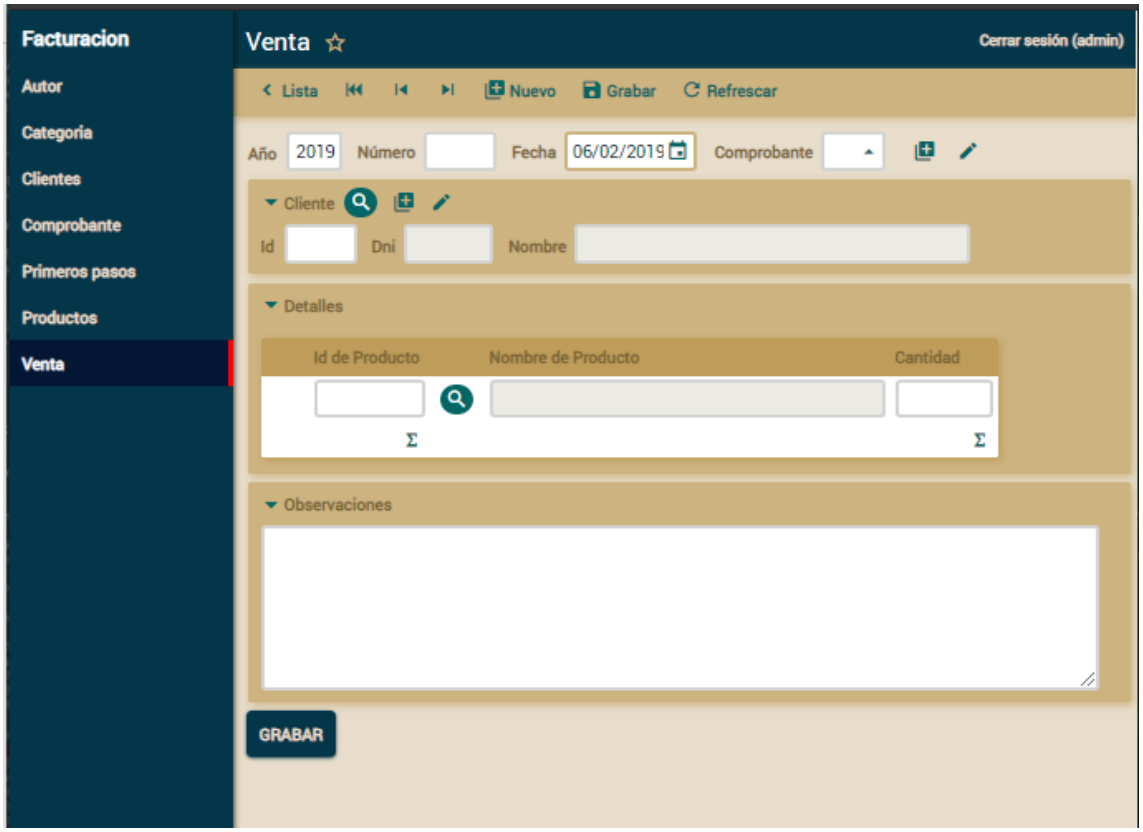

Figura N.º 4.26. CU07 Realizar venta.

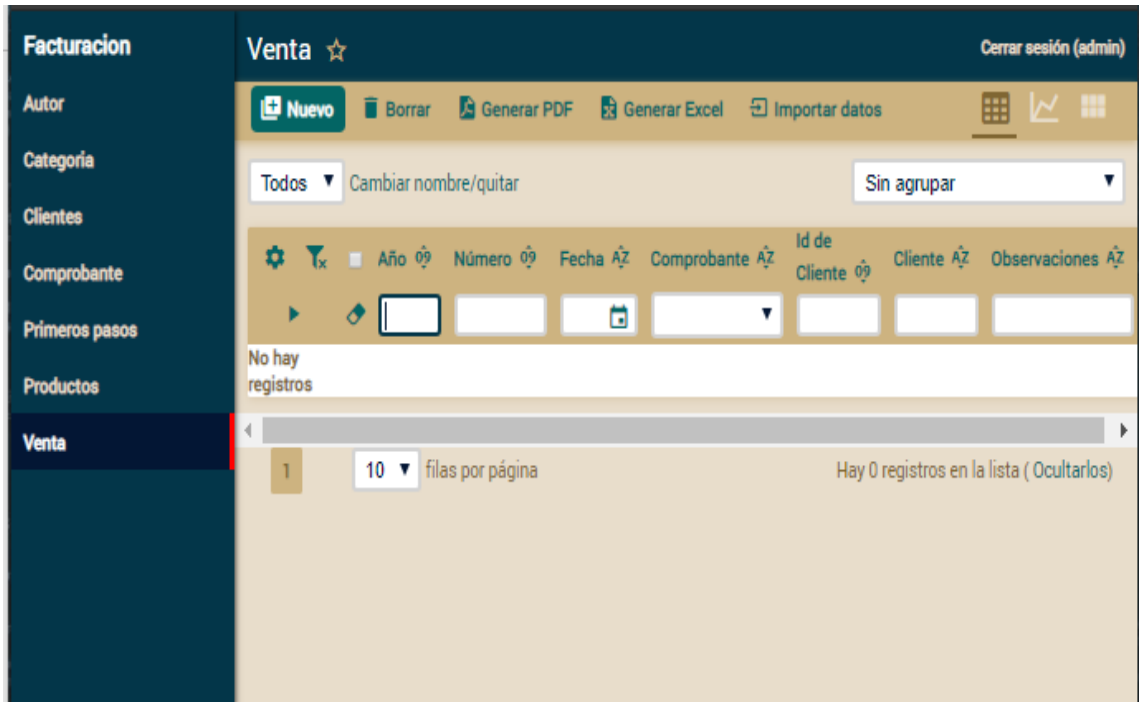

Figura N.º 4.27. CU07 Realizar venta.

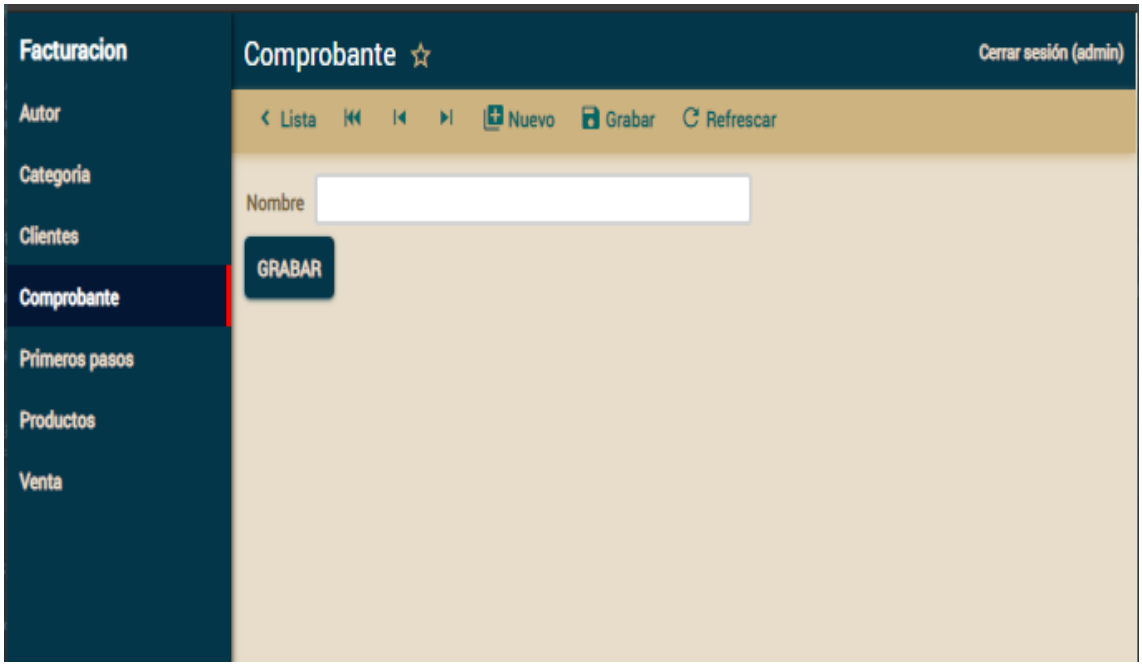

Figura N.º 4.28. CU08 Mantener Comprobante.

Figura N.º 4.29. CU08 Mantener Comprobante.

4.2.3. CASOS DE USO REVISADO

C U	DESCRIPCIÓN
CU01: autenticar usuario	**Curso básico:** **1.** El **usuario** del sistema se encuentra en la parte inicio del sistema, el sistema le muestra la pantalla de inicio de sesión. **2.** El **usuario** del sistema, para iniciar sesión en la aplicación, introduce el usuario en el campo usuario, además de introducir la contraseña en el campo Contraseña. **3**. El usuario del sistema hace click en entrar al sistema, si los datos son correctos este ingresa al menú de facturación, **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos.**" si alguno de los campos está vacio. **2.** El sistema muestra el mensaje de error "usuario no autorizado", si el usuario no es válido.

CU02: Mantener Clientes.	**Curso básico:** 1. El usuario del sistema hace clic en el menú principal **"Clientes"**, el sistema le muestra los submenús: **"Nuevo"**, **"Listar".** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Registro de nuevo Cliente".** 3. El usuario del sistema completa todos los campos de datos a rellenar, el sistema le muestra el formulario **"Registrar recurso"** dentro del cual se encuentra el campo, nombre, apellidos, dirección, correo para ingresar el nombre del recurso y los botones "**Guardar**" y "**Cancelar**". 4. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar recurso"** y el sistema muestra la página **"Lista de Clientes"** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se conecta a la base de datos.
CU04: mantener productos	**Curso básico:** 1. El usuario del sistema hace clic en el menú principal **"Productos"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Producto",** el sistema le muestra el formulario **"producto"** dentro del cual se encuentra los campos **precio, nombre, cantidad, marca, autor, foto** para para ingresar las observaciones de la actividad, **"observación"** para ingresar la descripción de la actividad; y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".**

	5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar producto"** y el sistema muestra la página **"Lista" .** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.
CU05: mantener autores	**Curso básico:** 1. El usuario del sistema hace clic en el menú principal **"Autores"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Autor",** el sistema le muestra el formulario **"Autor"** dentro del cual se encuentra los campos **nombre** para para ingresar los datos del autor y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar Autor"** y el sistema muestra la página **"Lista"** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.
	Curso básico: 1. El usuario del sistema hace clic en el menú principal **"Categoría"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Categoría",** el sistema le muestra el formulario **"Categoría"** dentro del cual se encuentra los campos **Categoría** para para ingresar los datos categoría y los botones "**Guardar**".

CU06: mantener categoría	3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar Categoría"** y el sistema muestra la página **"Lista"** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.
CU07: realizar venta	**Curso básico:** 1. El usuario del sistema hace clic en el menú principal **"Venta"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Venta",** el sistema le muestra el formulario **"Venta"** dentro del cual se encuentra los campos **Año, numero, fecha, comprobante,** para para ingresar los datos de la venta, además de **menú cliente, DNI, Nombre, y menú detalle, Nombre Producto, Cantidad producto** y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar Venta"** y el sistema muestra la página **"Lista".** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.
	Curso básico: 1. El usuario del sistema hace clic en el menú principal **"Comprobante"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"**

CU08: mantener comprobante	2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Comprobante",** el sistema le muestra el formulario **"Comprobante"** dentro del cual se encuentra los campos **Comprobante** para para ingresar los datos y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar Comprobante"** y el sistema muestra la página **"Lista"** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.

Tabla N.º 4.7. Listado de casos de uso revisado.

4.3. DISEÑO PRELIMINAR

4.3.1. CASOS DE USO DESAMBIGUADO

CASO DE USO	DESCRIPCIÓN
	Curso básico: 1. El usuario del sistema hace clic en el menú principal **"CLIENTES"**, el sistema le muestra los submenús: **"NUEVO"**, **"LISTAR".** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Registro de nuevo Cliente".** 3. El usuario del sistema completa todos los campos de datos a rellenar, el sistema le muestra el formulario

CU02: Mantener Clientes.	**“Registrar recurso”** dentro del cual se encuentra el campo, nombre, apellidos, dirección, correo para ingresar el nombre del recurso y los botones “**Guardar**” y “**Cancelar**”. 4. El usuario del sistema hace clic en “**Guardar**” y el sistema le muestra el mensaje **“Registro correcto”.** 5. El usuario del sistema hace clic en “**Cancelar**” para cerrar el formulario **“Registrar Cliente”** y el sistema muestra la página **“Lista de Clientes”** **6.** El usuario del sistema hace clic en el menú “**Listar**”, el sistema le muestra el sub menú Lista, y los botones de Reporte. 7. el usuario de sistema hace clic en G**enerar Reporte,** el sistema he muestra una nueva ventana de reporte generado. 8. El usuario del sistema hace clic en **cerrar vista** parar cerrar vista **Reporte** **Curso alterno:** 1. El sistema muestra el mensaje de error “**Completar los campos.”** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **“Error en el registro.”** si no se conecta a la base de datos
CU04: mantener productos	**Curso básico:** 1. El usuario del sistema hace clic en el menú principal **“Productos”**, el sistema le muestra los submenús: **“Nuevo”**, **“Lista”** 2. El usuario del sistema hace clic en el submenú **“Nuevo”** y el sistema le muestra la página **“Nuevo Producto”,** el sistema le muestra el formulario **“producto”** dentro del cual se encuentra los campos **precio, nombre, cantidad, marca, autor, foto** para para ingresar las observaciones de la actividad, **“observación”**

	para ingresar la descripción de la actividad; y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar producto"** y el sistema muestra la página **"Lista"** **6.** El usuario del sistema hace clic en el menú "**Listar**", el sistema le muestra el sub menú Lista, y los botones de Reporte. 7. el usuario de sistema hace clic en G**enerar Reporte,** el sistema he muestra una nueva ventana de reporte generado. 8. El usuario del sistema hace clic en **cerrar vista** parar cerrar vista **Reporte** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.
	Curso básico: 1. El usuario del sistema hace clic en el menú principal **"Categoría"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Categoría",** el sistema le muestra el formulario **"Categoría"** dentro del cual se encuentra los campos **Categoría** para para ingresar los datos categoría y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".**

CU06: mantener categoría	5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar Categoría"** y el sistema muestra la página **"Lista".** **6.** El usuario del sistema hace clic en el menú "**Listar**", el sistema le muestra el sub menú Lista, y los botones de Reporte. 7. el usuario de sistema hace clic en G**enerar Reporte,** el sistema he muestra una nueva ventana de reporte generado. 8. El usuario del sistema hace clic en **cerrar vista** parar cerrar vista **Reporte.** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.
CU07: realizar venta	**Curso básico:** 1. El usuario del sistema hace clic en el menú principal **"Venta"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Venta",** el sistema le muestra el formulario **"Venta"** dentro del cual se encuentra los campos **Año, numero, fecha, comprobante,** para para ingresar los datos de la venta, además de **menú cliente, DNI, Nombre, y menú detalle, Nombre Producto, Cantidad producto** y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar Venta"** y el sistema muestra la página **"Lista".**

	6. El usuario del sistema hace clic en el menú "**Listar**", el sistema le muestra el sub menú Lista, y los botones de **Reporte**. 7. el usuario de sistema hace clic en G**enerar Reporte,** el sistema he muestra una nueva ventana de reporte generado. 8. El usuario del sistema hace clic en **cerrar vista** parar cerrar vista **Reporte.** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.

Tabla N.º 4.8. Listado de casos de uso desambiguado.

4.3.2. DIAGRAMA DE ROBUSTEZ

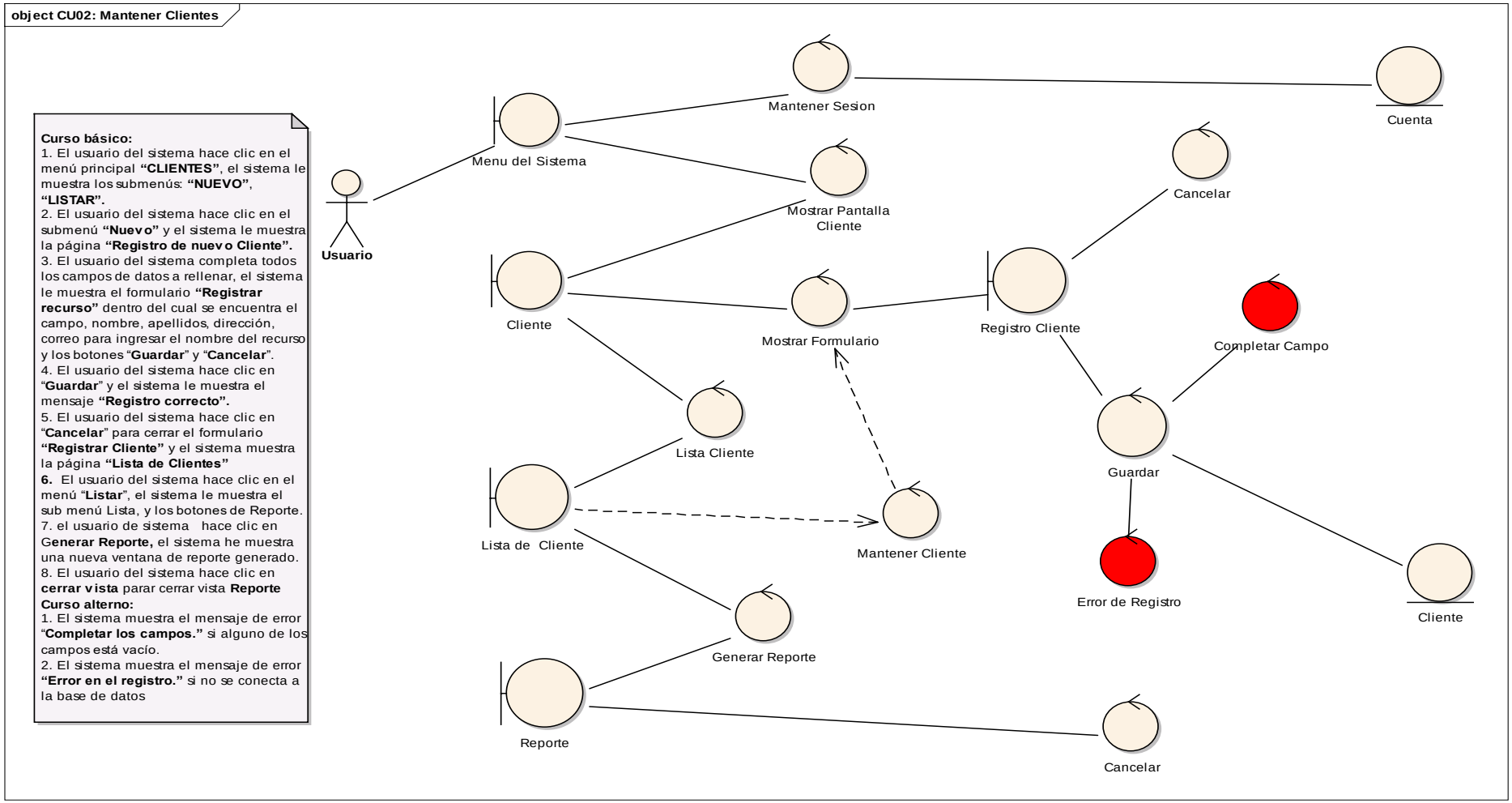

Figura N.º 4.30. CU02 Mantener Cliente.

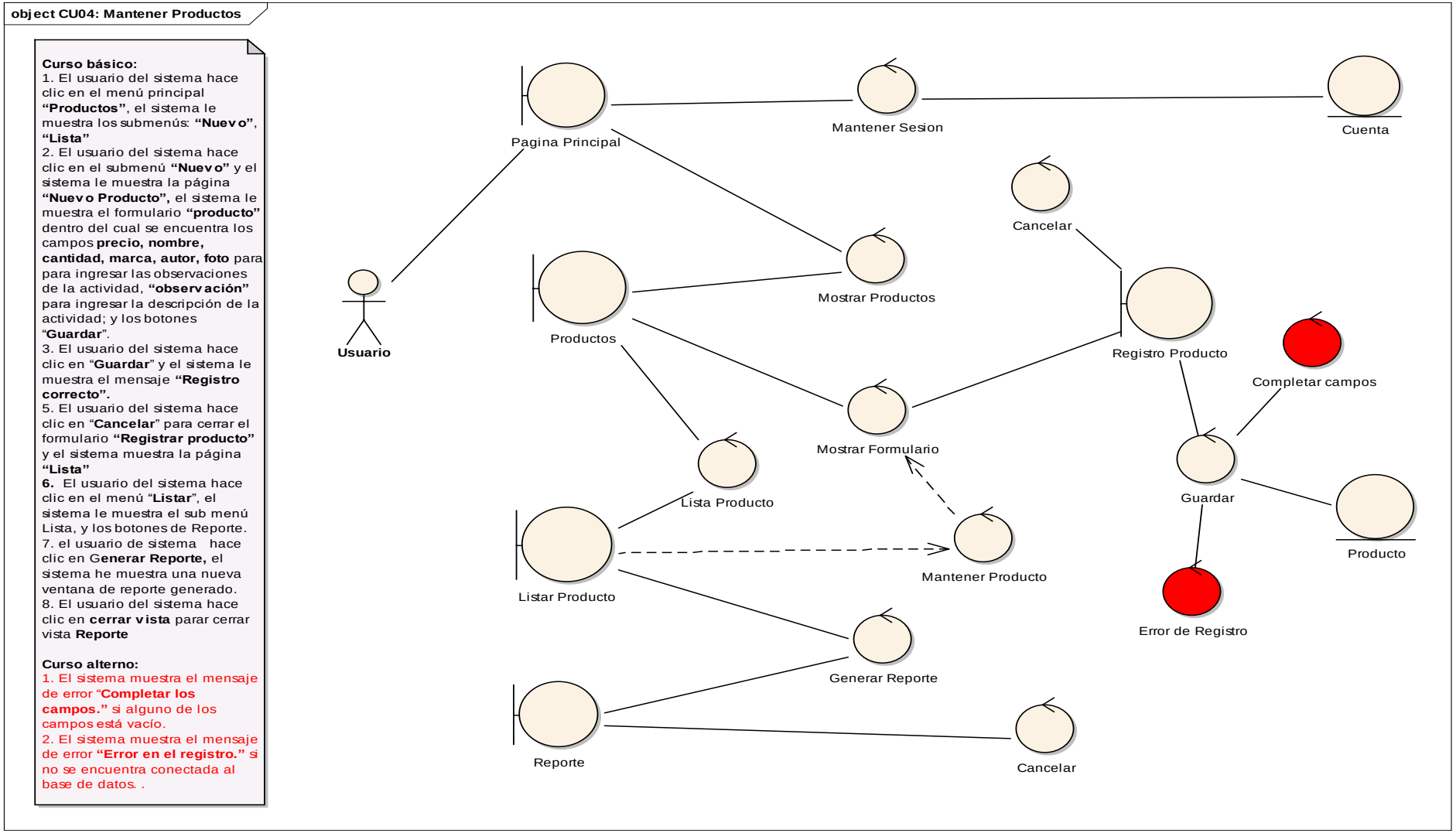

Figura N.º 4.31. CU04 Mantener Producto.

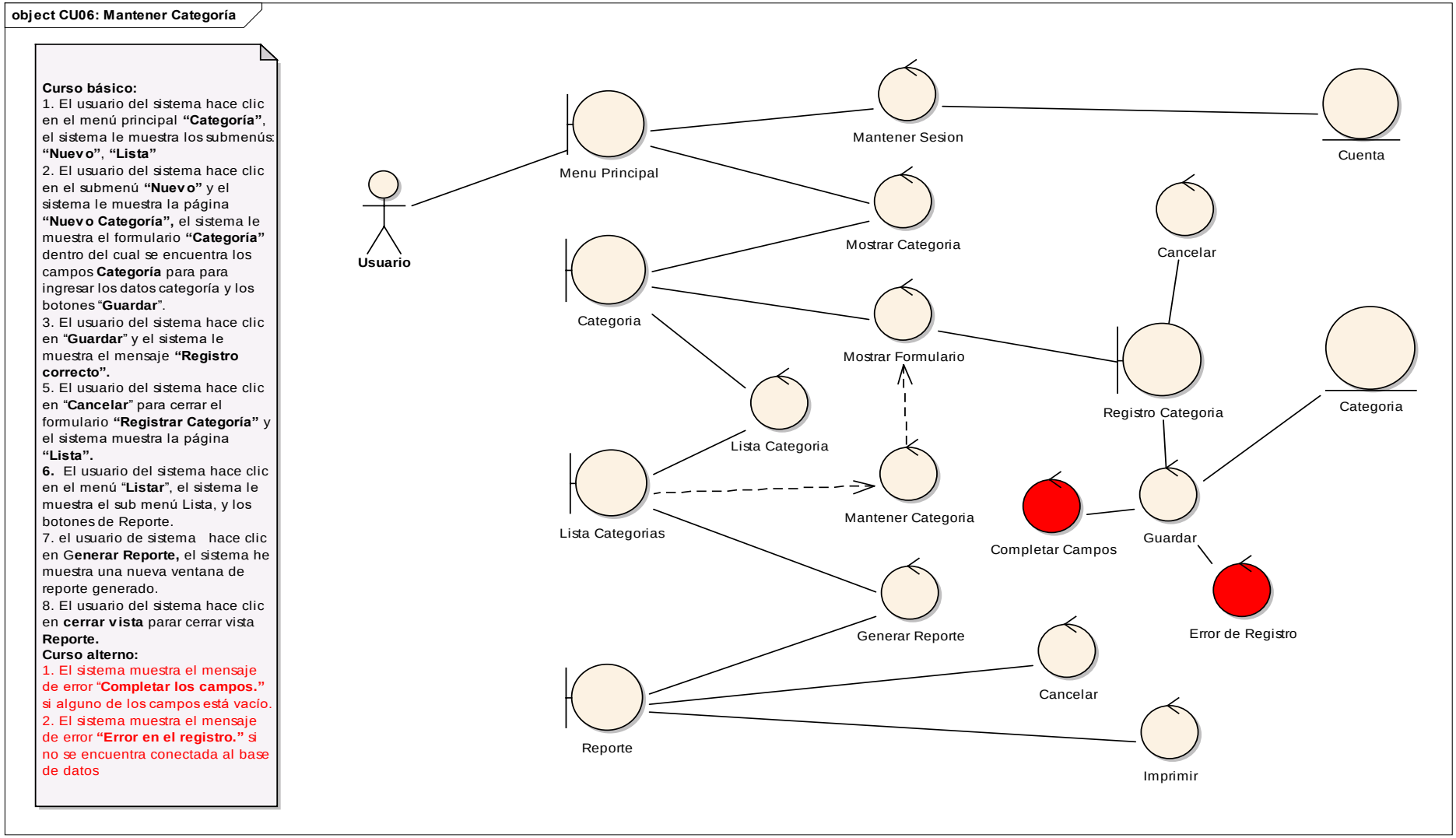

Figura N.º 4.32. CU06 Mantener Categoría.

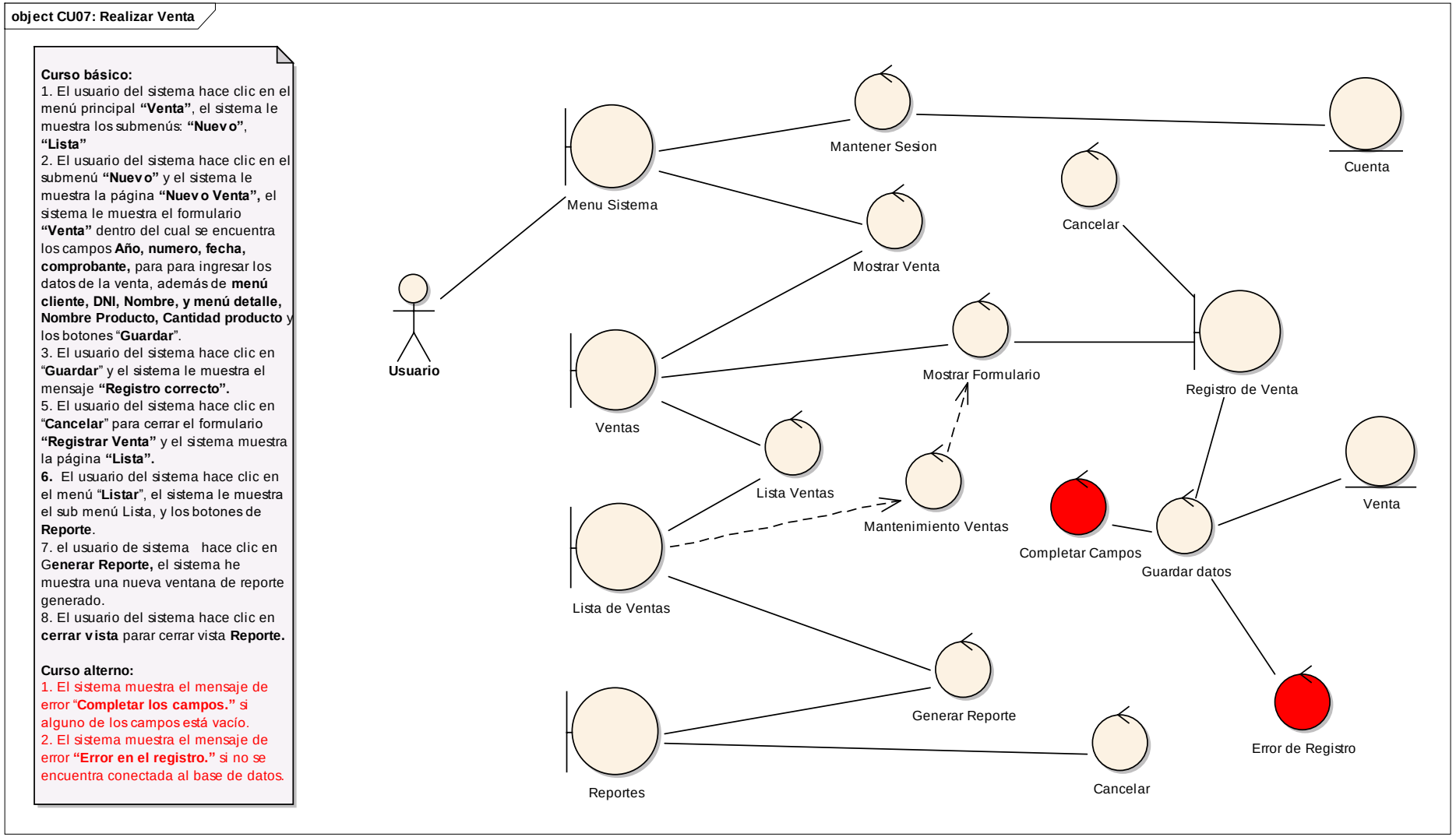

Figura N.º 4.33. CU07 Realizar Venta.

4.3.3. MODELO DE DOMINIO ACTUALIZADO

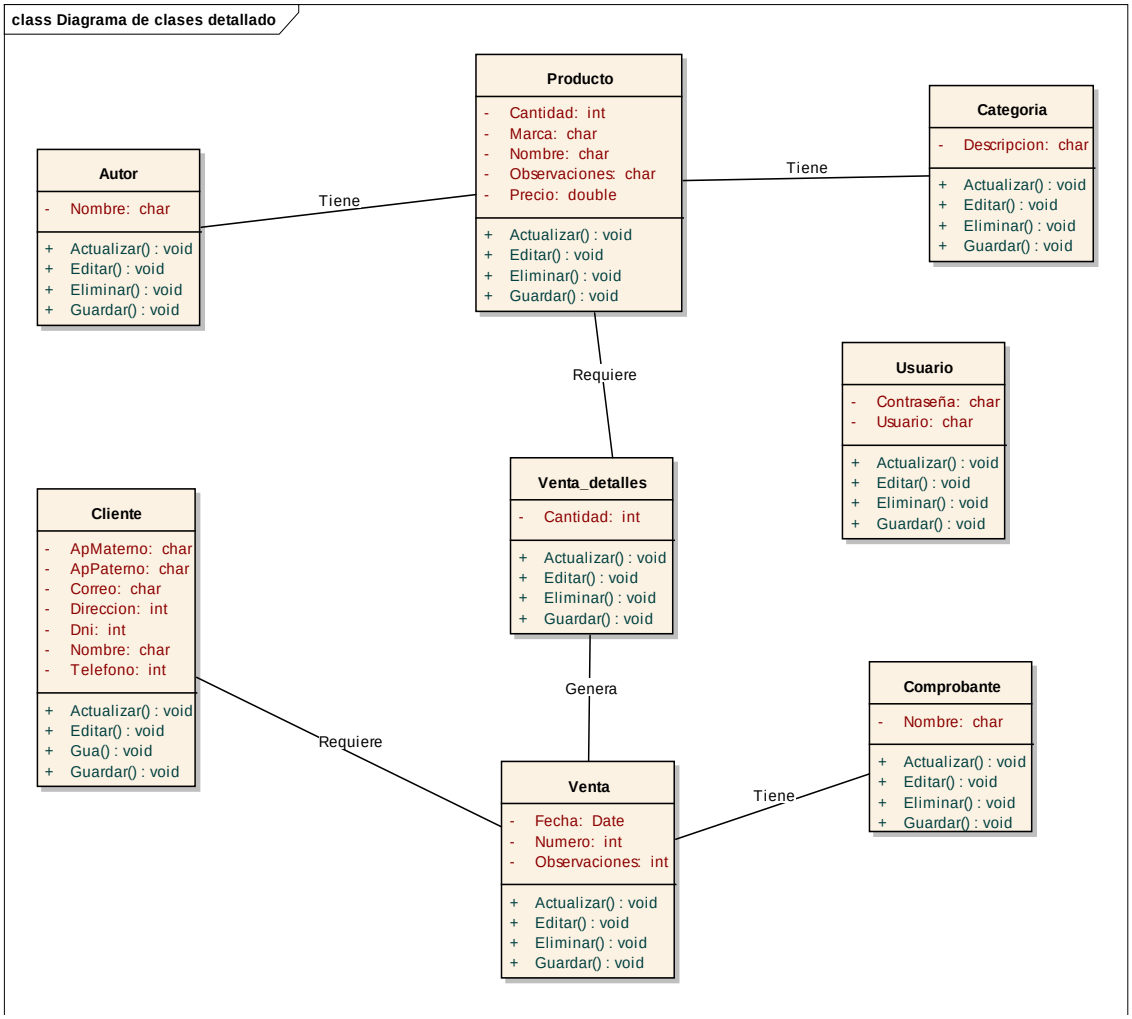

Figura N.º 4.34. Modelo de Dominio Actualizado

4.4. REVISION DE DISEÑO PRELIMINAR

4.4.1. REVISIÓN DE LA DESCRIPCIÓN DE LOS CASOS DE USO

CASO DE USO	DESCRIPCIÓN
	Curso básico: 1. El usuario del sistema hace clic en el menú principal **"CLIENTES"**, el sistema le muestra los submenús: **"NUEVO"**, **"LISTAR".** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Registro de nuevo Cliente".** 3. El usuario del sistema completa todos los campos de datos a rellenar, el sistema le muestra el formulario

CU02: Mantener Clientes.	**“Registrar recurso”** dentro del cual se encuentra el campo, nombre, apellidos, dirección, correo para ingresar el nombre del recurso y los botones “**Guardar**” y “**Cancelar**”. 4. El usuario del sistema hace clic en “**Guardar**” y el sistema le muestra el mensaje **“Registro correcto”.** 5. El usuario del sistema hace clic en “**Cancelar**” para cerrar el formulario **“Registrar Cliente”** y el sistema muestra la página **“Lista de Clientes”** **6.** El usuario del sistema hace clic en el menú “**Listar**”, el sistema le muestra el sub menú Lista, y los botones de Reporte. 7. el usuario de sistema hace clic en G**enerar Reporte,** el sistema he muestra una nueva ventana de reporte generado. 8. El usuario del sistema hace clic en **cerrar vista** parar cerrar vista **Reporte** **Curso alterno:** 1. El sistema muestra el mensaje de error “**Completar los campos.”** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **“Error en el registro.”** si no se conecta a la base de datos.
	Curso básico: 1. El usuario del sistema hace clic en el menú principal **“Productos”**, el sistema le muestra los submenús: **“Nuevo”**, **“Lista”** 2. El usuario del sistema hace clic en el submenú **“Nuevo”** y el sistema le muestra la página **“Nuevo Producto”,** el sistema le muestra el formulario **“producto”** dentro del cual se encuentra los campos **precio, nombre, cantidad, marca, autor, foto** para para ingresar las observaciones de la actividad, **“observación”**

CU04: mantener productos	para ingresar la descripción de la actividad; y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar producto"** y el sistema muestra la página **"Lista"** **6.** El usuario del sistema hace clic en el menú "**Listar**", el sistema le muestra el sub menú Lista, y los botones de Reporte. 7. el usuario de sistema hace clic en G**enerar Reporte,** el sistema he muestra una nueva ventana de reporte generado. 8. El usuario del sistema hace clic en **cerrar vista** parar cerrar vista **Reporte** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.
	Curso básico: 1. El usuario del sistema hace clic en el menú principal **"Categoría"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Categoría",** el sistema le muestra el formulario **"Categoría"** dentro del cual se encuentra los campos **Categoría** para para ingresar los datos categoría y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".**

CU06: mantener categoría	5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar Categoría"** y el sistema muestra la página **"Lista".** **6.** El usuario del sistema hace clic en el menú "**Listar**", el sistema le muestra el sub menú Lista, y los botones de Reporte. 7. el usuario de sistema hace clic en G**enerar Reporte,** el sistema he muestra una nueva ventana de reporte generado. 8. El usuario del sistema hace clic en **cerrar vista** parar cerrar vista **Reporte.** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.
CU07: realizar venta	**Curso básico:** 1. El usuario del sistema hace clic en el menú principal **"Venta"**, el sistema le muestra los submenús: **"Nuevo"**, **"Lista"** 2. El usuario del sistema hace clic en el submenú **"Nuevo"** y el sistema le muestra la página **"Nuevo Venta",** el sistema le muestra el formulario **"Venta"** dentro del cual se encuentra los campos **Año, numero, fecha, comprobante,** para para ingresar los datos de la venta, además de **menú cliente, DNI, Nombre, y menú detalle, Nombre Producto, Cantidad producto** y los botones "**Guardar**". 3. El usuario del sistema hace clic en "**Guardar**" y el sistema le muestra el mensaje **"Registro correcto".** 5. El usuario del sistema hace clic en "**Cancelar**" para cerrar el formulario **"Registrar Venta"** y el sistema muestra la página **"Lista".**

	6. El usuario del sistema hace clic en el menú "**Listar**", el sistema le muestra el sub menú Lista, y los botones de **Reporte**. 7. el usuario de sistema hace clic en G**enerar Reporte,** el sistema he muestra una nueva ventana de reporte generado. 8. El usuario del sistema hace clic en **cerrar vista** parar cerrar vista **Reporte.** **Curso alterno:** 1. El sistema muestra el mensaje de error "**Completar los campos."** si alguno de los campos está vacío. 2. El sistema muestra el mensaje de error **"Error en el registro."** si no se encuentra conectada al base de datos.

Tabla N.º 4.9. Revisión de casos de uso..

4.4.2. REVISIÓN DEL MODELO DE DOMINIO ACTUALIZADO

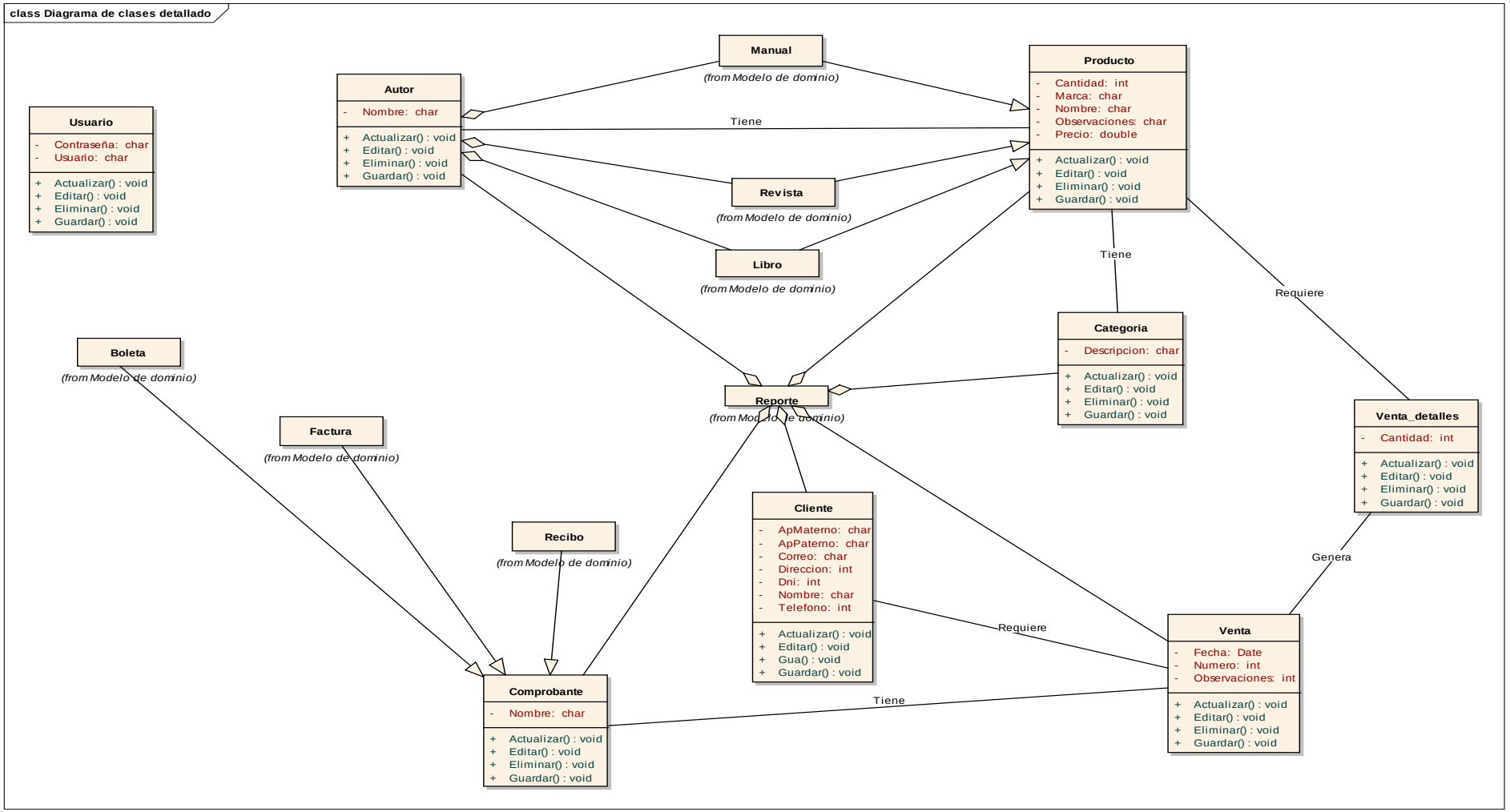

Figura N.º 4.35. Modelo de Dominio Revisado.

4.5. ARQUITECTURA TÉCNICA

4.5.1. ARQUITECTURA TÉCNICA POR CAPAS

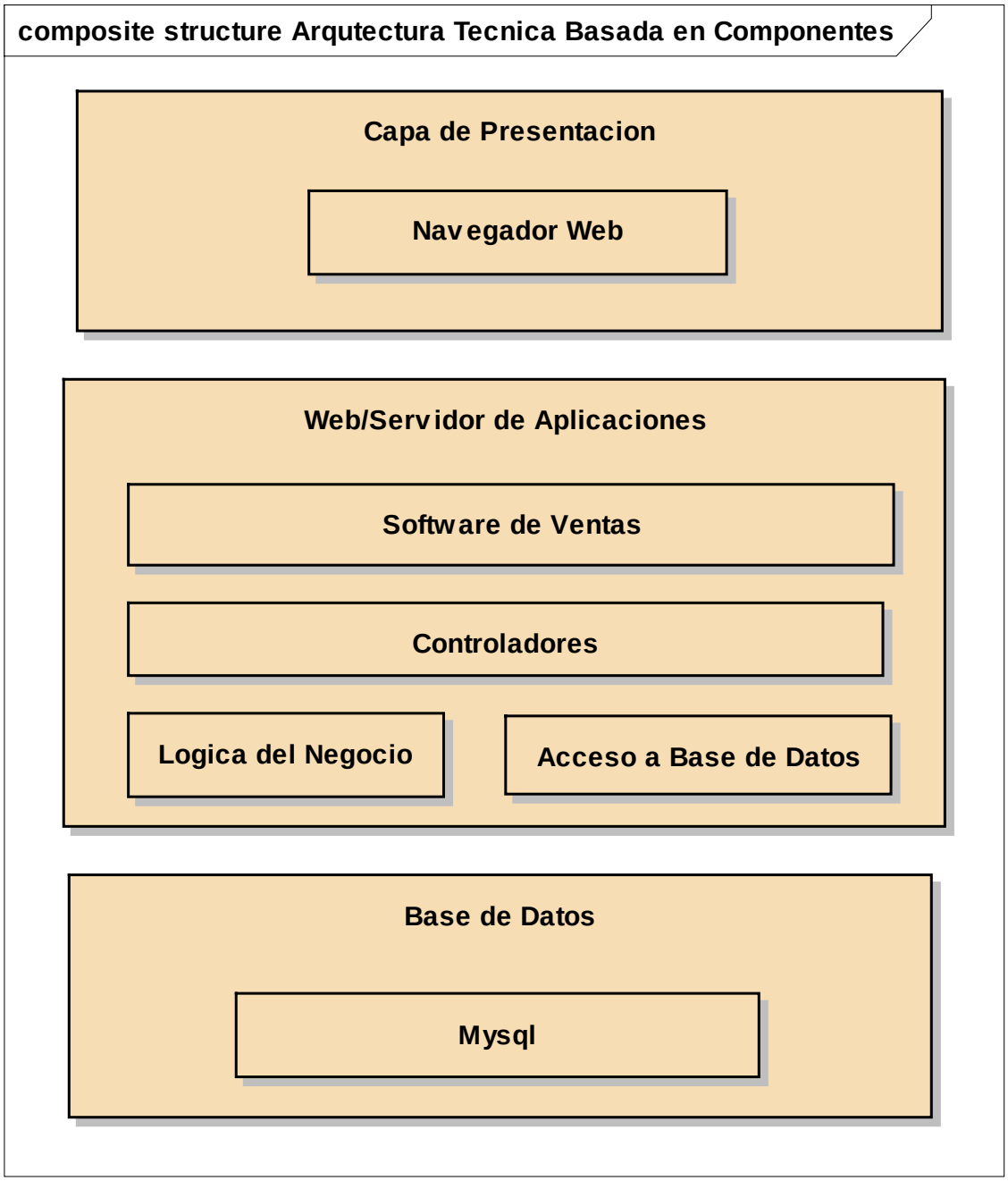

Figura N.º 4.36. Arquitectura basada en componentes.

4.5.2. DIAGRAMA DE COMPONENTES

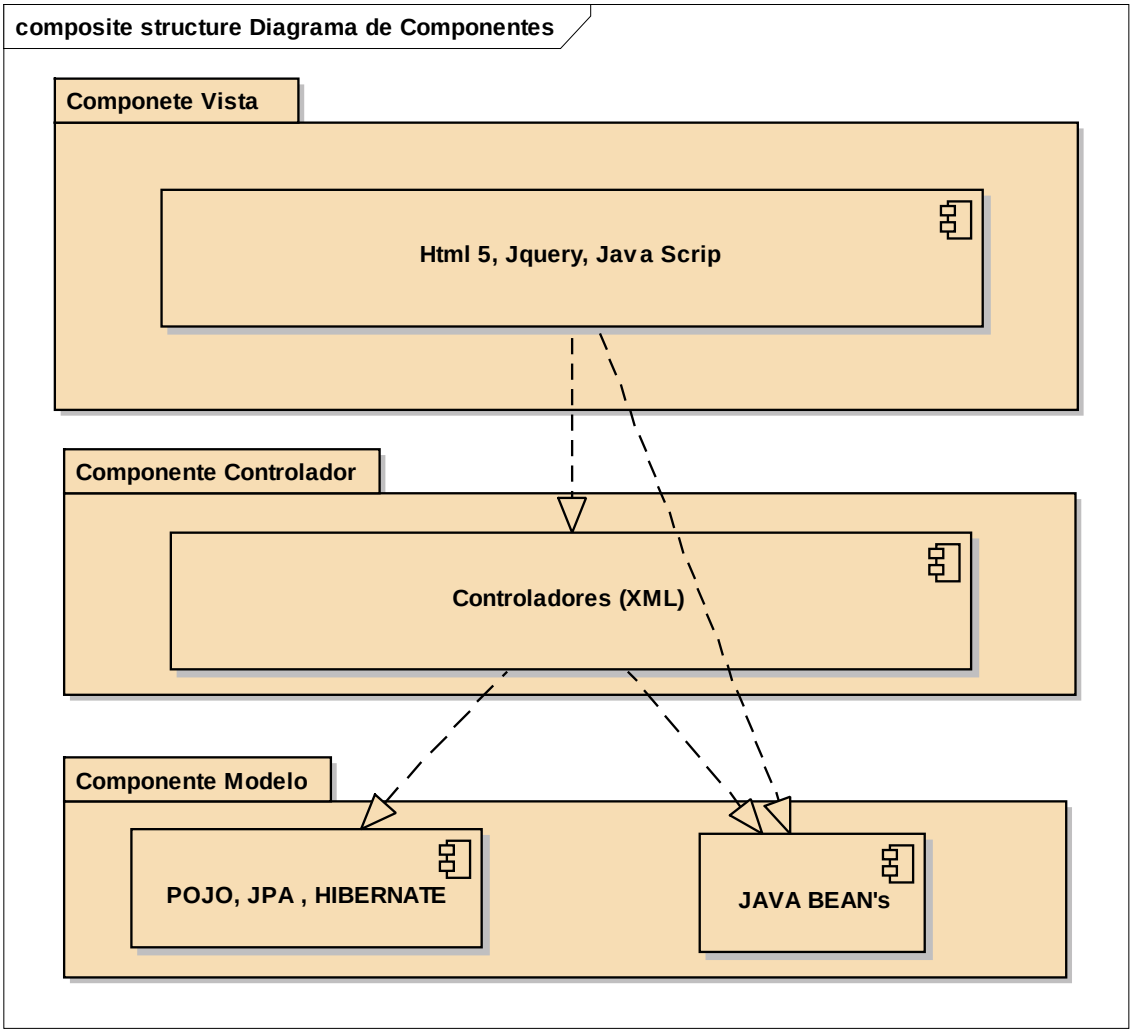

Figura N.º 4.37. Diagrama basado en componentes.

4.5.3. DIAGRAMA DE DESPLIEGUE

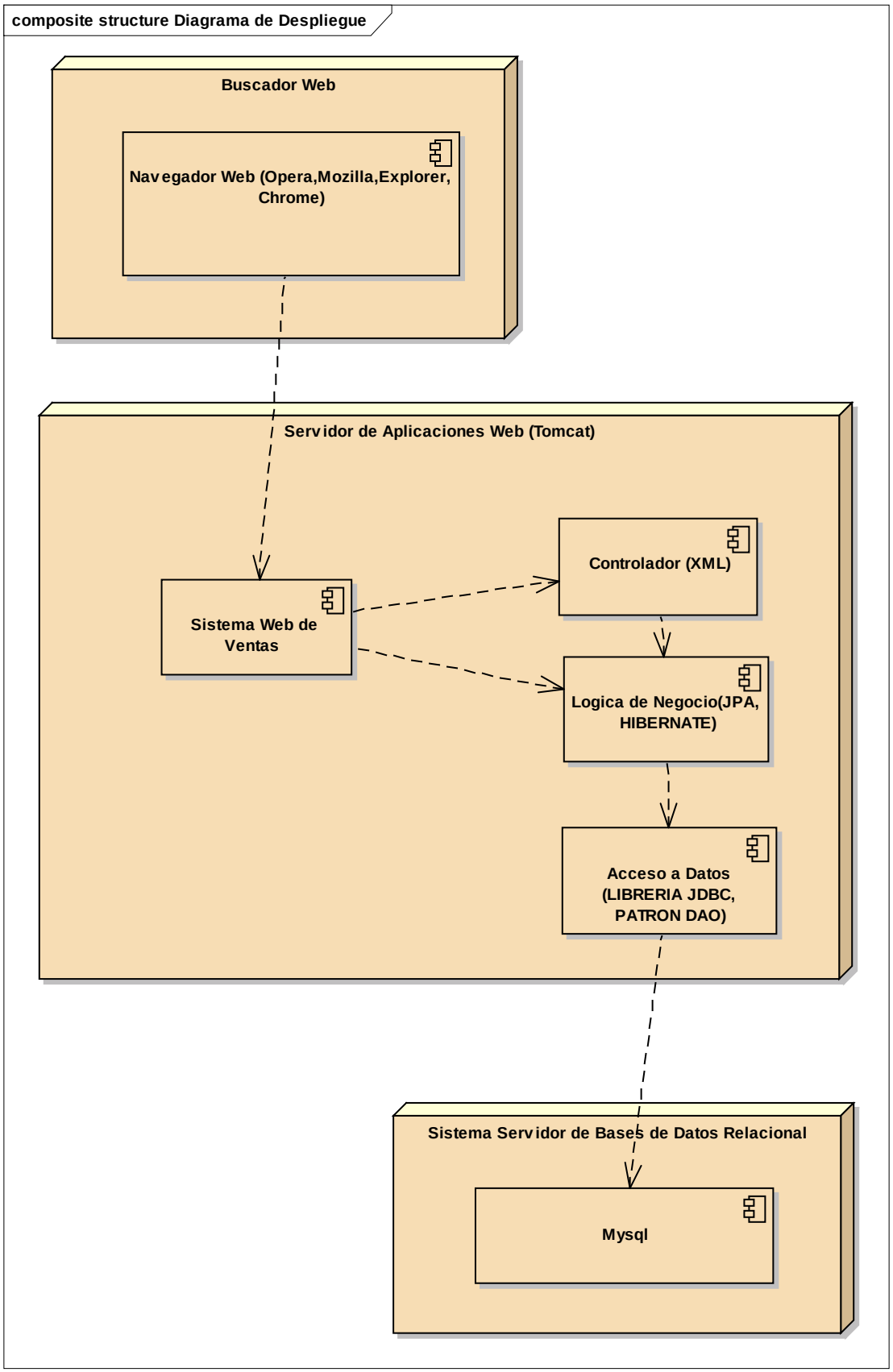

Figura N.º 4.38. Diagrama basado en componentes.

4.6. DISEÑO

4.6.1. DIAGRAMAS DE SECUENCIA

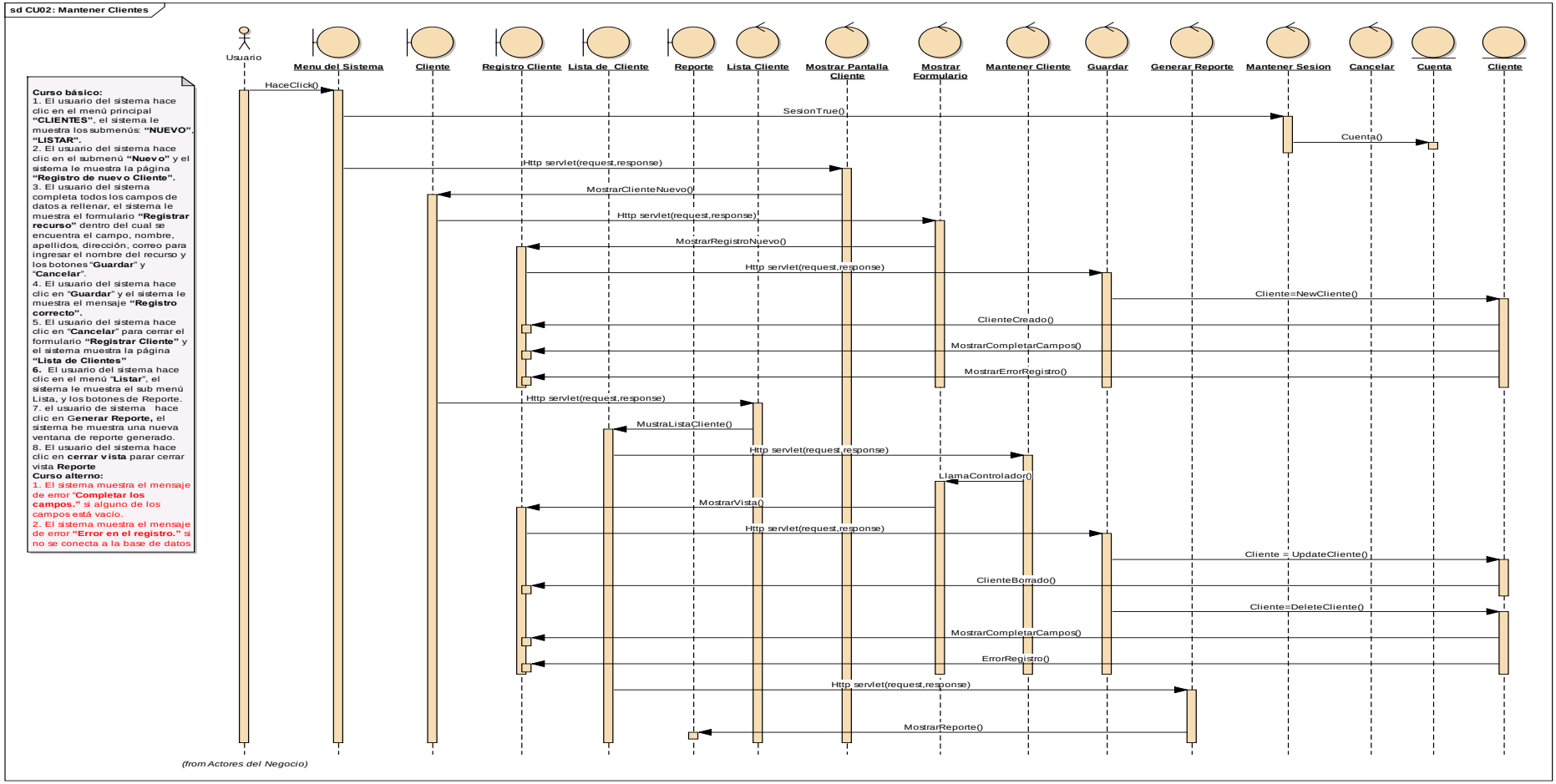

Figura N.º 4.39. Diagrama de secuencia, CU Mantener Cliente.

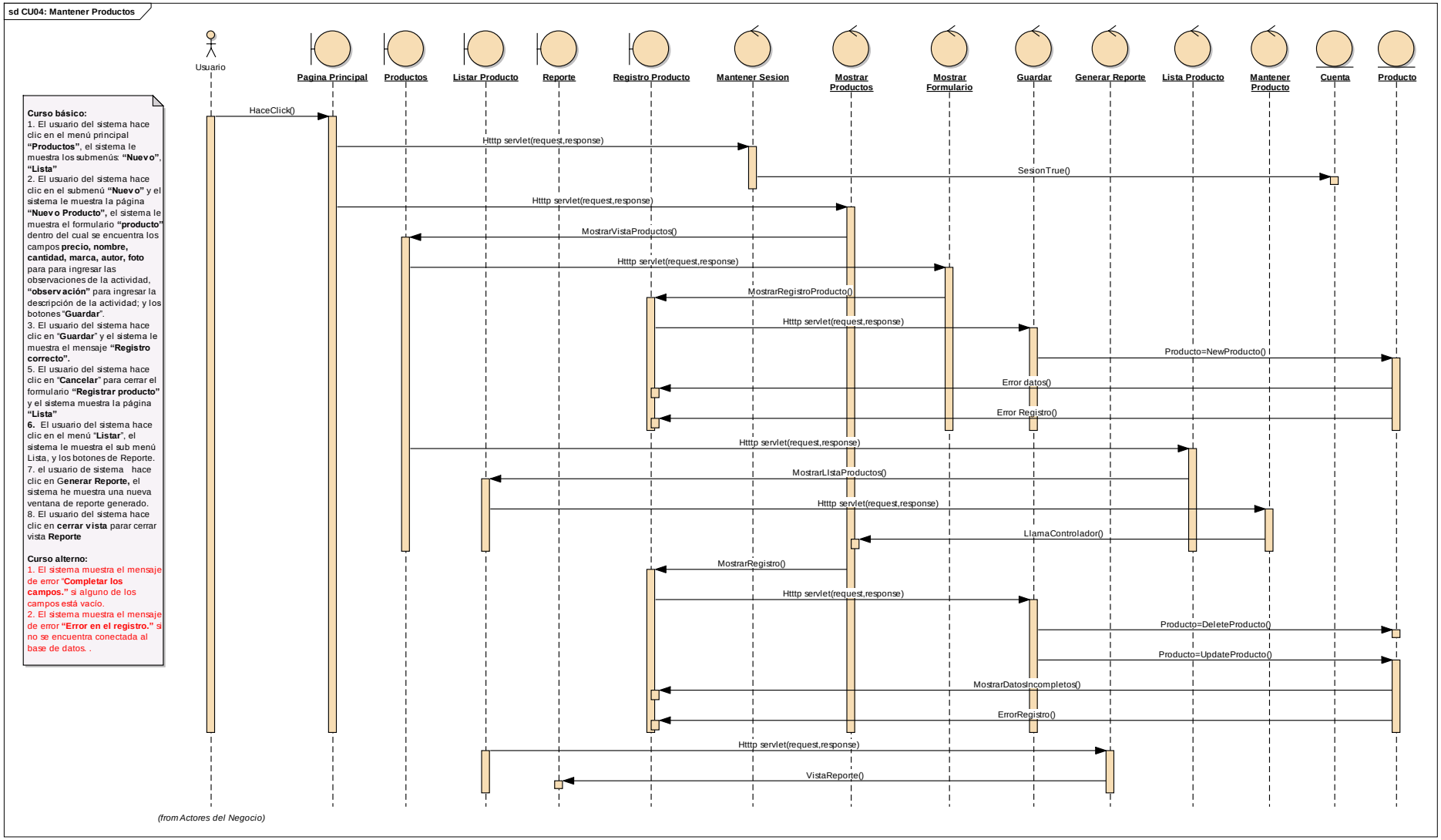

Figura N.º 4.40. Diagrama de secuencia, CU Mantener Producto.

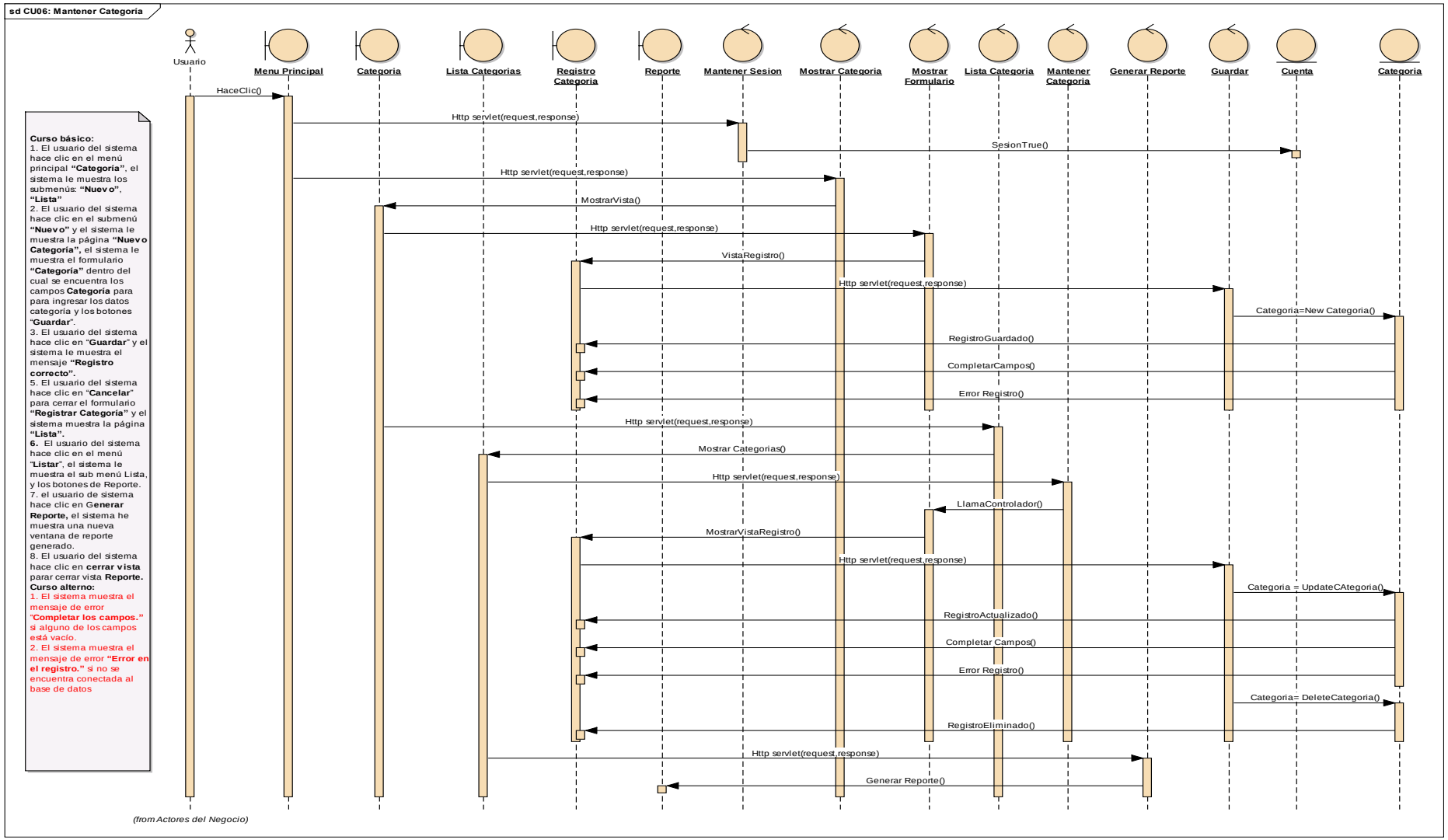

Figura N.º 4.41. Diagrama de secuencia, CU Mantener Categoría.

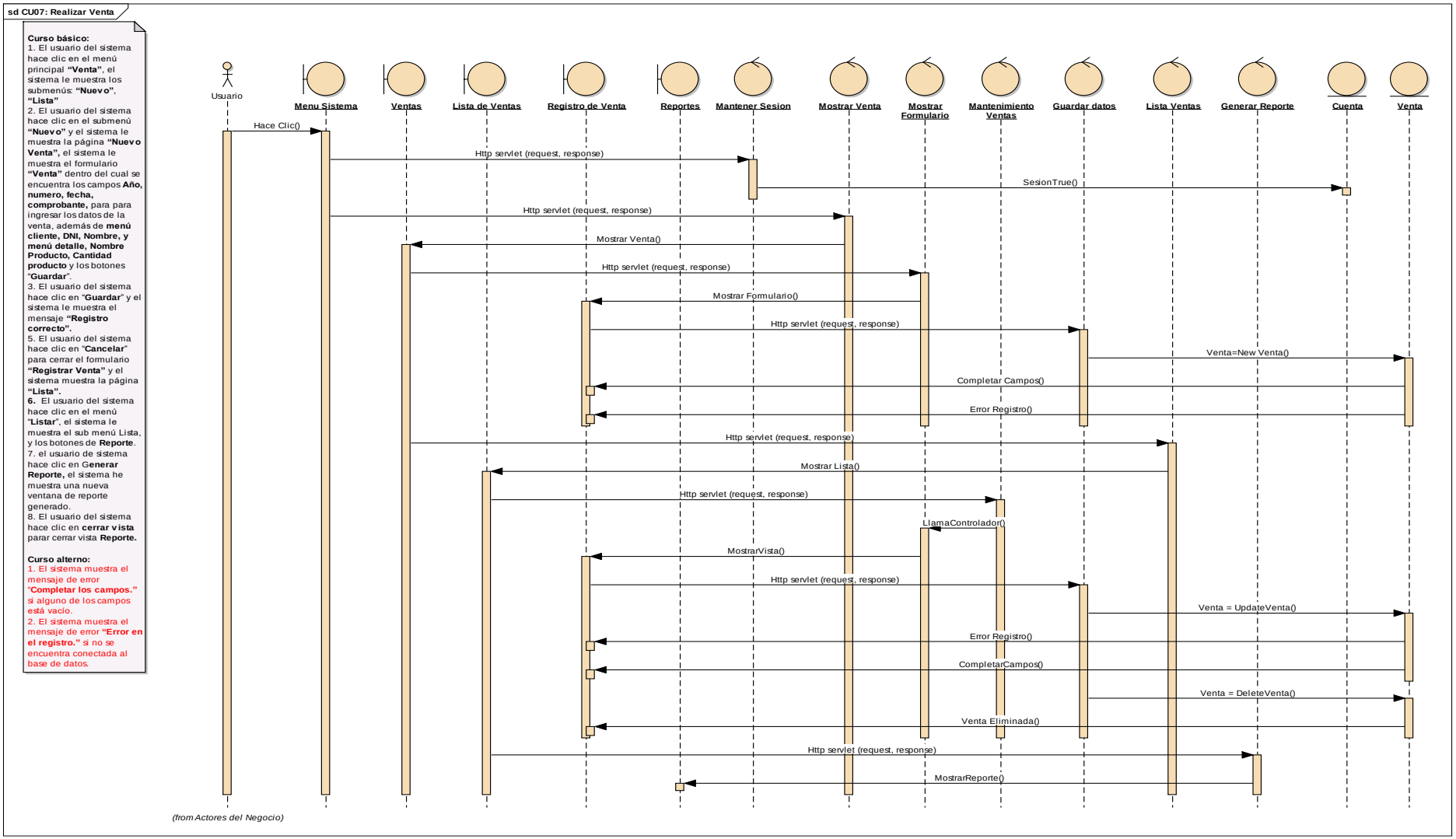

Figura N.º 4.42. Diagrama de secuencia, CU Realizar Venta.

4.6.2. BASE DE DATOS FISICA

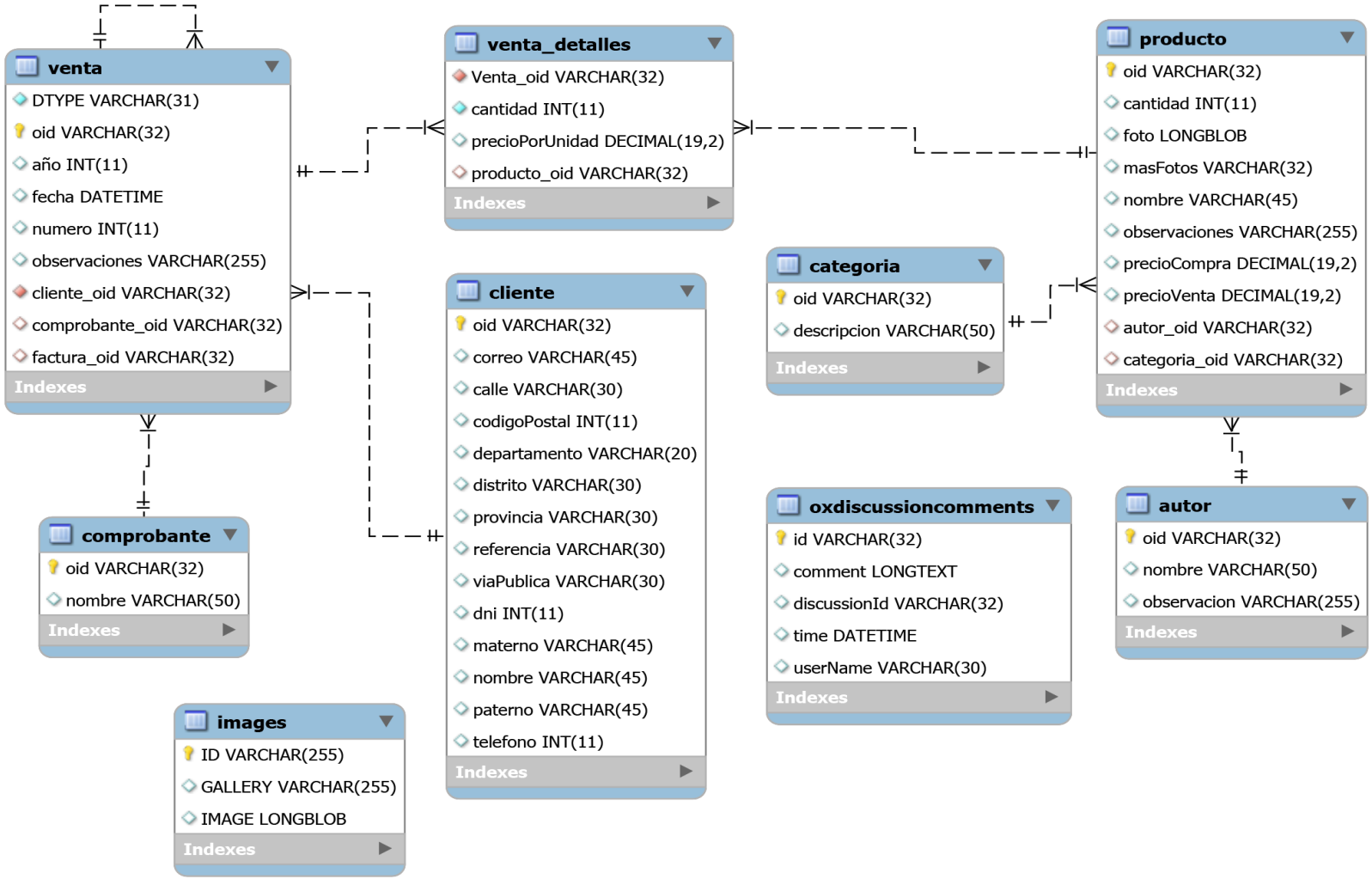

Figura N.º 4.43. Modelo de Base de Datos Física.

4.6.3. DIAGRAMA DE CLASES DE DISEÑO

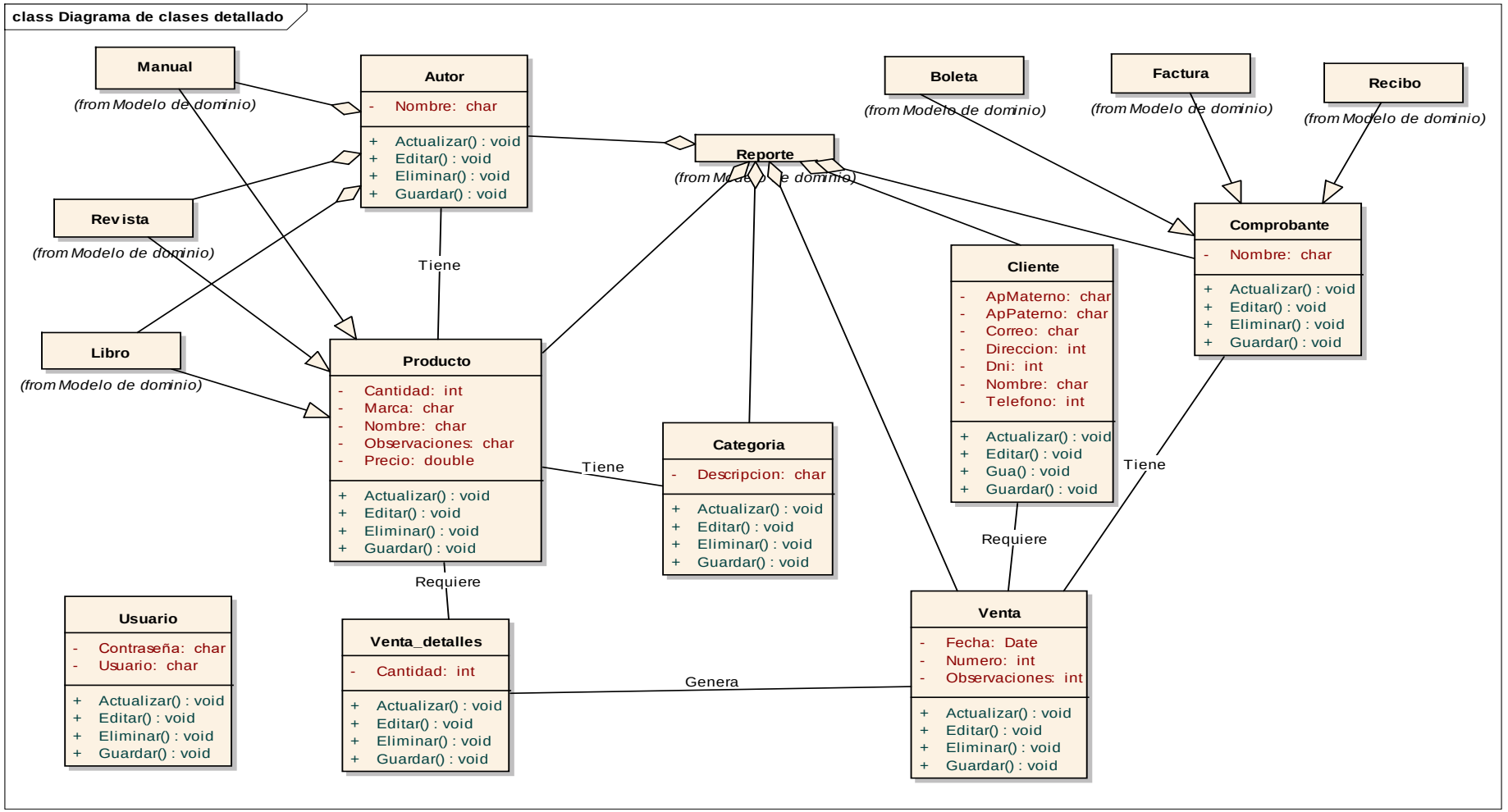

Figura N.º 4.44. Diagrama de clases de Diseño.

4.7. IMPLEMENTACIÓN

4.7.1. CREANDO LA BASE DE DATOS

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
-- -----------------------------------------------------
-- Schema facturacion
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `facturacion` DEFAULT CHARACTER SET utf8 ;
USE `facturacion` ;
-- -----------------------------------------------------
-- Table `facturacion`.`autor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `facturacion`.`autor` (
  `oid` VARCHAR(32) NOT NULL,
  `nombre` VARCHAR(50) NULL DEFAULT NULL,
  `observacion` VARCHAR(255) NULL DEFAULT NULL,
  PRIMARY KEY (`oid`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `facturacion`.`categoria`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `facturacion`.`categoria` (
  `oid` VARCHAR(32) NOT NULL,
```

```
  `descripcion` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`oid`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `facturacion`.`cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `facturacion`.`cliente` (
  `oid` VARCHAR(32) NOT NULL,
  `correo` VARCHAR(45) NULL DEFAULT NULL,
  `calle` VARCHAR(30) NULL DEFAULT NULL,
  `codigoPostal` INT(11) NULL DEFAULT NULL,
  `departamento` VARCHAR(20) NULL DEFAULT NULL,
  `distrito` VARCHAR(30) NULL DEFAULT NULL,
  `provincia` VARCHAR(30) NULL DEFAULT NULL,
  `referencia` VARCHAR(30) NULL DEFAULT NULL,
  `viaPublica` VARCHAR(30) NULL DEFAULT NULL,
  `dni` INT(11) NULL DEFAULT NULL,
  `materno` VARCHAR(45) NULL DEFAULT NULL,
  `nombre` VARCHAR(45) NULL DEFAULT NULL,
  `paterno` VARCHAR(45) NULL DEFAULT NULL,
  `telefono` INT(11) NULL DEFAULT NULL,
  PRIMARY KEY (`oid`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `facturacion`.`comprobante`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `facturacion`.`comprobante` (
  `oid` VARCHAR(32) NOT NULL,
  `nombre` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`oid`))
ENGINE = InnoDB
```

```
DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `facturacion`.`images`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `facturacion`.`images` (

  `ID` VARCHAR(255) NOT NULL,

  `GALLERY` VARCHAR(255) NULL DEFAULT NULL,

  `IMAGE` LONGBLOB NULL DEFAULT NULL,

  PRIMARY KEY (`ID`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `facturacion`.`oxdiscussioncomments`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `facturacion`.`oxdiscussioncomments` (

  `id` VARCHAR(32) NOT NULL,

  `comment` LONGTEXT NULL DEFAULT NULL,

  `discussionId` VARCHAR(32) NULL DEFAULT NULL,

  `time` DATETIME NULL DEFAULT NULL,

  `userName` VARCHAR(30) NULL DEFAULT NULL,

  PRIMARY KEY (`id`),

  INDEX `UK_bhlx4vunn24ym1tbrlgq8g30m` (`discussionId` ASC))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `facturacion`.`producto`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `facturacion`.`producto` (

  `oid` VARCHAR(32) NOT NULL,

  `cantidad` INT(11) NULL DEFAULT NULL,

  `foto` LONGBLOB NULL DEFAULT NULL,

  `marca` VARCHAR(10) NULL DEFAULT NULL,

  `masFotos` VARCHAR(32) NULL DEFAULT NULL,
```

```
  `nombre` VARCHAR(45) NULL DEFAULT NULL,
  `observaciones` VARCHAR(255) NULL DEFAULT NULL,
  `precio` DECIMAL(19,2) NULL DEFAULT NULL,
  `autor_oid` VARCHAR(32) NULL DEFAULT NULL,
  `categoria_oid` VARCHAR(32) NULL DEFAULT NULL,
  PRIMARY KEY (`oid`),
  INDEX `FK_hdubfflyc8k6u4vqr9gsuimj0` (`autor_oid` ASC),
  INDEX `FK_24q3u9oddswxjsaw6b0asasom` (`categoria_oid` ASC),
  CONSTRAINT `FK_24q3u9oddswxjsaw6b0asasom`
    FOREIGN KEY (`categoria_oid`)
    REFERENCES `facturacion`.`categoria` (`oid`),
  CONSTRAINT `FK_hdubfflyc8k6u4vqr9gsuimj0`
    FOREIGN KEY (`autor_oid`)
    REFERENCES `facturacion`.`autor` (`oid`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `facturacion`.`venta`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `facturacion`.`venta` (
  `oid` VARCHAR(32) NOT NULL,
  `año` INT(11) NULL DEFAULT NULL,
  `fecha` DATETIME NULL DEFAULT NULL,
  `numero` INT(11) NULL DEFAULT NULL,
  `observaciones` VARCHAR(255) NULL DEFAULT NULL,
  `cliente_oid` VARCHAR(32) NOT NULL,
  `comprobante_oid` VARCHAR(32) NULL DEFAULT NULL,
  PRIMARY KEY (`oid`),
  INDEX `FK_4mjidf6nb5ev58m4qbpg1w2nl` (`cliente_oid` ASC),
  INDEX `FK_jlct50ukctrhmk04ap30iv2xn` (`comprobante_oid` ASC),
  CONSTRAINT `FK_4mjidf6nb5ev58m4qbpg1w2nl`
    FOREIGN KEY (`cliente_oid`)
    REFERENCES `facturacion`.`cliente` (`oid`),
```

```
  CONSTRAINT `FK_jlct50ukctrhmk04ap30iv2xn`
    FOREIGN KEY (`comprobante_oid`)
    REFERENCES `facturacion`.`comprobante` (`oid`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------
-- Table `facturacion`.`venta_detalles`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `facturacion`.`venta_detalles` (
  `Venta_oid` VARCHAR(32) NOT NULL,
  `cantidad` INT(11) NOT NULL,
  `producto_oid` VARCHAR(32) NULL DEFAULT NULL,
  INDEX `FK_61tdy5tvn5jeichldrgr77eqe` (`producto_oid` ASC),
  INDEX `FK_4naevd594a300wxm1h51jdnon` (`Venta_oid` ASC),
  CONSTRAINT `FK_4naevd594a300wxm1h51jdnon`
    FOREIGN KEY (`Venta_oid`)
    REFERENCES `facturacion`.`venta` (`oid`),
  CONSTRAINT `FK_61tdy5tvn5jeichldrgr77eqe`
    FOREIGN KEY (`producto_oid`)
    REFERENCES `facturacion`.`producto` (`oid`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Tabla N° 4.9: Código SQL para generar la Base de Datos.

4.7.2. IMPLEMENTACIÓN POR CASOS DE USO

```
package org.eliaquim.facturacion.modelo;
import javax.persistence.*;
import javax.persistence.Entity;
import org.hibernate.annotations.GenericGenerator;
import org.openxava.annotations.*;
@Entity
@View(name = "Simple",
                members = "dni, nombre"
)
public class Cliente {
        @Id
        @GeneratedValue(generator = "system-uuid")
        @Hidden
        @GenericGenerator(name = "system-uuid", strategy = "uuid")
        @Column(length = 32)
        private String oid;

        @Column(length = 8) // La longitud de columna se usa a nivel UI  y DB
        @Required
        private int dni;
        @Column(length = 45) // La longitud de columna se usa a nivel UI y DB
        @Required // Se mostrará un error de validación
        private String nombre;
        @Column(length = 45)
        private String paterno;
        @Column(length = 45)
        private String materno;
        @Column(length = 45) // La longitud de columna se usa a nivel UI
        private int telefono;
        @Column(length = 45) // La longitud de columna se usa a nivel UI DB
        private String correo;
```

```
@Embedded // clase incrustable
private Direccion direccion; // Una referencia Java
public Direccion getDireccion() {
        if (direccion == null)
                direccion = new Direccion();
        return direccion;
}
public void setDireccion(Direccion direccion) {
        this.direccion = direccion;
}
public String getOid() {
        return oid;
}
public void setOid(String oid) {
        this.oid = oid;
}
public int getDni() {
        return dni;
}
public void setDni(int dni) {
        this.dni = dni;
}
public String getNombre() {
        return nombre;
}
public void setNombre(String nombre) {
        this.nombre = nombre;
}
public String getPaterno() {
        return paterno;
}
public void setPaterno(String paterno) {
        this.paterno = paterno;
```

```
        }
        public String getMaterno() {
                return materno;
        }
        public void setMaterno(String materno) {
                this.materno = materno;
        }
        public int getTelefono() {
                return telefono;
        }
        public void setTelefono(int telefono) {
                this.telefono = telefono;
        }
        public String getCorreo() {
                return correo;
        }
        public void setCorreo(String correo) {
                this.correo = correo;
        }
}
```

Tabla N° 4.10: Código generado, CU Mantener Cliente.

```
package org.eliaquim.facturacion.modelo;
import javax.persistence.*;
@Embeddable // Usamos @Embeddable en vez de @Entity
public class Direccion {
        @Column(length = 30) // Los miembros se anotan igual que en las
        private String viaPublica;
        @Column(length = 5)
        private int codigoPostal;
        @Column(length = 20)
```

```
    private String departamento;
    @Column(length = 30)
    private String provincia;
    @Column(length = 30)
    private String distrito;
    @Column(length = 30)
    private String calle;
    @Column(length = 30)
    private String referencia;

    public String getViaPublica() {
        return viaPublica;
    }
    public void setViaPublica(String viaPublica) {
        this.viaPublica = viaPublica;
    }
    public int getCodigoPostal() {
        return codigoPostal;
    }
    public void setCodigoPostal(int codigoPostal) {
        this.codigoPostal = codigoPostal;
    }
    public String getDepartamento() {
        return departamento;
    }
    public void setDepartamento(String departamento) {
        this.departamento = departamento;
    }
    public String getProvincia() {
        return provincia;
    }
    public void setProvincia(String provincia) {
        this.provincia = provincia;
```

```
        }
        public String getDistrito() {
                return distrito;
        }
        public void setDistrito(String distrito) {
                this.distrito = distrito;
        }
        public String getCalle() {
                return calle;
        }
        public void setCalle(String calle) {
                this.calle = calle;
        }
        public String getReferencia() {
                return referencia;
        }
        public void setReferencia(String referencia) {
                this.referencia = referencia;
        }
}
```

Tabla N° 4.11: Código generado, CU Mantener Cliente – Dirección cliente.

```
package org.eliaquim.facturacion.modelo;
import java.math.*;
import javax.persistence.*;
import javax.persistence.Entity;
import org.hibernate.annotations.*;
import org.openxava.annotations.*;
@Entity
@View(members = "nombre,categoria,autor;" + "precio,cantidad,marca;" +
"foto,masFotos;" + "observaciones")
```

```
public class Producto {
        @Id
        @GeneratedValue(generator = "system-uuid")
        @Hidden
        @GenericGenerator(name = "system-uuid", strategy = "uuid")
        @Column(length = 32)
        private String oid;
        @Column(length = 45)
        @Required
        private String nombre;
        @ManyToOne( // La referencia se almacena como una relación e
                        fetch = FetchType.LAZY, // La referencia se carga
                        optional = true) // La referencia pue de estar sin valor
        @DescriptionsList // Así la referencia se visualiza usando un combo
        private Categoria categoria;

        @Stereotype("DINERO") // La propiedad precio se usa para almacenar dinero
        private BigDecimal precio;
        @Column(length = 4)
        private int cantidad;
        @Column(length = 10)
        private String marca;
        @Stereotype("FOTO") // El usuario puede ver y cambiar una foto
        @Column(length = 16777216) // Este tamaño para poder guardar fotos
        private byte[] foto;
        @Stereotype("GALERIA_IMAGENES") // Una galería de fotos
        @Column(length = 32) // La cadena de 32 de longitud es para
        private String masFotos;
        @ManyToOne(fetch = FetchType.LAZY)
        @DescriptionsList
        private Autor autor;
```

```
	@Stereotype("TEXTO_GRANDE") // Esto es para un texto grande, se usará un área de texto o equivalente
	private String observaciones;
	public String getOid() {
		return oid;
	}
	public void setOid(String oid) {
		this.oid = oid;
	}
	public Autor getAutor() {
		return autor;
	}
	public void setAutor(Autor autor) {
		this.autor = autor;
	}
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public int getCantidad() {
		return cantidad;
	}
	public void setCantidad(int cantidad) {
		this.cantidad = cantidad;
	}
	public String getObservaciones() {
		return observaciones;
	}
	public void setObservaciones(String observaciones) {
		this.observaciones = observaciones;
	}
```

```
        public String getMasFotos() {
                return masFotos;
        }
        public void setMasFotos(String masFotos) {
                this.masFotos = masFotos;
        }
        public byte[] getFoto() {
                return foto;
        }
        public void setFoto(byte[] foto) {
                this.foto = foto;
        }
        public BigDecimal getPrecio() {
                return precio;
        }
        public void setPrecio(BigDecimal precio) {
                this.precio = precio;
        }
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public Categoria getCategoria() {
                return categoria;
        }
        public void setCategoria(Categoria categoria) {
                this.categoria = categoria;
        }
}
```

Tabla N° 4.12: Código generado, CU Mantener Producto.

```
package org.eliaquim.facturacion.modelo;
import java.util.*;
import javax.persistence.*;
import org.hibernate.annotations.GenericGenerator;
import org.openxava.annotations.*;
@Entity
public class Autor {
	@Id
	@GeneratedValue(generator = "system-uuid")
	@Hidden
	@GenericGenerator(name = "system-uuid", strategy = "uuid")
	@Column(length = 32)
	private String oid;
	@Column(length = 50)
	@Required
	private String nombre;
	@Stereotype("TEXTO_GRANDE")
	private String observacion;
	@OneToMany(mappedBy = "autor")
	@ListProperties("nombre, precio")
	private Collection<Producto> productos;
	public Collection<Producto> getProductos() {
		return productos;
	}
	public void setProductos(Collection<Producto> productos) {
		this.productos = productos;
	}
	public String getOid() {
		return oid;
	}
	public void setOid(String oid) {
		this.oid = oid;
	}
```

```
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public String getObservacion() {
                return observacion;
        }
        public void setObservacion(String observacion) {
                this.observacion = observacion;
        }
}
```

Tabla N° 4.13: Código generado, CU Mantener Autores.

```
package org.eliaquim.facturacion.modelo;
import javax.persistence.*;
import org.hibernate.annotations.GenericGenerator;
import org.openxava.annotations.*;
@Entity
public class Categoria {
        @Id
        @Hidden // Es un identificador interno
        @GeneratedValue(generator="system-uuid") // Identificador Universal
        @GenericGenerator(name="system-uuid", strategy = "uuid")
        @Column(length=32)
        private String oid;
                @Column(length=50)
        private String descripcion;
                public String getOid() {
```

```
            return oid;
        }

        public void setOid(String oid) {
            this.oid = oid;
        }
        public String getDescripcion() {
            return descripcion;
        }
        public void setDescripcion(String descripcion) {
            this.descripcion = descripcion;
        }
    }
```

Tabla N° 4.14: Código generado, CU Mantener Categoría.

```
package org.eliaquim.facturacion.modelo;
import java.util.*;
import javax.persistence.*;
import org.eliaquim.facturacion.calculadores.*;
import org.hibernate.annotations.GenericGenerator;
import org.openxava.annotations.*;
import org.openxava.calculators.*;
@Entity

@View(members = "año, numero, fecha,comprobante;" + "cliente;" + "detalles;" +
"observaciones")
public class Venta {
    @Id
    @GeneratedValue(generator = "system-uuid")
    @Hidden
    @GenericGenerator(name = "system-uuid", strategy = "uuid")
```

```
        @Column(length = 32)
        private String oid;
        @Column(length = 4)
        @DefaultValueCalculator(CurrentYearCalculator.class) // Año actual
        private int año;
        @Column(length = 6)
        @DefaultValueCalculator(value = CalculadorSiguienteNumeroParaAño.class,
properties = @PropertyValue(name = "año"))
        private int numero;
        @Required
        @DefaultValueCalculator(CurrentDateCalculator.class) // Fecha actual
        private Date fecha;

        @ManyToOne(fetch = FetchType.LAZY)
        @DescriptionsList
        private Comprobante comprobante;

        @ManyToOne(fetch = FetchType.LAZY, optional = false) // El cliente es
obligatorio
        @ReferenceView("Simple") // La vista llamada 'Simple' se usará para
visualizar esta referencia
        private Cliente cliente;
        @ElementCollection
        @ListProperties("producto.nombre, cantidad")
        private Collection<Detalle> detalles;

        public Comprobante getComprobante() {
                return comprobante;
        }
        public void setComprobante(Comprobante comprobante) {
                this.comprobante = comprobante;
        }
        @Stereotype("TEXTO_GRANDE")
```

```
private String observaciones;

public Collection<Detalle> getDetalles() {
        return detalles;
}
public void setDetalles(Collection<Detalle> detalles) {
        this.detalles = detalles;
}
public Cliente getCliente() {
        return cliente;
}
public void setCliente(Cliente cliente) {
        this.cliente = cliente;
}
public String getOid() {
        return oid;
}
public void setOid(String oid) {
        this.oid = oid;
}
public int getAño() {
        return año;
}
public void setAño(int año) {
        this.año = año;
}
public int getNumero() {
        return numero;
}
public void setNumero(int numero) {
        this.numero = numero;
}
public Date getFecha() {
```

```
            return fecha;
        }
        public void setFecha(Date fecha) {
            this.fecha = fecha;
        }
        public String getObservaciones() {
            return observaciones;
        }
        public void setObservaciones(String observaciones) {
            this.observaciones = observaciones;
        }
}
```

Tabla N° 4.15: Código generado, CU Realizar Venta.

```
package org.eliaquim.facturacion.modelo;
import javax.persistence.*;

@Embeddable

public class Detalle {
        private int cantidad;
        @ManyToOne(fetch = FetchType.LAZY, optional = true)
        private Producto producto;

        public int getCantidad() {
            return cantidad;
        }
        public void setCantidad(int cantidad) {
            this.cantidad = cantidad;
        }
        public Producto getProducto() {
```

```
            return producto;
        }
        public void setProducto(Producto producto) {
            this.producto = producto;
        }
}
```

Tabla N° 4.16: Código generado, CU Realizar Venta - Detalle de producto.

```
package org.eliaquim.facturacion.calculadores;
import javax.persistence.*;
import org.openxava.calculators.*;
import org.openxava.jpa.*;

public class CalculadorSiguienteNumeroParaAño implements ICalculator {

        private int año; // Este valor se inyectará (usando su setter) antes de calcular

        public Object calculate() throws Exception { // Hace el cálculo
            Query query = XPersistence.getManager() // Una consulta JPA
                    .createQuery("select max(f.numero) from Factura f
where f.año = :año"); // La consulta devuelve
            // el número de factura máximo del año indicado
            query.setParameter("año", año); // Ponemos el año inyectado como
parámetro de la consulta
            Integer ultimoNumero = (Integer) query.getSingleResult();
            return ultimoNumero == null ? 1 : ultimoNumero + 1; // Devuelve el
último número
            // de factura del año + 1 o 1 si no hay último número
        }

        public int getAño() {
```

```
            return año;
        }

        public void setAño(int año) {
            this.año = año;
        }
}
```

Tabla N° 4.17: Código generado, CU Realizar Venta – Numero de Venta.

```
package org.eliaquim.facturacion.modelo;
import javax.persistence.*;
import org.hibernate.annotations.GenericGenerator;
import org.openxava.annotations.*;
@Entity
public class Comprobante {
        @Id
        @Hidden // La propiedad no se muestra al usuario. Es un identificador interno
        @GeneratedValue(generator = "system-uuid") // Identificador Universal
        @GenericGenerator(name = "system-uuid", strategy = "uuid")
        @Column(length = 32)
        private String oid;
        @Column(length = 50)
        private String nombre;

        public String getOid() {
            return oid;
        }
        public void setOid(String oid) {
            this.oid = oid;
        }
```

```
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
}
```

Tabla N° 4.18: Código generado, CU Mantener Comprobante.

4.8. PRUEBAS

Nº	CASO DE USO	MÉTODO	ESTADO
CU01	Autenticar Usuario	List(int id)	Exitoso
CU02	Mantener Clientes	Save()	Exitoso
		Edit()	Exitoso
		Delete()	Exitoso
		List(int id)	Exitoso
CU03	Generar Reportes	Save()	Exitoso
		Edit()	Exitoso
		Delete()	Exitoso
		List(int id)	Exitoso
CU04	Mantener Producto	Save()	Exitoso
		Edit()	Exitoso
		Delete()	Exitoso
		List(int id)	Exitoso
CU05	Mantener Autores	Save()	Exitoso
		Edit()	Exitoso
		Delete()	Exitoso
		List(int id)	Exitoso
CU06	Mantener Categoría	Save()	Exitoso
		Edit()	Exitoso

		Delete()	Exitoso
		List(int id)	Exitoso
CU07	Realizar Venta	Save()	Exitoso
		Edit()	Exitoso
		Delete()	Exitoso
		List(int id)	Exitoso
CU08	Mantener Comprobante	Save()	Exitoso
		Edit()	Exitoso
		Delete()	Exitoso
		List(int id)	Exitoso

Tabla N° 4.19: Tabla – Pruebas de Casos de Uso.

4.9. RESULTADOS

Se plantea la utilización del modelo de Framework de código abierto java como una alternativa que mejora el tiempo de desarrollo y producción e las aplicaciones web empresariales basados en componentes de negocio para entornos agiles, como se vio en el caso de análisis, desarrollo y entrega de un sistema de ventas y facturación.

a) La utilización de los Plain Old Java Object (POJOs) permite mejorar la funcionalidad del componente negocio, debido a la facilidad que nos permite mapear y generar las clases del programa, orientado a los componentes a través de los casos de uso planteados. Ya que con los Plain Old Java Object (POJOs) se crean las tablas, las relaciones y las bases de datos de manera automática y estas se actualizan en el caso de las modificaciones que existan de acuerdo a las necesidades de las empresas.

b) A través de la utilización de las anotaciones, llegamos a conocer un sin fin de posibilidades de mejoras en el diseño de los sistemas de información empresariales, con las cuales podemos hacer que el diseño adecuado sea los más exigente para lograr una Interfaz amigable que sea aprobada por los usuarios finales, para ello con ayuda de las anotaciones podemos generar listas, menús,

combos, mensajes pop-up, además de la generación de código HTML5 (formularios, botones, inputs, labels), todo estas opciones simbolizan una ventaja a la hora de planificar el desarrollo de las aplicaciones web pues no demanda mucho el tiempo y más nos enfocamos al análisis de los negocios. Sin embargo, esto también significa que hay que profundizar el análisis de las herramientas frontend, con esto nos obliga a conocer más a fondo el lenguaje HTML, JS y CSS, lo cual en el fondo tampoco es tan malo, pues nos ayudara mucho si queremos ampliar más los conocimientos.

c) Se propone la utilización de Java Persistence API (JPA) en el desarrollo de las aplicaciones web empresariales, pues mejora de gran manera el procesamiento de los datos, uno de los aspectos fundamentales es pues que a través de este medio java puede recuperar la información de manera eficiente y rápida desde los sistemas de almacenamiento no volátiles, además dentro del conjunto de aplicaciones para gestionar las persistencia de datos en la aplicaciones web, Java Persistence API (JPA) proporciona mayor estabilidad y mejor tiempo de respuesta para gestionar desde las bases de datos relacionales y de esta manera logran ser más fructíferas. Otra de la característica que podemos añadir es que Java Persistence API (JPA) simplifica el desarrollo de la persistencia de datos.

4.10. ANÁLISIS DE RESULTADOS

Visto la correcta ejecución de la aplicación sistema de ventas para una librería, en todo el Capítulo IV análisis y resultados aplicando la metodología ICONIX respectivamente, se concluye que el modelo de Framework de código abierto en Java, Plain Old Java Object (POJOs), Anotaciones y Java Persistence API (JPA) facilitan el rápido desarrollo de Aplicaciones Web Empresariales Basadas En Componentes De Negocio Para Entornos Agiles.

CAPITULO V

CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

a) De acuerdo al numeral 4.9), del Capítulo IV - Resultados, se afirma coherentemente, que el modelo de Framework de código abierto java facilita el rápido desarrollo de aplicaciones web empresariales basadas en componentes de negocio para entornos agiles.

b) De acuerdo al numeral 4.9), del Capítulo IV - Resultados, se afirma que los Plain Old Java Object (POJOs) facilitan el rápido desarrollo de Aplicaciones Web Empresariales Basadas En Componentes De Negocio Para Entornos Agiles

c) De acuerdo al numeral 4.9), del Capítulo IV - Resultados, se afirma que las Anotaciones facilita el rápido desarrollo de Aplicaciones Web Empresariales Basadas En Componentes De Negocio Para Entornos Agiles.

d) De acuerdo al numeral 4.9), del Capítulo IV - Resultados, se afirma que Java Persistence API (JPA) facilita el rápido desarrollo de Aplicaciones Web Empresariales Basadas En Componentes De Negocio Para Entornos Agiles.

5.2. RECOMENDACIONES

a) Se sugiere a la comunidad Open Source y programadores java utilizar el modelo de Framework de código abierto java para el desarrollo de sus aplicaciones web empresariales.

b) Se recomienda investigar más sobre las interfaces de las aplicaciones empresariales desde el punto de vista de manejo de las anotaciones en el lenguaje java.

c) Se recomienda el análisis del modelo de Framework de código abierto java visto desde otras metodologías agiles de desarrollo como Scrum y/o XP.

BIBLIOGRAFÍA

Agustin Yagüe y Juan Garbajosa. (2009). Comparativa práctica de las pruebas en entornos tradicionales y ágiles. *Revista Española de Innovación, Calidad e Ingeniería del Software, Vol.5, No. 4,*.

Antonio, J. (2001). *El gran libro del protocolo.* Madrid, España: Autor.

BARROS, O. (2002). COMPONENTES DE LÓGICA DEL NEGOCIO DESARROLLADOS A PARTIR DE PATRONES DE PROCESOS. *REVISTA INGENIERÍA DE SISTEMAS*, 4 - 19.

Bernal, C. . (2006). *Metodología de la investigación (3ª ed.).* Bogotá, Colombia: Pearson educación.

CAFASSI, E. (1998). *Internet: Políticas y comunicaciones. .* Buenos Aires, Argentina: Biblos.

ECURED. (2015). *ECURED*. Obtenido de https://www.ecured.cu/Framework

Esencial, C. (2011). *Esencial Internet Explorer 9.* Cataluña, España: Editions ENI.

Esencial, C. (2011). *Esencial Internet Explorer 9.* Cataluña, España: Editions ENI.

GÓMEZ, V. H. (2010). *SISTEMA DE INFORMACION PARA EL CONTROL, SEGUIMIENTO Y MANTENIMIENTO DEL EQUIPAMIENTO HOSPITALARIO.* LIMA - PERU.

Gourley D. y Totty B. (2002). *HTTP: The Definitive Guide [HTTP: La Guía Definitiva]*. California, US: O'Reilly Media, Inc.

Grajales, T. (2000). *TIPOS DE INVESTIGACION*.

Groussard, T. (2010). *Recursos Informaticos Java Enterprise Edition – Desarrollo de aplicadiones web con JEE 6.* Cataluña, España: Editions ENI.

Hibernate. (2011). *hibernate.org*. Obtenido de hibernate.org: http://hibernate.org/

IBM. (2015). *IBM Tivoli Application Dependency Discovery Manager.* EE.UU.: Licensed Materials - Property of IBM. © Copyright IBM.

IBM. (s.f.). *IBM* . Obtenido de IBM knowledge center: https://www.ibm.com/support/knowledgecenter/es/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_jpa.html

Johnson, R. (17 de enero de 2005). *Frameworks de desarrollo J2EE*. Obtenido de https://ieeexplore.ieee.org/document/1381270/authors

Jose Miguel oedax Casaa, P. A.-U. (2012). *PROGRAMACION WEB EN JAVA* . ESPAÑA: Aula Mentor.

Kroenke, D. (2003). *Procesamiento de base de datos: fundamentos, diseño e implementación (8ª Ed.).* Juárez, México: Pearson.

Luján, S. (2001). *Programación en internet: Clientes web.* Alicante, España: Club Universitario.

M. Degiovannini. (02 de 2007). *Comparativa de Frameworks Web.* Obtenido de JavaHispano.org: http://static1.1.sqspcdn.com/static/f/923743/15025206/1320739503647/frameworks_web.pdf?token=0p0jLXicjEHOLxPkNFSBDnZXMYo%3D

Mora, S. L. (2006). *Programacion de aplicasiones web: historia, principios basicos y clientes web.*

Ojeda, Á. V. (2008). *Análisis, Diseño e Implementación de un DataWarehouse de Soporte de Decisiones para un Hospital del Sistema de Salud Público.* Lima - Peru .

Oracle, J. (2013). *The Java Enterprise Edition* .

OSI, O. S. (22 de 03 de 2007). *Open Source Initiative*. Obtenido de https://opensource.org/

Pérez, J. E. (2008). *Introducción a AJAX*.

Porras, E. (2011). *La Metodologia Agil y Formal ICONIX para el Desarrollo de Software: Teoria y Practica.* . Ayacucho,Peru.

PROJECT, W. (24 de 12 de 2018). *Wikipedias*. Obtenido de https://es.wikipedia.org/wiki/Procesamiento_de_datos

RAE. (2011). *RAE*. Obtenido de http://dle.rae.es/?id=Bskzsq5|BsnXzV1

RAE. (2014). Obtenido de http://dle.rae.es/?id=UFLxCoW

REYES, C. T. (2013). *SISTEMA DE PAGOS BASADO EN EL ESTÁNDAR ISO 8583 Y NORMAS PCI DSS UTILIZANDO LECTORES DE BANDA MAGNÉTICA DESAROLLADO EN JAVA.* GUAYAQUIL – ECUADOR.

Rober-Alonso, C.-C. (2014). *Estándar de usabilidad para la interfaz gráfica de usuario en los proyectos de desarrollo de software.* LOJA-ECUADOR.

ROBERTO, M. P. (2012). *METODOLOGÍA DE LA.* Lima: UNIVERSIDAD NACIONAL DE EDUCACIÓN ENRIQUE GUZMÁN Y VALLE.

Rodríguez, D. I. (2015). *INTERFACES GRÁFICAS DE USUARIOS.*

Romero, L. (1997). *Publicar en Internet: guía práctica para la creación de documentos HTML*. Cantabria, España: Universidad de Cantabria.

Rosenberg, D. S. (2007). *Use Case Driven Object Modeling with UML: Teory and Practive.* EEUU-Apress.

SÁNCHEZ, J. L. (2007). *DEL SOFTWARE LIBRE AL CONOCIMIENTO LIBRE: ARGUMENTOS DE CARÁCTER TÉCNICO PARA ASPIRAR A UNA SOCIEDAD DIGITAL UNIVERSAL, IGUALITARIA Y LIBRE.*

Scuse, D. (2018). *Eclipse 3.1.* University of Manitoba.

Suh, W. (2004). *Web Engineering: Principles and Techniques [Ingeniería web: principios y técnicas]*. Pensilvania, USA: Idea Group Publishing.

Supo, D. J. (2012). *Seminarios de Investigacion Cientifica.* CreateSpace Independent Publishing Platform .

Tamayo, M. (2004). *El proceso de la investigación científica (4ª ed.).* México,México: Limusa.

TAMAYO, M. T. (2004). *EL PROCESO DE LA INVESTIGACION CIENTIFICA* . MEXICO : LIMUSA, S.A. GRUPO NORIEGA EDITORES.

Thibaud, C. (2006). *Mysql 5 , Instalacion, Implementacion, Administracion, Programacion.* Barcelona: Editions ENI.

Tomás, J. (2009). *Fundamentos de bioestadística y análisis de datos para enfermería (1ª ed.).* Barcelona, España: Servei de Publicacions.

VILLALOBOS, G. M. (2010). *DISEÑO DE FRAMEWORK WEB PARA EL DESARROLLO DINÁMICO DE APLICASIONES* . Pereira - Colombia.

Vivona, I. (2011). *Java.* Buenos Aires : Fox Andina.

Wikipedia, J. (30 de diciembre de 2017). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Hibernate

Yeager, N. y. (1996). *Web Server Technology [Tecnología de Servidor Web]*. San Francisco, US: Morgan Kaufmann Publishers, Inc.

ANEXOS

ANEXO A: LISTA DE ANOTACIONES

TIPO DE ANOTACIÓN	DESCRIPCIÓN
Action	Asocia una acción a una propiedad o referencia en la vista.
Actions	Un grupo de asociados al mismo miembro.@Actions
AddAction	Le permite definir su acción personalizada para comenzar a agregar nuevos elementos a una colección eligiendo entre los existentes.
AddActions	Un grupo de asociados a la misma colección. @AddAction
AsEmbedded	Hace que el comportamiento en la vista para una referencia (o colección) a una entidad sea como en el caso de un objeto incrustado (o una colección a entidades con CascadeType.REMOVE).
Calculation	Expresión para hacer el cálculo de una propiedad.
Collapsed	El miembro se mostrará colapsado para las vistas indicadas.
CollectionView	La vista del objeto referenciado (cada elemento de colección) que se usa para mostrar el detalle.
CollectionViews	Un grupo de asociados a la misma colección.@**CollectionView**
Condition	Restringe los elementos que aparecen en la colección.
DefaultValueCalculator	Para calcular su valor inicial.
Depends	Declara que una propiedad depende de otra (s).
DescriptionsList	Con @DescriptionsList usted puede indicar a OpenXava que visualice las referencias como una lista de descripciones (en realidad un combo).
DescriptionsLists	Un grupo de asociados al mismo miembro. @DescriptionsList

DetailAction	En la vista de colección para agregar acciones en modo de detalle, generalmente las acciones cuyo alcance es el detalle que se está editando.
DetailActions	Un grupo de asociados a la misma colección. @DetailAction
DisplaySize	El tamaño en caracteres del editor en la interfaz de usuario utilizado para mostrar esta propiedad.
DisplaySizes	Un grupo de asociados a la misma propiedad. @**DisplaySize**
EditAction	Le permite definir su acción personalizada para ver un elemento de colección.
EditActions	Un grupo de asociados a la misma colección. @EditAction
EditOnly	El usuario final puede modificar los elementos existentes en la colección, pero no agregar o eliminar elementos de la colección.
Editor	Nombre del editor que se usará para mostrar el miembro en esta vista.
Editors	Un grupo de asociados al mismo miembro. @**Editor**
EntityValidator	Este validador permite definir una validación a nivel de entidad.
EntityValidators	Un grupo de asociados a la misma entidad.@**EntityValidator**
Hidden	Una propiedad oculta tiene un significado para el desarrollador, pero no para el usuario.
HideDetailAction	En una colección, le permite definir su acción personalizada para ocultar la vista detallada.
HideDetailActions	Un grupo de asociados a la misma colección. @HideDetailAction
LabelFormat	Formato para mostrar la etiqueta de esta propiedad o referencia (mostrada como lista de descripciones).
LabelFormats	Un grupo de asociados al mismo miembro. @LabelFormat

LabelStyle	Estilo para mostrar la etiqueta de esta propiedad o referencia (mostrada como lista de descripciones).
LabelStyles	Un grupo de asociados al mismo miembro.@LabelStyle
ListAction	En colecciones para agregar acciones en modo lista; Usualmente las acciones cuyo alcance es la colección completa.
ListActions	Un grupo de asociados a la misma colección. @ListAction
ListProperties	Propiedades a mostrar en la lista para la visualización de una colección.
ListsProperties	Un grupo de asociados a la misma colección. @ListProperties
ListSubcontroller	Permite definir un subcontrolador en una colección.
ListSubcontrollers	Un grupo de asociados a la misma colección. @ListSubcontroller
NewAction	Le permite definir su acción personalizada para comenzar a crear un nuevo elemento en una colección.
NewActions	Un grupo de asociados a la misma colección. @NewAction
NoCreate	El usuario final no puede crear nuevos objetos del tipo referenciado desde aquí.
NoFrame	La referencia no se muestra dentro de un marco.
NoModify	El usuario final no puede modificar el objeto de referencia actual desde aquí.
NoSearch	El usuario no tendrá un enlace para realizar búsquedas con una lista, filtros, etc.
OnChange	Acción a ejecutar cuando el valor de esta propiedad / referencia cambia.
OnChanges	Un grupo de asociados al mismo miembro. @OnChange
OnChangeSearch	Acción a ejecutar para realizar la búsqueda de una referencia cuando el usuario ingresa el valor clave.
OnChangeSearchs	Un grupo de asociados al mismo miembro. @OnChangeSearch

OnSelectElementAction	Permite definir una acción que se ejecutará cuando un elemento de la colección esté seleccionado o no seleccionado.
OnSelectElementActions	Un grupo de asociados a la misma colección. @OnSelectElementAction
PostCreate	Devolución de llamada ejecutada antes de crear un objeto.
PreCreate	Devolución de llamada ejecutada antes de crear un objeto.
PreDelete	Devolución de llamada ejecutada antes de crear un objeto.
PropertyValidator	El validador ejecuta la lógica de validación en el valor asignado a la propiedad justo antes del almacenamiento.
PropertyValidators	Un grupo de asociados a la misma propiedad. @**PropertyValidator**
PropertyValue	Encapsular el valor para inyectarlo en una propiedad.
ReadOnly	El usuario nunca será editable por el usuario final en las vistas indicadas.
ReferenceView	Vista del objeto referenciado usado para mostrarlo en una referencia.
ReferenceViews	Un grupo de asociados a la misma referencia. @**ReferenceView**
RemoveAction	Permite definir una acción personalizada para eliminar el elemento de la colección.
RemoveActions	Un grupo de asociados a la misma colección. @**RemoveAction**
RemoveSelectedAction	Permite definir una acción personalizada para eliminar los elementos seleccionados de la colección.
RemoveSelectedActions	Un grupo de asociados a la misma colección.@**RemoveSelectedAction**
RemoveValidator	El @RemoveValidatores un validador de modelo de nivel, se ejecuta justo antes de la eliminación de un objeto, y tiene la posibilidad de negar la eliminación.

RemoveValidators	Un grupo de asociados a la misma entidad.@**RemoveValidator**
Required	Indica si esta propiedad o referencia es requerida.
RowAction	En colecciones para agregar una acción en cada fila, pero no en la barra de botones de colección.
RowActions	Un grupo de asociados a la misma colección.@RowAction
RowStyle	Para indicar el estilo de fila para **Tab**s y colecciones.
RowStyles	Un grupo de asociados a la misma colección. @RowStyle
SaveAction	Permite definir una acción personalizada para guardar el elemento de colección.
SaveActions	Un grupo de asociados a la misma colección. @SaveAction
SearchAction	Le permite especificar su propia acción para la búsqueda.
SearchActions	Un grupo de asociados a la misma referencia. @SearchAction
SearchKey	El usuario utiliza una propiedad o referencia de clave de búsqueda para buscar.
SearchListCondition	Define una condición que se utilizará al mostrar la lista de elementos seleccionables para agregar elementos a una colección o asignar un valor a una referencia.
SearchListConditions	Un grupo de asociados a la misma referencia o colección.@**SearchListCondition**
Stereotype	Un estereotipo es la forma de determinar un comportamiento específico de un tipo.
Tab	Defina el comportamiento para la presentación de datos de **pestañas** (modo de lista).
Tabs	Un grupo de asociados a la misma entidad.@Tab
Tree	Con @Tree puedes instruir para que visualice las colecciones como un árbol en lugar de una lista.
Trees	Un grupo de asociados a la misma colección.@Tree
View	Define con precisión el formato de la interfaz de usuario o vista.

ViewAction	Le permite definir su acción personalizada para ver un elemento de colección.
ViewActions	Un grupo de asociados a la misma colección.@**ViewAction**
Views	Un grupo de asociados a la misma entidad.@View
XOrderBy	La versión extendida de @OrderBy.

ANEXO B: GUIA DE OBSERVACIÓN DE LAS HERRAMIENTAS FRAMEWORK

GUIA DE OBSERVACION DE LAS HERRAMIENTAS FRAMEWORK					
Observador					
Framework					
Fecha					
Criterios		ALTO	MEDIO	BAJO	NULO
1	Grado de madurez				
2	Tamaño y grado de actividad de la comunidad				
3	Disponibilidad de librerías y aplicaciones de terceros				
4	Estructura predefinida de la aplicación				
5	Dificultad de la curva de aprendizaje				
6	Compatibilidad con el resto del ecosistema				
7	Rendimiento y escalabilidad				
8	Uso de patrones de diseño				
9	Trabajo en equipo				
10	Garantizar funciones de seguridad				
11	Uso de Licencias				

ANEXO C: FICHA DE ANALISIS DOCUMENTAL DE HERRAMIENTAS FRAMEWORK.

FICHA DE ANÁLISIS DOCUMENTAL DE HERRAMIENTAS FRAMEWORK		
DATOS DEL ANÁLISIS		
Numero de análisis		
Fecha		
Elaborado por		
Tiempo utilizado		
DATOS DESCRIPTIVOS		
Título de la publicación		
Fecha de publicación		
Título del proyecto		
Autor/es		
Palabras clave		
Lugar de creación		
ASPECTOS RELEVANTES		
Descripción general	Objetivo	
	Justificación	
Fundamentación teórica	Conceptos de sistematización	
	Enfoque	
Metodología del proceso de sistematización	Fases	
	Actividades	
	Recursos	
	Mecanismos para generar de fuentes de información	
	Análisis e interpretación	

Conclusión final	
Referentes base	
Observaciones	

ANEXO D: OPERACIONALIZACIÓN DE VARIABLES

VARIABLE	INDICADOR	ITEM
Framework de código abierto en Java	Plain Old Java Object (POJOs)	¿Cómo se define una clase Plain Old Java Object (POJOs)?
		¿Cómo describir una clase Plain Old Java Object (POJOs)?
		¿Cómo permite el acceso a las propiedades mediante los métodos getter y setter?
		¿Plain Old Java Object (POJOs) genera código repetitivo?
	Anotaciones	¿Cómo añade metadatos al código fuente Java?
		¿Qué grado de accesibilidad tiene con el programador?
		¿Las anotaciones toman la forma de una declaración de interfaz?
		¿Cómo permiten al programador definir el comportamiento de un software?
	Java Persistence API (JPA)	¿Cómo interactúa con una base de datos?
		¿utiliza los metadatos de tipo objeto relacional?
		¿Cómo probar las funciones en las aplicaciones?
		¿Cómo gestiona la persistencia de los datos?
Aplicaciones web empresariales basadas en	Funcionalidad del componente negocio	¿Qué lógica de negocio es necesaria con las funcionalidades de una aplicación?
		¿Existe las transacciones, persistencia, seguridad y concurrencia generales de las aplicaciones empresariales?
		¿Qué grado de beneficio alcanza para los sistemas construidos a base de componentes de negocio?
		¿Un componente de negocio es reutilizable?

componentes de negocios para entornos agiles.	Interfaz amigable	¿Qué grado de usabilidad debe tener una interfaz para cumplir y ayudar al usuario?
		¿Cómo representar la información y las acciones disponibles en la interfaz?
		¿Cómo Ajax contribuye a generar interfaces interactivas?
		¿Cómo reutilizar los elementos del diseño de la interfaz?
	Procesamiento de los datos	¿Qué manera es segura de almacenar los datos?
		¿Cómo asegurar que los datos suministrados sean limpios, correctos y útiles?
		¿Cómo organizar, analizar, interpretar y presentar los datos?