UNIVERSIDAD NACIONAL DE CHIMBORAZO
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA EN SISTEMAS Y COMPUTACIÓN

"Trabajo de grado previo a la obtención del Título de Ingeniero en Sistemas y Computación."

TRABAJO DE GRADUACIÓN

Título del Proyecto

ANÁLISIS DE LA TECNOLOGÍA ORM-HIBERNATE CON RESPECTO A LA PRODUCTIVIDAD, APLICADO AL SISTEMA DE GESTIÓN DE DOCUMENTACIÓN DEL DEPARTAMENTO DE PROCURADURÍA DE LA UNACH.

AUTOR:

Tixi Cuzco Ismael Franklin

Director: Ing. Diego Palacios

Riobamba – Ecuador

2016

Los miembros del Tribunal de Graduación del proyecto de investigación de título: **"ANÁLISIS DE LA TECNOLOGÍA ORM-HIBERNATE CON RESPECTO A LA PRODUCTIVIDAD, APLICADO AL SISTEMA DE GESTIÓN DE DOCUMENTACIÓN DEL DEPARTAMENTO DE PROCURADURÍA DE LA UNACH."** presentado por: Ismael Franklin Tixi Cuzco, dirigido por el Ing. Diego Palacios.

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

Para constancia de lo expuesto firman:

Ing. Danny Velasco ----------
Presidente del Tribunal Firma

Ing. Diego Palacios ----------
Director del Proyecto Firma

Ing. Samuel Moreno ----------
Miembro del Tribunal Firma

AUTORÍA DE LA INVESTIGACIÓN

Yo, **Ismael Franklin Tixi Cuzco** Autor de la Tesis "Análisis de la tecnología ORM-Hibernate con respecto a la productividad, aplicado al Sistema de Gestión de Documentación del Departamento de Procuraduría de la Unach", reconocemos y aceptamos el derecho de la Universidad Nacional de Chimborazo, de publicar este trabajo por cualquier medio conocido o por conocer, al ser este requisito para la obtención de Mi título de Ingeniero en Sistemas. El uso que la Universidad Nacional de Chimborazo hiciere de este trabajo, no implicara afección alguna de nuestros derechos morales o patrimoniales como autores.

Riobamba, 10 de Marzo del 2016

Ismael Franklin Tixi Cuzco

C.I. 060396594-8

AGRADECIMIENTO

En el presente trabajo de investigación quiero agradecer a Dios por brindarme la salud, la vida y por permitir hacer realidad mi anhelado sueño.

Expreso mi agradecimiento a la **Universidad Nacional de Chimborazo**, la cual me abrió sus puertas para culminar con éxito una etapa más de mi vida, preparándome para un futuro competitivo y poder servir a la sociedad con mi sólido conocimiento para el progreso del país.

Un reconocimiento especial a mi Director de Tesis, **Ing. Diego Palacios** por su calidad humana y todo el apoyo brindado al instruirme y guiarme a realizar el presente trabajo investigativo.

A los docentes de la Escuela de Ingeniería en Sistema y Computación porque todos han aportado con sus conocimientos y consejos para mi formación profesional.

Son muchas las personas que han formado parte de mi vida a las que me encantaría agradecerles su amistad, consejos, apoyo, y ánimo.

Para ellos muchas gracias y que Dios les bendiga

Ismael Tixi

DEDICATORIA

Doy gracias a Dios, por estar conmigo en cada paso que doy, por guiarme e iluminar mi mente y permitirme cumplir una meta más en mi vida. A mi padre Jorge Tixi y mi madre María Elena Cuzco por bridarme incondicionalmente su amor, consejos, ánimos para alcanzar mi más grandioso sueño el ser un profesional. A mis hermanos, hermana y sobrinos. Y a todas las personas que han sido mi soporte y compañía durante toda mi formación académica.

Ismael Tixi

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

RESUMEN

El presente trabajo de Tesis de Grado tiene como objetivo investigar la tecnología ORM-Hibernate con respectó a la productividad enfocado al acceso de datos y exponer sus principales conceptos, características, servicios y prestaciones para el desarrollo del Sistema de Gestión de Documentación para el Departamento de Procuraduría General de la Universidad Nacional de Chimborazo.

La presente investigación se compone de IV capítulos, en el Capítulo I inicia con un marco referencial, seguido de los objetivos y la debida justificación de esta investigación.

En el Capítulo II se sustenta teóricamente el presente trabajo con toda la información necesaria y complementaria donde se muestran los principales conceptos y características de la tecnología ORM-Hibernate y la arquitectura de desarrollo N-Capas.

En el capítulo III desarrolla el análisis comparativo de la arquitectura de desarrollo N-Capas y la tecnología ORM-Hibernate con respecto a la productividad enfocado al acceso de datos y las encuestas realizadas a expertos en el desarrollo de aplicaciones web para cumplir con los objetivos de la investigación.

En el capítulo IV se desarrolla el Sistema de Gestión de Documentación aplicando la tecnología ORM-Hibernate, empleando la metodología de desarrollo XP en sus fases de Planificación, Diseño, Codificación y Pruebas.

Se desarrolló el Sistema de Gestión de Documentación para el Departamento de Procuraduría General de la UNACH, donde la información sobre los procesos de los oficios, convenios, contratos y juicios tuvo una organización adecuada y transparente. Además el desarrollo de esta aplicación es un proyecto de la Procuraduría General registrado en el sistema UPR de la Universidad Nacional de Chimborazo.

UNIVERSIDAD NACIONAL DE CHIMBORAZO
FACULTAD DE INGENIERÍA
CENTRO DE IDIOMAS

Lic. Byron Soria

09 de Marzo de 2016

SUMMARY

The main objective of this Thesis is to research technology-Hibernate ORM. With respect to productivity focused on data access and expose their main concepts, features, services and performance, for the development of Document Management System for the Department of Attorney General of the National University of Chimborazo.

This research has four chapters. Chapter I begins with a framework, followed by the objectives and proper justification of this study. In Chapter II is theoretically supports this work with all the necessary information and complementary where the main concepts and features of the ORM-Hibernate technology and development architecture N-tier shown.

Chapter III develops comparative analysis of architecture N-tier development and technology-Hibernate ORM concerning productivity focused data access and surveys of experts in the development of web applications to meet the objectives of the investigation.

Chapter IV Management System Documentation is developed by applying the Hibernate ORM-tech, development methodology using XP in phases of planning, design, coding and testing.

Management System Documentation for the Department of Attorney General developed UNACH where information about processes trades, agreements, contracts and trials had adequate and transparent organization. Furthermore, the development of this application is a project of the Attorney General registered in the UPR system of the National University of Chimborazo.

INTRODUCCIÓN

En la actualidad existen diferentes arquitecturas y tecnologías para el desarrollo de aplicaciones web dificultando a los desarrolladores elegir una metodología adecuada. La constante evolución de la tecnológica, el acceso a la información por medios informáticos inducen a escoger una metodología adecuada que permita resolver problemas.

Las principales metodologías para desarrollar aplicaciones web son: la arquitectura de desarrollo N-Capas y el Mapeo Objeto/Relacional. Para escoger una metodología de desarrollo adecuada debemos tener en cuenta distintos enfoques como la productividad, el mantenimiento y rendimiento. La productividad en un punto importante a la hora de elegir una metodología ya que las empresas, entidades e instituciones tratan optimizar sus recursos.

La metodología de desarrollo Mapeo Objeto/Relacional o que comúnmente se conoce como ORM dispone de diferentes framework para su manejo, estas son: Doctrine, Propel, Hibernate, Linq.

El presente trabajo de investigación se enfoca a estudiar la tecnología ORM utilizando el framework Hibernate con respecto a la productividad enfocado al acceso de datos. Además el análisis de sus funciones, componentes, servicios y prestaciones.

CAPÍTULO I

MARCO REFERENCIAL

1.1 TÍTULO DEL PROYECTO

ANÁLISIS DE LA TECNOLOGÍA ORM-HIBERNATE CON RESPECTO A LA PRODUCTIVIDAD, APLICADO AL SISTEMA DE GESTIÓN DE DOCUMENTACIÓN DEL DEPARTAMENTO DE PROCURADURÍA DE LA UNACH

1.2 PROBLEMATIZACIÓN

En la actualidad existen diferentes arquitecturas y tecnologías para el desarrollo de aplicaciones web, dificultando a los desarrolladores elegir la más adecuada para lograr este objetivo de manera productiva. La constante evolución tecnológica, el acceso a la información por medios informáticos inducen a escoger una sistemática adecuada que permita resolver problemas.

Las empresas, instituciones y entidades tratan de optimizar sus recursos y los desarrolladores deben prestar importante atención a la hora de escoger una tecnología adecuada; los programadores usan la arquitectura de desarrollo N-Capas porque es la forma tradicional para crear las aplicaciones web. Mediante esta arquitectura, para lograr acceder a los datos se debe crear las funciones de forma manual, lo que ocasiona una importante inversión de tiempo.

1.2.1 IDENTIFICACIÓN Y DESCRIPCIÓN DEL PROBLEMA

Los programadores que emplean la arquitectura de desarrollo N-Capas para construir aplicaciones web, invierten notablemente tiempo y líneas de código.

En la actualidad el Departamento de Procuraduría General de la Universidad Nacional de Chimborazo no cuenta con una aplicación web, los procesos son gestionados de forma manual o semi automatizada (Microsoft Excel).

La Procuraduría General recibe un sin número de oficios de los diferentes Departamentos dentro o fuera de la Institución, la elaboración de convenios y contratos que son producto de los procesos de contratación pública. Además los seguimientos de los procesos judiciales que se encuentran inmersos los empleados de la Universidad Nacional de Chimborazo.

1.2.2 ANÁLISIS CRÍTICO

ORM-Hibernate es una tecnología que tiende a crecer, situada entre la capa de base de datos y la capa de aplicación, que propone el mapeo objeto/relacional (ORM) y la herramienta de persistencia Hibernate, que permite facilitar las funciones de CRUD y el leguaje de consulta SGBD.

Permite reducir el código de las operaciones de persistencia y recuperación de los objetos, proporcionan interfaces más simples para el manejo de objetos a través de su propio lenguaje de consulta.

En el caso de la parte aplicativa este Sistema de Gestión de Documentación supone la automatización de los procesos de recepción y envió de los oficios que operan por medio de SessionFactory, Transacciones y Sesiones (funciones CRUD y lenguaje de consulta HQL).

Al registrar los datos en un formulario el sistema crea la transacción y la sesión para guardar la información en la capa de base de datos.

1.2.3 PROGNOSIS

La implementación de la tecnología ORM-Hibernate o la arquitectura de desarrollo N-Capas con respecto a la productividad enfocado al acceso de datos facilitaran el desarrollo del Sistema de Gestión de Documentación para el Departamento de Procuraduría General de la Universidad Nacional de Chimborazo.

La aplicación web permitirá visualizar la información y gestionar los procesos de oficios recibidos, oficios enviados, convenios y contratos realizados, además de los criterios jurídicos resueltos por los abogados del Departamento.

1.2.4 DELIMITACIÓN

El proyecto se lo enfocara en la realización del Sistema de Gestión de Documentación basada en la tecnología ORM-Hibernate para el Departamento de Procuraduría General de la Universidad Nacional de Chimborazo para la gestión de los procesos de oficios, convenios, contratos y juicios. Para el desarrollo de la aplicación web se utilizara software libre.

El Sistema de Gestión de Documentación va a contener los siguientes módulos:

- Módulo de gestión de oficios recibidos y enviados.
- Módulo de gestión de convenios.
- Módulo de gestión de contratos.
- Módulo de gestión de juicios.

1.2.5 FORMULACIÓN DEL PROBLEMA

¿Cómo incide el uso de la tecnología ORM-Hibernate en la mejora de la productividad con respecto al desarrollo, del Sistema de Gestión de Documentación del Departamento de Procuraduría de la UNACH.?

1.2.6 HIPÓTESIS

La tecnología ORM-Hibernate mejora el desarrollo de aplicaciones Java con respecto a la productividad.

1.2.7 IDENTIFICACIÓN DE VARIABLES

- **Variable Independiente**

ORM-Hibernate.

- **Variable Dependiente**

Productividad.

1.2.8 OPERACIONALIZACIÓN DE VARIABLES

Tabla 1 Operacionalización de variables

Variable	Tipo	Definición Conceptual	Dimensión	Indicadores
ORM-Hibernate	Independiente	Hibernate es una tecnología de alto rendimiento Objeto-Relacional de persistencia y consulta que está bajo la licencia de código abierto.	• Concurrencia • Extensibilidad • Alto rendimiento	• Número de usuarios conectados. • Número de métodos de levantar y configurar la interfaz de la aplicación desde la base de datos. • Números de conexiones realizadas por petición.
Productividad	Dependiente	La relación entre la cantidad de bienes y servicios producidos y la cantidad de recursos utilizados.	• Tiempo • Línea de código	• Número de hora de empleadas. • Número de líneas de código.

Fuente: Autor

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

- Analizar la tecnología ORM-Hibernate con respecto a la productividad, aplicado al Sistema de Gestión de Documentación del Departamento de Procuraduría de la UNACH.

1.3.2 OBJETIVOS ESPECÍFICOS

- Analizar las características de la tecnología ORM-Hibernate para el desarrollo de aplicaciones Java.
- Comparar la metodología tradicional de desarrollo de aplicaciones de N-Capas con Java versus la tecnología ORM-Hibernate.
- Desarrollar el Sistema de Gestión de Documentación del Departamento de Procuraduría de la UNACH.

1.4 JUSTIFICACIÓN

Definido anteriormente en la problematización, la presente investigación pretende analizar la tecnología ORM-Hibernate con respectó a la productividad enfocado al acceso de datos y exponer sus principales conceptos, características, servicios y prestaciones para el desarrollo del Sistema de Gestión de Documentación para el Departamento de Procuraduría General de la Universidad Nacional de Chimborazo.

El Sistema de Gestión de Documentación funciona como motor principal del campo de trabajo, por ello tomaremos este motor como pieza fundamental de la creación de una aplicación web. El Sistema de Gestión de Documentación va a contener los módulos de oficios, convenios, contratos y juicios.

CAPÍTULO II

FUNDAMENTACIÓN TEÓRICA

2.1 DEFINICIÓN DE PRODUCTIVIDAD

La definición de lo que se quiere medir es un elemento clave para cualquier medición dado que sin este elemento es imposible establecer una medida (Tangen, 2005). De forma inversa, es posible decir que una forma de conocer qué se quiere medir es obtener la definición de la medida (Sink, et al., 1984). A pesar de las diferentes definiciones de productividad, estas pueden ser clasificadas en tres grupos (Ghobadian & Husband, 1990): (1) tecnológicas: relación entre ratios de salidas y entradas utilizadas, (2) ingenieriles: la relación entre la salida actual y la potencial de un proceso, y (3) económicas: la eficiencia de la asignación de recursos. (Payá Martín, 2016).

La productividad en IS es comúnmente medida utilizando una medida tecnológica basada en un ratio entre el tamaño del producto desarrollado y el esfuerzo requerido para producirlo (MacCormack, Kemerer, Cusumano, & Crandall, 2003), por ejemplo las líneas de código por unidad de tiempo (SLOC/t) (Maxwell, et al., 1996) o alguna variante de puntos función por unidad de tiempo (PF/t) (Low & Jeffery, 1990). En esta línea, la norma IEEE 1045-1992 define la productividad como la relación de una primitiva de salida (líneas de código, puntos función o documentos) y su correspondiente primitiva de entrada (esfuerzo, tiempo) para desarrollar software. Por otro lado, la norma ISO 9126-4 define la productividad basándose en factores de calidad como la capacidad del producto software para permitir a los usuarios emplear cantidades de recursos adecuados en relación con la efectividad alcanzada en un contexto de uso específico. Esta norma está centrada en la calidad, el usuario final y el contexto de uso, por lo que la definición de la productividad gira en torno a estos tres conceptos. (Payá Martín, 2016)

Además, esta norma, en su Anexo F, presenta tres medidas de productividad relacionadas con los tipos de recursos empleados: Productividad Humana

(Efectividad/Esfuerzo), Productividad Temporal (Efectividad/Tiempo), y Productividad Económica (Efectividad/Coste). Las definiciones aportadas por ambas normas son complementarias, en cuanto a que la visión de la productividad es distinta (Cheikhi, Al-Qutaish, & Idri, 2012); la norma IEEEE 1045-1992 se centra en las salidas y las entradas, y la norma ISO 9126-4 en la calidad, el usuario final y el contexto de uso. (Payá Martín, 2016)

A nivel lingüístico, la Real Academia de la Lengua Española, define productividad como:

1. Cualidad de productivo.
2. Capacidad o grado de producción por unidad de trabajo, superficie de tierra cultivada, equipo industrial, etc.
3. Relación entre lo producido y los medios empleados, tales como la mano de obra, materiales, energía, etc.

(Payá Martín, 2016)

2.2 MAPEO OBJETO/RELACIONAL

2.2.1 INTRODUCCIÓN A ORM

El mapeo del dominio de objetos dentro de un modelo relacional es importante para el proceso de desarrollo del software moderno. Los lenguajes de programación, orientados objetos como Java, C# y C++ son los más comúnmente aplicados para el desarrollo de nuevos sistemas de software. Las bases de datos relacionales siguen el enfoque preferido para el almacenamiento de información persistente y es probable que siga siéndolo para un futuro. (Doroshenki y Romanenko, 2005)

El mapeo objeto/relacional permite formar el mapeo de una base de datos relacional, generando en la aplicación orientada a objetos clases que son directamente la estructura de la base de datos que se posee. Es decir, en la aplicación se puede tener una base de datos orientada a objetos virtuales sobre la base de datos relacional. Esta característica permite aplicar conceptos de orientación a objetos como herencia y polimorfismo, a los datos almacenados de forma relacional. (Doroshenki y Romanenko, 2005)

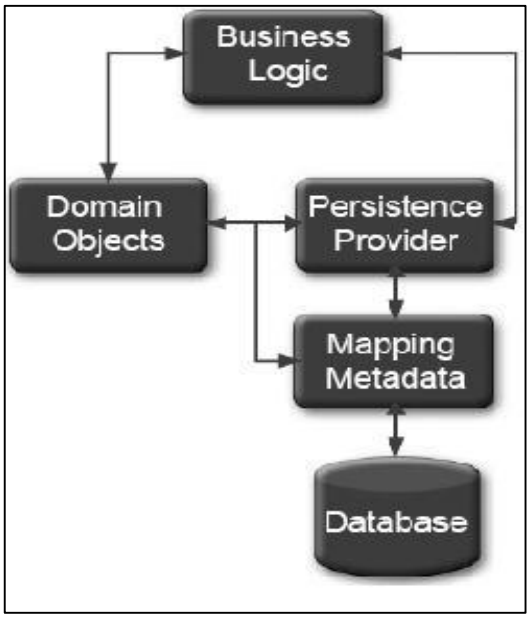


Ilustración 1 Modelo ORM
Fuente: (Russell, Meswani, White), 2007

La solución de problemas en la aplicación de mapeo objeto/relacional ha modificado la forma de desarrollar aplicaciones, porque los lenguajes orientados a objetos y bases de datos relacionales se han generalizado desde hace mucho tiempo y con frecuencia se usan juntos con el desarrollo de patrones de diseño de la metodología. (Doroshenki y Romanenko, 2005)

2.2.2 COMPONENTES DE ORM

Una solución ORM consta de los cuatros aspectos siguientes:

- Una API para realizar las operaciones básicas CRUD sobre objetos de clases persistentes.
- Un lenguaje o API para especificar consultas que hacen referencia a las clases y sus propiedades.
- Facilidad para especificar mapeo de metadatos.
- Una técnica para que la implementación del ORM pueda llevar a cabo búsquedas, asociaciones y otras funciones de optimización.

(Doroshenki y Romanenko, 2005)

Mediante el ORM, la aplicación interactúa con el ORM API y las clases del modelo de dominio y se abstrae de la subyacente SQL/JDBC. Dependiendo de las características o de la implementación particular.(Doroshenki y Romanenko, 2005).

2.2.3 JDBC: COMPONENTE INDISPENSABLE PARA LOS ORM

Sin tener en cuenta la solución de mapeo objeto/relacional que se vaya a utilizar para comunicarse con la base de datos relacional, todos ellos dependen de JDBC. Teniendo en cuenta que la mayor parte de las aplicaciones se comunican con bases de datos relacionales, es fundamental considerar cada uno de los niveles del software (desde el código del programa hasta la fuente de datos) para asegurar que el diseño de persistencia objeto/relacional sea óptimo. (Reyes Freire, 2016)

Tal y como se verá más adelante, cada una de las soluciones de mapeo objeto/relacional tiene una dependencia particular en el driver JDBC para poder comunicarse con la base de datos de una forma eficiente. Si el driver JDBC que va a participar en la comunicación no es óptimo, la posible gran eficiencia de cualquier Framework quedará debilitada. Por tanto, elegir el driver JDBC que mejor se adapte a la aplicación es esencial a la hora de construir un sistema eficiente en el que participe una solución de mapeo objeto/relacional. (Reyes Freire, 2016)

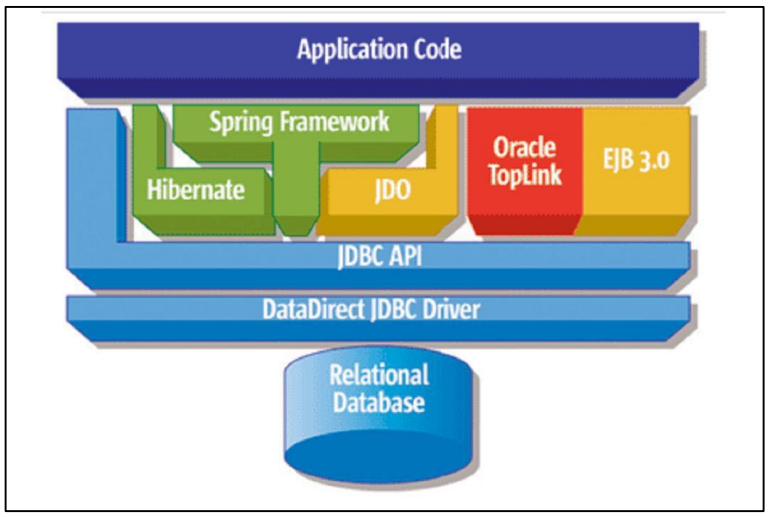


Ilustración 2 Interacción de Componentes

Fuente: (Carmen Gonzales, 2016)

La figura muestra una representación de los diferentes mecanismos o soluciones de mapeo objeto/relacional y cómo se relacionan con el código de la aplicación y con los recursos de datos relacionados. Esto muestra claramente la función crítica que desempeña el driver JDBC puesto que está situado en la base de cada uno de los Frameworks. (Reyes Freire, 2016)

La eficiencia del driver JDBC tiene importantes consecuencias en el comportamiento de las aplicaciones. Cada mecanismo de mapeo objeto/relacional es completamente dependiente del driver, sin tener en cuenta el diseño de la API del Framework que esté expuesta al código fuente de la aplicación. (Reyes Freire, 2016)

Como los mecanismos de mapeo objeto/relacional generan llamadas eficientes para acceder a la base de datos, mucha gente defiende que la importancia del driver JDBC se ha visto reducida. Sin embargo, como en cualquier arquitectura, la totalidad de eficiencia en una aplicación siempre estará afectada por el nivel más débil del sistema. (Reyes Freire, 2016)

Independientemente del código JDBC generado, los mecanismos de mapeo objeto/relacional son incapaces de controlar cómo los drivers interactúan con la base de datos. Entonces la eficiencia de la aplicación depende en gran parte de la habilidad que tenga el driver del nivel JDBC para mover todos los datos manejados entre la aplicación y la base de datos. (Reyes Freire, 2016)

Aunque hay múltiples factores que considerar a la hora de elegir un driver JDBC, seleccionar el mejor driver JDBC posible basándose en comportamiento, escalabilidad y fiabilidad es la clave para obtener el máximo beneficio de cualquier aplicación basada en un Framework de mapeo objeto/relacional. (Reyes Freire, 2016)

2.2.4 VENTAJAS DE ORM

Rapidez en el desarrollo: La mayoría de las herramientas actuales permiten la creación del modelo por medio del esquema de la base de datos, leyendo el esquema, nos crea el modelo adecuado. (Cando Cando, 2016)

Abstracción de la base de datos: Al utilizar un sistema ORM, lo que conseguidos es separarnos totalmente del sistema de Base de datos que utilicemos, y así si en un futuro debemos de cambiar de motor de bases de datos, tendremos la seguridad de que este cambio no nos afectará a nuestro sistema, siendo el cambio más sencillo. (Cando Cando, 2016)

Reutilización: Nos permite utilizar los métodos de un objeto de datos desde distintas zonas de la aplicación, incluso desde aplicaciones distintas. (Cando Cando, 2016)

Seguridad: Los ORM suelen implementar sistemas para evitar tipos de ataques como los SQL injections. (Cando Cando, 2016)

Mantenimiento del código: Nos facilita el mantenimiento del código debido a la correcta ordenación de la capa de datos, haciendo que el mantenimiento del código sea mucho más sencillo. (Cando Cando, 2016)

Lenguaje propio para realizar las consultas: Estos sistemas de mapeo traen su propio lenguaje para hacer las consultas, lo que hace que los usuarios dejen de utilizar las sentencias SQL para que pasen a utilizar el lenguaje propio de cada herramienta. (Cando Cando, 2016)

2.2.5 DESVENTAJAS DE ORM

Tiempo utilizado en el aprendizaje: Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto y ver todo el partido que se le puede sacar. (Cando Cando, 2016)

Aplicaciones algo más lentas: Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá de transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos. (Cando Cando, 2016)

2.2.6 TIPOS DE FRAMEWORK ORM

En la actualidad hay muchos tipos de Framework que nos devuelven el mapeo objeto relacional, según el lenguaje que estemos utilizando. Vamos a nombrar algunos de los más utilizados.

- Doctrine
- Propel
- Hibernate
- Linq

(Cando Cando, 2016).

2.3 FRAMEWORK HIBERNATE

2.3.1 INTRODUCCIÓN A HIBERNATE

Ilustración 3 Logo Hibernate
Fuente: (Hibernate, 2016)

Trabajar con software orientado a objetos y bases de datos relacionales puede ser engorroso y lento. Los costos de desarrollo son significativamente más altos debido a la falta de coincidencia entre el paradigma de cómo se representa los datos en los objetos frente a bases de datos relacionales. Hibernate es una solución de mapeo objeto/relacional para entornos Java. El término Mapeo Objeto/Relacional se refiere a la técnica de mapeo de datos a partir de una representación del modelo de objetos a un modelo de datos relacional de representación. (Hibernate, 2016)

Hibernate no sólo se encarga de la asignación de clases de Java a las tablas de base de datos (y de tipos de datos Java con tipos de datos SQL), sino que también proporciona consulta de datos e instalaciones de recuperación. Se puede reducir significativamente el tiempo de desarrollo empleado de otro modo con los datos de manuales de manejo en SQL y JDBC. Objetivo de diseño de Hibernate es aliviar el desarrollador el 95% de las tareas de programación relacionado con la persistencia

de datos común al eliminar la necesidad de manual, el procesamiento de datos artesanal utilizando SQL y JDBC. (Hibernate, 2016)

2.3.2 ARQUITECTURA INTEGRAL DE HIBERNATE

La arquitectura "completa" abstrae la aplicación de las APIs de JDBC/JTA y permite que Hibernate se encargue de los detalles. (Hibernate, 2016)

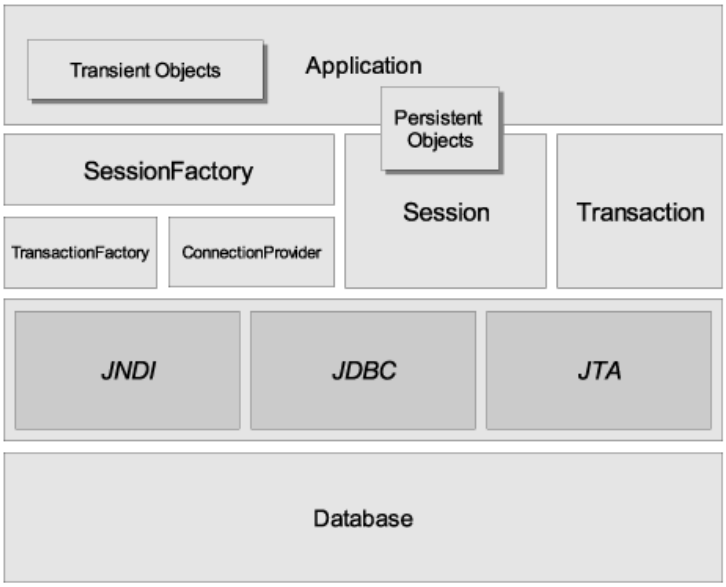


Ilustración 4 Arquitectura Integral
Fuente: (Hibernate, 2016)

SessionFactory (org.hibernate.SessionFactory)

Un caché threadsafe (inmutable) de mapeos compilados para una sola base de datos. Una fábrica de Session y un ConnectionProvider, SessionFactory puede mantener un caché opcional (de segundo nivel) de datos reusables entre transacciones a nivel de proceso o de clúster. (Hibernate, 2016)

Session (org.hibernate.Session)

Un objeto mono-hebra, de corta vida que representa una conversación entre la aplicación y el almacenamiento persistente. Envuelve una conexión JDBC y es una fábrica de Transaction. Session mantiene un caché requerido de primer nivel de objetos persistentes, que se utiliza cuando se navega el gráfico de objetos o mientras se buscan objetos por identificador. (Hibernate, 2016)

Objetos y colecciones persistentes

Objetos de corta vida, mono-hebra contienen un estado persistente así como una funcionalidad empresarial. Estos pueden ser JavaBeans/POJOs normales. Estos se encuentran asociados con exactamente una Session. Tan pronto como la Session se cierre, serán separados y estarán libres para utilizarlos en cualquier capa de aplicación. (Hibernate, 2016)

Objetos y colecciones transitorias y separadas

Instancias de clases persistentes que no se encuentran actualmente asociadas con una Session. Pueden haber sido instanciadas por la aplicación y aún no haber sido persistidas, o pueden haber sido instanciadas por una Session cerrada. (Hibernate, 2016)

Transaction (org.hibernate.Transaction)

Un objeto de corta vida, mono-hebra que la aplicación utiliza para especificar unidades atómicas de trabajo. Abstrae la aplicación de las transacciones subyacentes JDBC, JTA o CORBA. En algunos casos, una Session puede extenderse sobre varias Transactiones. Sin embargo, la demarcación de la transacción, ya sea utilizando la API subyacente o Transaction, nunca es opcional. (Hibernate, 2016)

ConnectionProvider (org.hibernate.connection.ConnectionProvider)

Una fábrica y pool de conexiones JDBC. Abstrae a la aplicación del Datasource o DriverManager subyacente. No se expone a la aplicación, pero puede ser extendido/implementado por el desarrollador. (Hibernate, 2016)

TransactionFactory (org.Hibernate.TransactionFactory)

Una fábrica de instancias de Transaction. No se expone a la aplicación pero puede ser extendido/implementado por el desarrollador. (Hibernate, 2016)

Extensión Interfaces

Hibernate ofrece un rango de interfaces de extensión opcionales que puede implementar para personalizar el comportamiento de su capa de persistencia. (Hibernate, 2016)

Dada una arquitectura "sencilla", la aplicación evita las APIs de Transaction/TransactionFactory y/o ConnectionProvider, para comunicarse directamente con JTA o JDBC. (Hibernate, 2016)

2.3.2.1 ESTADOS DE INSTANCIA

Una instancia de una clase persistente puede estar en uno de tres estados diferentes. Estos estados se definen con respecto a su contexto de persistencia. El objeto Session de Hibernate es el contexto de persistencia. Los tres estados diferentes son los siguientes (Hibernate, 2016)

Transitorio

La instancia no está asociada con un contexto de persistencia. No tiene identidad persistente o valor de clave principal. (Hibernate, 2016)

Persistente

La instancia se encuentra actualmente asociada con un contexto de persistencia. Tiene una identidad persistente (valor de clave principal) y puede tener una fila correspondiente en la base de datos. Para un contexto de persistencia en particular, Hibernate garantiza que la identidad persistente es equivalente a la identidad Java en relación con la ubicación del objeto. (Hibernate, 2016)

Separado

La instancia estuvo alguna vez asociada con un contexto de persistencia, pero ese contexto se cerró, o la instancia fue serializada a otro proceso. Tiene una identidad persistente y puede tener una fila correspondiente en la base de datos. Para las instancias separadas, Hibernate no establece ninguna garantía sobre la relación entre identidad persistente e identidad Java. (Hibernate, 2016)

2.3.3 CONFIGURACIÓN DE HIBERNATE

Hibernate está diseñado para operar en muchos entornos diferentes, hay una amplia gama de parámetros de configuración. Afortunadamente, la mayoría tienen valores predeterminados sensibles e Hibernate se distribuye con un ejemplo hibernate.properties archivo, que muestra las distintas opciones. En pocas palabras el archivo de ejemplo en su ruta de clase y personalizarlo para que se adapte a sus necesidades. (Hibernate, 2016)

2.3.3.1 CONFIGURACIÓN PROGRAMÁTICA

Una instancia de org.hibernate.cfg.Configuration representa un conjunto entero de mapeos de los tipos Java de una aplicación a una base de datos SQL. La org.hibernate.cfg.Configuration se utiliza para construir una org.hibernate.SessionFactory inmutable. Los mapeos se compilan desde varios archivos de mapeo XML. (Hibernate, 2016)

Puede obtener una instancia de org.hibernate.cfg.Configuration instanciándola directamente y especificando los documentos de mapeo XML. Si los archivos de mapeo están en la ruta de clase, utilice addResource ( ). Por ejemplo (Hibernate, 2016)

```
Configuration cfg = new Configuration()
    .addResource("Item.hbm.xml")
    .addResource ("Bid.hbm.xml");
```

Ilustración 5 Configuración XML
Fuente: (Hibernate, 2016)

2.3.3.2 OBTENCIÓN DE UNA SESSIONFACTORY

Cuando la org.hibernate.cfg.Configuration ha analizado sintácticamente todos los mapeos, la aplicación tiene que obtener una fábrica para las instancias org.hibernate.Session. Esta fábrica está concebida para que todos los hilos de la aplicación la compartan (Hibernate, 2016)

```
SessionFactory sessions = cfg.buildSessionFactory ();
```

Ilustración 6 SessionFactory
Fuente: (Hibernate, 2016)

Hibernate permite que su aplicación instancie más de una org.hibernate.SessionFactory. Esto es útil si está utilizando más de una base de datos (Hibernate, 2016)

2.3.3.3 CONEXIONES JDBC

Se aconseja que la org.hibernate.SessionFactory cree y almacene en pool conexiones JDBC. Si adopta este enfoque, el abrir una org.hibernate.Session es tan simple como (Hibernate, 2016)

```
Session session = sessions.openSession (); // open a new Session
```

Ilustración 7 Session
Fuente: (Hibernate, 2016)

En el momento en que inicie una tarea que requiera acceso a la base de datos, se obtendrá una conexión JDBC del pool. (Hibernate, 2016)

Para que esto funcione, primero necesita pasar algunas las propiedades de conexión JDBC a Hibernate. Todos los nombres de las propiedades de Hibernate y su semántica están definidas en la clase org.hibernate.cfg.Environment. Ahora describiremos las configuraciones más importantes para la conexión JDBC. (Hibernate, 2016)

Hibernate obtendrá y tendrá en pool las conexiones utilizando java.sql.DriverManager si configura las siguientes propiedades (Hibernate, 2016)

Tabla 2 Propiedades JDBC de Hibernate

NOMBRE DE LA PROPIEDAD	PROPÓSITO
hibernate.connection.driver_class	JDBC driver class
hibernate.connection.url	JDBC URL
hibernate.connection.username	database user
hibernate.connection.password	database user password
hibernate.connection.pool_size	maximum number of pooled connections

Fuente: (Hibernate, 2016)

Sin embargo, el algoritmo de pooling de la conexión propia de Hibernate es algo rudimentario. Está concebido para ayudarle a comenzar y no para utilizarse en un sistema de producción ni siquiera para pruebas de rendimiento. Para alcanzar un mejor rendimiento y estabilidad debe utilizar un pool de terceros. Sólo remplace la

propiedad hibernate.connection.pool_size con configuraciones específicas del pool de conexiones. Esto desactivará el pool interno de Hibernate. Por ejemplo, es posible utilizar C3P0. (Hibernate, 2016)

C3P0 es un pool de conexiones JDBC de código abierto distribuido junto con Hibernate en el directorio lib. Hibernate utilizará su org.hibernate.connection.C3P0ConnectionProvider para pooling de conexiones si establece propiedades hibernate.c3p0.*. Si quiere utilizar Proxool refiérase a hibernate.properties incluido en el paquete y al sitio web de Hibernate para obtener más información. (Hibernate, 2016)

```
hibernate.connection.driver_class = org.postgresql.Driver
hibernate.connection.url = jdbc: postgresql: //localhost/mydatabase
hibernate.connection.username = myuser
hibernate.connection.password = secret
hibernate.c3p0.min_size=5
hibernate.c3p0.max_size=20
hibernate.c3p0.timeout=1800
hibernate.c3p0.max_statements=50
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

Ilustración 8 Conexión JDBC
Fuente: (Hibernate, 2016)

Para su utilización dentro de un servidor de aplicaciones, casi siempre usted debe configurar Hibernate para obtener conexiones de un javax.sql.Datasource del servidor de aplicaciones registrado en JNDI. Necesitará establecer al menos una de las siguientes propiedades. (Hibernate, 2016)

Tabla 3 Propiedades de la Fuente de Datos de Hibernate

NOMBRE DE LA PROPIEDAD	PROPÓSITO
hibernate.connection.datasource	datasource JNDI name
hibernate.jndi.url	URL del proveedor JNDI (opcional)
hibernate.jndi.class	clase del JNDI InitialContextFactory (opcional)
hibernate.connection.username	usuario de la base de datos (opcional)
hibernate.connection.password	contraseña del usuario de la base de datos (opcional)

Fuente: (Hibernate, 2016)

He aquí un archivo hibernate.properties de ejemplo para una fuente de datos JNDI provisto por un servidor de aplicaciones. (Hibernate, 2016)

2.3.3.4 ARCHIVO DE CONFIGURACIÓN XML

Un enfoque alternativo de configuración es especificar una configuración completa en un archivo llamado hibernate.cfg.xml. Este archivo se puede utilizar como un remplazo del archivo hibernate.properties o en el caso de que ambos se encuentren presentes, para sobrescribir propiedades. (Hibernate, 2016)

El archivo de configuración XML por defecto se espera en la raíz de su CLASSPATH.

```
<?xml version='1.0' encoding='utf-8'?>
<! DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <!-- a SessionFactory instance listed as /jndi/name -->
   <session-factory
      name="java:hibernate/SessionFactory">

      <!-- properties -->
      <property name="connection.datasource"
>java:/comp/env/jdbc/MyDB</property>
      <property name="dialect"
>org.hibernate.dialect.MySQLDialect</property>
      <property name="show_sql"
>false</property>
      <property name="transaction.factory_class">
         org.hibernate.transaction.JTATransactionFactory
      </property>
      <property name="jta.UserTransaction"
>java:comp/UserTransaction</property>

      <!-- mapping files -->
      <mapping resource="org/hibernate/auction/Item.hbm.xml"/>
      <mapping resource="org/hibernate/auction/Bid.hbm.xml"/>

      <!-- cache settings -->
      <class-cache class="org.hibernate.auction.Item" usage="read-write"/>
      <class-cache class="org.hibernate.auction.Bid" usage="read-only"/>
         <collection-cache collection="org.hibernate.auction.Item.bids" usage="read-
write"/>

   </session-factory>
</hibernate-configuration
>
```

Ilustración 9 Archivo de configuración XML
Fuente: (Hibernate, 2016)

La ventaja de este enfoque es la externalización de los nombres de los archivos de mapeo a la configuración. El hibernate.cfg.xml también es más práctico una vez que haya afinado el caché de Hibernate. (Hibernate, 2016)

Puede escoger ya sea hibernate.properties o hibernate.cfg.xml. Ambos son equivalentes, excepto por los beneficios de utilizar la sintaxis XML que mencionados anteriormente. (Hibernate, 2016)

2.3.4 CLASES PERSISTENTES

Las clases persistentes son clases en una aplicación que implementan las entidades del problema empresarial (por ejemplo, Customer y Order en una aplicación de comercio electrónico). No se considera que todas las instancias de una clase persistente estén en estado persistente. Por ejemplo, una instancia puede ser transitoria o separada. (Hibernate, 2016)

Hibernate funciona mejor si estas clases siguen algunas reglas simples, también conocidas como el modelo de programación POJO (Plain Old Java Object). Sin embargo, ninguna de estas reglas son requerimientos rígidos. De hecho, Hibernate asume muy poco acerca de la naturaleza de sus objetos persistentes. Puede expresar un modelo de dominio en otras formas (por ejemplo, utilizando árboles de instancias de Map). (Hibernate, 2016)

2.3.4.1 EJEMPLO SIMPLE DE POJO

La mayoría de aplicaciones Java requieren una clase persistente que represente a los felinos. Por ejemplo:

```
package eg;
import java.util.Set;
import java.util.Date;

public class Cat {
    private Long id; // identifier

    private Date birthdate;
    private Color color;
    private char sex;
    private float weight;
    private int litterId;

    private Cat mother;
    private Set kittens = new HashSet();

    private void setId(Long id) {
        this.id=id;
    }
    public Long getId() {
        return id;
    }

    void setBirthdate(Date date) {
        birthdate = date;
    }
    public Date getBirthdate() {
        return birthdate;
    }

    void setWeight(float weight) {
        this.weight = weight;
    }
    public float getWeight() {
        return weight;
    }

    public Color getColor() {
        return color;
    }
    void setColor(Color color) {
        this.color = color;
    }

    void setSex(char sex) {
        this.sex=sex;
    }
    public char getSex() {
        return sex;
    }

    void setLitterId(int id) {
        this.litterId = id;
    }
    public int getLitterId() {
        return litterId;
    }

    void setMother(Cat mother) {
        this.mother = mother;
    }
    public Cat getMother() {
        return mother;
    }
    void setKittens(Set kittens) {
        this.kittens = kittens;
    }
    public Set getKittens() {
        return kittens;
    }

    // addKitten not needed by Hibernate
    public void addKitten(Cat kitten) {
            kitten.setMother(this);
        kitten.setLitterId( kittens.size() );
        kittens.add(kitten);
    }
}
```

Ilustración 10 Ejemplo Pojo
Fuente: (Hibernate, 2016)

2.3.5 MAPEO BÁSICO O/R

2.3.5.1 DECLARACIÓN DE MAPEO

Los mapeos objeto/relacional usualmente se definen en un documento XML. El documento de mapeo está diseñado para que se pueda leer y editar a mano. El lenguaje de mapeo está centrado en Java, lo que significa que los mapeos se construyen alrededor de declaraciones de clases persistentes y no alrededor de declaraciones de tablas. (Hibernate, 2016)

Ejemplo de mapeo:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="eg">

        <class name="Cat"
            table="cats"
            discriminator-value="C">

                <id name="id">
                        <generator class="native"/>
                </id>

                <discriminator column="subclass"
                     type="character"/>

                <property name="weight"/>

                <property name="birthdate"
                    type="date"
                    not-null="true"
                    update="false"/>

                <property name="color"
                    type="eg.types.ColorUserType"
                    not-null="true"
                    update="false"/>

                <property name="sex"
                    not-null="true"
                    update="false"/>

                <property name="litterId"
                    column="litterId"
                    update="false"/>

                <many-to-one name="mother"
                    column="mother_id"
                    update="false"/>

                <set name="kittens"
                    inverse="true"
                    order-by="litter_id">
                        <key column="mother_id"/>
                        <one-to-many class="Cat"/>
                </set>

                <subclass name="DomesticCat"
                    discriminator-value="D">

                        <property name="name"
                            type="string"/>

                </subclass>

        </class>

        <class name="Dog">
                <!-- mapping for Dog could go here -->
        </class>

</hibernate-mapping>
```

Ilustración 11 Declaración de mapeo
Fuente: (Hibernate, 2016)

Ahora vamos a discutir el contenido del documento de mapeo. Solamente describiremos los elementos y atributos del documento que Hibernate utiliza en tiempo de ejecución. El documento de mapeo también comprende algunos atributos y elementos extra opcionales que afectan los esquemas de la base de datos exportados por la herramienta de exportación de esquemas. (Hibernate, 2016)

2.3.6 TIPOS DE DATOS

Como solución de mapeo objeto/relacional, Hibernate trata tanto las representaciones Java y JDBC de datos de la aplicación. Una aplicación catálogo en línea, por ejemplo, tiene muy probablemente del producto objeto con un número de atributos tales como un SKU, nombre, etc. Para estos atributos individuales, Hibernate debe ser capaz de leer los valores de la base de datos y escribir de nuevo. Este 'marshalling' es la función de un tipo de hibernación, que es una implementación de la org.hibernate.type.Typeinterfaz. (Hibernate, 2016)

2.3.6.1 LOS TIPOS DE VALOR BÁSICO

Tabla 4 Tipo de valor básico

TIPO HIBERNATE	TIPO JAVA	TIPO BASE DE DATOS (MYSQL)
Integer	int , java.lang.Integer	INTEGER
Long	long, java.lang.Long	BIGINT
Short	short , java.lang.Short	SMALLINT
Float	float , java.lang.Float	FLOAT
Doublé	double , java.lang.Double	DOUBLE
character	char , java.lang.Character	CHAR
byte	byte , java.lang.Byte	TINYINT
boolean	boolean , java.lang.Boolean	TINYINT. Guarda el true como ”1” y el false como ”0”
yes_no	boolean , java.lang.Boolean	CHAR (1). Guarda el true como ”Y” y el false como ”N”
true_false	boolean , java.lang.Boolean	CHAR (1). Guarda el true como ”T” y el false como ”F”
string	java.lang.String	VARCHAR
date	java.util.Date	DATE. Solo se guarda solo la información de la fecha (año, mes y día)
time	java.util.Date	TIME. Solo se guarda solo la información de la hora (horas, minutos y segundos)
timestamp	java.util.Date	DATETIME. Se guarda la información de la fecha y la hora (año, mes , día, horas, minutos y segundos)
binary	byte[]	TINYBLOB
big_decimal	java.math.BigDecimal	DECIMAL

Fuente: Autor

2.3.7 MAPEOS DE COLECCIÓN

2.3.7.1 COLECCIONES PERSISTENTES

Naturalmente Hibernate también permite colecciones persistentes. Estas colecciones persistentes pueden contener casi cualquier otro tipo de Hibernate, incluyendo: tipos básicos, tipos personalizados, componentes y referencias a otras entidades. La distinción entre el valor de referencia y la semántica es en este contexto muy importante. Un objeto en una colección puede ser manejado con la semántica de "valor" (su vida depende totalmente en el propietario de la colección), o podría ser una referencia a otra entidad con su propio ciclo de vida. En este último caso, sólo el "vínculo" entre los dos objetos se considera que es un estado en poder de la colección. (Hibernate, 2016)

```
Cat = new DomesticCat();
Cat kitten = new DomesticCat();
....
Set kittens = new HashSet();
kittens.add(kitten);
cat.setKittens(kittens);
session.persist(cat);

kittens = cat.getKittens(); // Okay, kittens collection is a Set
(HashSet) cat.getKittens(); // Error!
```

Ilustración 12 Colección de persistencia
Fuente: (Hibernate, 2016)

Las colecciones persistentes inyectadas por Hibernate se comportan como HashMap, HashSet, TreeMap, TreeSet o ArrayList, dependiendo del tipo de interfaz. (Hibernate, 2016)

Colecciones casos tienen el comportamiento usual de los tipos de valor. Ellos se conservan de forma automática cuando se hace referencia por un objeto persistente y se borran automáticamente cuando no son referenciados. Si una colección se pasa de un objeto persistente a otro, sus elementos pueden ser movidos de una mesa a otra. Dos entidades no pueden compartir una referencia a la misma instancia de colección. Debido al modelo relacional subyacente, las propiedades de recolección de valores no soportan la semántica de valor nulo. Hibernate no distingue entre una referencia de colección nula y una colección vacía. (Hibernate, 2016)

2.3.7.2 SOCIACIONES UNIDIRECCIONALES

2.3.7.2.1 Muchos-a-uno

Una asociación unidireccional muchos-a-uno es el tipo más común de asociación unidireccional. (Hibernate, 2016)

```
<class name="Person">
  <id name="id" column="personId">
    <generator class="native"/>
  </id>
  <many-to-one name="address"
    column="addressId"
    not-null="true"/>
</class>
<class name="Address">
  <id name="id" column="addressId">
    <generator class="native"/>
  </id>
</class>
```

Ilustración 13 Muchos-a-uno
Fuente: (Hibernate, 2016)

2.3.7.2.2 Uno-a-uno

Una unidireccional uno-ha-una asociación de una clave primaria generalmente utiliza un generador especial Identificación En este ejemplo, sin embargo, hemos invertido la dirección de la asociación. (Hibernate, 2016)

```
<class name="Person">
  <id name="id" column="personId">
    <generator class="native"/>
  </id>
</class>

<class name="Address">
  <id name="id" column="personId">
    <generator class="foreign">
      <param name="property">person</param>
    </generator>
  </id>
  <one-to-one name="person" constrained="true"/>
</class>
create table Person ( personId bigint not null primary key )
create table Address ( personId bigint not null primary key )
```

Ilustración 14 Uno-a-uno
Fuente: (Hibernate, 2016)

2.3.7.2.3 Uno-a-muchos

Una asociación unidireccional uno-a-muchos en una clave externa es un caso inusual, y no se recomienda. (Hibernate, 2016)

```
<class name="Person">
  <id name="id" column="personId">
    <generator class="native"/>
  </id>
  <set name="addresses">
    <key column="personId"
      not-null="true"/>
    <one-to-many class="Address"/>
  </set>
</class>
<class name="Address">
  <id name="id" column="addressId">
    <generator class="native"/>
  </id>
</class>
```

Ilustración 15 Uno-a-muchos
Fuente: (Hibernate, 2016)

2.3.8 TRABAJO CON OBJETOS

Hibernate es una solución completa de mapeo objeto/relacional que no sólo protege al desarrollador de los detalles del sistema de administración de la base datos subyacente, sino que además ofrece administración de estado de objetos. Contrario a la administración de declaraciones SQL en capas comunes de persistencia JDBC/SQL, esta es una vista natural orientada a objetos de la persistencia en aplicaciones Java. (Hibernate, 2016)

En otras palabras, los desarrolladores de aplicaciones de Hibernate siempre deben pensar en el estado de sus objetos, y no necesariamente en la ejecución de declaraciones SQL. Hibernate se ocupa de esto y es solamente relevante para el desarrollador de la aplicación al afinar el rendimiento del sistema. (Hibernate, 2016)

2.3.8.1 ESTADOS DE OBJETO DE HIBERNATE

Hibernate define y soporta los siguientes estados de objeto:

- **Transitorio -** un objeto es transitorio si ha sido recién instanciado utilizando el operador new, y no está asociado a una Session de Hibernate. No tiene una representación persistente en la base de datos y no se le ha asignado un valor identificador. Las instancias transitorias serán destruidas por el

recolector de basura si la aplicación no mantiene más una referencia. Utiliza la Session de Hibernate para hacer un objeto persistente. (Hibernate, 2016)

- **Persistente** - una instancia persistente tiene una representación en la base de datos y un valor identificador. Puede haber sido guardado o cargado, sin embargo, por definición, se encuentra en el ámbito de una Session. Hibernate detectará cualquier cambio realizado a un objeto en estado persistente y sincronizará el estado con la base de datos cuando se complete la unidad de trabajo. Los desarrolladores no ejecutan declaraciones UPDATE manuales, o declaraciones DELETE cuando un objeto se debe poner como transitorio. (Hibernate, 2016)
- **Separado -** una instancia separada es un objeto que se ha hecho persistente, pero su Session ha sido cerrada. La referencia al objeto todavía es válida, por supuesto, y la instancia separada podría incluso ser modificada en este estado. Una instancia separada puede ser re-unida a una nueva Session más tarde, haciéndola persistente de nuevo (con todas las modificaciones). Este aspecto habilita un modelo de programación para unidades de trabajo de ejecución larga que requieren tiempo-para-pensar por parte del usuario. Las llamamos transacciones de aplicación, por ejemplo, una unidad de trabajo desde el punto de vista del usuario. (Hibernate, 2016)

2.3.9 TRANSACCIONES Y CONCURRENCIA

El punto más importante sobre Hibernate y el control de concurrencia es que es fácil de comprender. Hibernate usa directamente conexiones JDBC y recursos JTA sin agregar ningún comportamiento de bloqueo adicional. Le recomendamos bastante que tome algo de tiempo con la especificación de JDBC, ANSI y el aislamiento de transacciones de su sistema de gestión de base de datos. (Hibernate, 2016)

Hibernate no bloquea objetos en la memoria. Su aplicación puede esperar el comportamiento definido por el nivel de aislamiento de sus transacciones de las bases de datos. Gracias a la Session, la cual también es un caché con alcance de transacción, Hibernate proporciona lecturas repetidas para búsquedas del identificador y consultas de entidad y no consultas de reporte que retornan valores escalares. (Hibernate, 2016)

Además del versionado del control de concurrencia optimista automático, Hibernate también ofrece una API para bloqueo pesimista de filas, usando la sintaxis SELECT FOR UPDATE. Esta API y el control de concurrencia optimista se discuten más adelante en este capítulo. (Hibernate, 2016)

Comenzamos la discusión del control de concurrencia en Hibernate con la granularidad de Configuration, SessionFactory y Session, así como las transacciones de la base de datos y las conversaciones largas. (Hibernate, 2016)

2.3.9.1 ÁMBITOS DE SESIÓN Y DE TRANSACCIÓN

Una SessionFactory es un objeto seguro entre hilos y costoso de crear pensado para que todas las hebras de la aplicación lo compartan. Se crea una sola vez, usualmente en el inicio de la aplicación, a partir de una instancia Configuration. (Hibernate, 2016)

Una Session es un objeto de bajo costo, inseguro entre hilos que se debe utilizar una sola vez y luego se debe descartar: para un sólo pedido, una sola conversación o una sola unidad de trabajo. Una Session no obtendrá una Connection JDBC o un Datasource a menos de que sea necesario. No consumirá recursos hasta que se utilice. (Hibernate, 2016)

Una transacción de la base de datos tiene que ser tan corta como sea posible para reducir la contención de bloqueos en la base de datos. Las transacciones largas de la base de datos prevendrán a su aplicación de escalar a una carga altamente concurrente. Por lo tanto, no se recomienda que mantenga una transacción de la base de datos abierta durante el tiempo para pensar del usuario, hasta que la unidad de trabajo se encuentre completa. (Hibernate, 2016)

2.3.10 HQL: EL LENGUAJE DE CONSULTA DE HIBERNATE

Hibernate utiliza un lenguaje de consulta potente (HQL) que se parece a SQL. Sin embargo, comparado con SQL, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. (Hibernate, 2016)

2.3.10.1 SENSIBILIDAD A MAYÚSCULAS

Las consultas no son sensibles a mayúsculas, a excepción de los nombres de las clases y propiedades Java. De modo que SeLeCT es lo mismo que sELEct e igual a SELECT, pero org.hibernate.eg.FOO no es lo mismo que org.hibernate.eg.Foo y foo.barSet no es igual a foo.BARSET. (Hibernate, 2016)

Este manual utiliza palabras clave HQL en minúsculas. Algunos usuarios encuentran que las consultas con palabras clave en mayúsculas son más fáciles de leer, pero esta convención no es apropiada para las peticiones incluidas en código Java. (Hibernate, 2016)

2.3.10.2 LA CLÁUSULA FROM

La consulta posible más simple de Hibernate es de esta manera. (Hibernate, 2016)

```
from eg.Cat
```

Esto retorna todas las instancias de la clase eg.Cat. Usualmente no es necesario calificar el nombre de la clase ya que auto-import es el valor predeterminado. (Hibernate, 2016)

```
from Cat
```

Con el fin de referirse al Cat en otras partes de la petición, necesitará asignar un alias. (Hibernate, 2016)

```
from Cat as cat
```

Esta consulta asigna el alias cat a las instancias Cat, de modo que puede utilizar ese alias luego en la consulta. La palabra clave as es opcional. También podría escribir. (Hibernate, 2016)

```
from Cat cat
```

2.3.10.3 ASOCIACIONES Y UNIONES (JOINS)

También puede asignar alias a entidades asociadas o a elementos de una colección de valores utilizando una join. (Hibernate, 2016)

```
from Cat as cat
  inner join cat.mate as mate
  left outer join cat.kittens as kitten
```

Ilustración 16 Asociaciones y uniones
Fuente: (Hibernate, 2016)

Los tipos de uniones soportadas se tomaron prestados de ANSI SQL

- inner join
- left outer join
- right outer join
- full join (no es útil usualmente)

(Hibernate, 2016)

Las construcciones inner join, left outer join y right outer join se pueden abreviar. (Hibernate, 2016)

```
from Cat as cat
  join cat.mate as mate
  left join cat.kittens as kitten
```

Ilustración 17 Inner left
Fuente: (Hibernate, 2016)

Puede proveer condiciones extras de unión utilizando la palabra clave with de HQL. (Hibernate, 2016)

```
from Cat as cat
  left join cat.kittens as kitten
    with kitten.bodyWeight
> 10.0
```

Ilustración 18 Condiciones extras inner join
Fuente: (Hibernate, 2016)

2.3.10.4 FORMAS DE SINTAXIS UNIDA

HQL soporta dos formas de unión de asociación: implicit y explicit. Las consultas que se mostraron en la sección anterior todas utilizan la forma explicit, en donde la palabra clave join se utiliza explícitamente en la cláusula from. Esta es la forma recomendada. (Hibernate, 2016)

La forma implicit no utiliza la palabra clave join. Las asociaciones se "desreferencian" utilizando la notación punto. Uniones implicit pueden aparecer en cualquiera de las cláusulas HQL. La unión implicit causa uniones internas (inner joins) en la declaración SQL que resulta. (Hibernate, 2016)

```
from Cat as cat where cat.mate.name like '%s%'
```

2.3.10.5 REFERENCIA A LA PROPIEDAD IDENTIFICADORA

Hay dos maneras de referirse a la propiedad identificadora de una entidad.

- La propiedad especial (en minúsculas) id se puede utilizar para referenciar la propiedad identificadora de una entidad dado que la entidad no defina un id del nombre de la propiedad no-identificadora. (Hibernate, 2016)
- Si la entidad define una propiedad identificadora nombrada, puede utilizar ese nombre de propiedad. (Hibernate, 2016)

Las referencias a propiedades identificadoras compuestas siguen las mismas reglas de nombramiento. Si la entidad no tiene un id del nombre de la propiedad no identificadora, la propiedad identificadora compuesta solamente puede ser referenciada por su nombre definido. De otra manera se puede utilizar la propiedad id especial para referenciar la propiedad identificadora. (Hibernate, 2016)

2.3.10.6 LA CLÁUSULA SELECT

La cláusula select escoge qué objetos y propiedades devolver en el conjunto de resultados de la consulta. Considere lo siguiente. (Hibernate, 2016)

```
select mate
from Cat as cat
inner join cat.mate as mate
```

Ilustración 19 Clausula select
Fuente: (Hibernate, 2016)

La consulta seleccionará mate de otros Cats. Puede expresar esta consulta de una manera más compacta así. (Hibernate, 2016)

```
select cat.mate from Cat cat
```

2.3.10.7 FUNCIONES DE AGREGACIÓN

Las consultas HQL pueden incluso retornar resultados de funciones de agregación sobre propiedades. (Hibernate, 2016)

```
select avg(cat.weight), sum(cat.weight), max(cat.weight),
count(cat)
from Cat cat
```

Ilustración 20 Agregación
Fuente: (Hibernate, 2016)

Las funciones de agregación soportadas son:

- avg(...), sum(...), min(...), max(...)
- count(*)
- count(...), count(distinct ...), count(all...)

(Hibernate, 2016)

2.3.10.8 LA CLÁUSULA WHERE

La cláusula where le permite refinar la lista de instancias retornadas. Si no existe ningún alias, puede referirse a las propiedades por nombre. (Hibernate, 2016)

```
from Cat where name='Fritz'
```

Si existe un alias, use un nombre de propiedad calificado. (Hibernate, 2016)

```
from Cat as cat where cat.name='Fritz'
```

Esto retorna instancias de Cat llamadas 'Fritz'.

2.3.10.9 EXPRESIONES

Las expresiones utilizadas en la cláusula where incluyen lo siguiente:

- operadores matemáticos: +, -, *, /
- operadores de comparación binarios: =, >=, <=, <>, !=, like
- operadores lógicos and, or, not
- Paréntesis ( ) que indican agrupación

- in, not in, between, is null, is not null, is empty, is not empty, member of y not member of
- concatenación de cadenas...||... o concat(...,...)
- current_date(), current_time() y current_timestamp()
- second(...), minute(...), hour(...), day(...), month(...), and year(...)

(Hibernate, 2016)

2.4 ARQUITECTURA DE DESARROLLO N-CAPAS

La Arquitectura de la Aplicación se refiere básicamente a la forma, tanto física, como lógica en la que está constituida una aplicación que es de suma importancia en el éxito o fracaso de un proyecto. Se trata de un elemento fundamental para que el correcto desarrollo de una aplicación sea posible. (UCE, 2016)

Definimos a continuación los conceptos de Arquitectura Física de la Aplicación y Arquitectura Lógica de la Aplicación. (UCE, 2016)

- **Arquitectura Lógica:** Es el diseño conceptual de nuestra aplicación. Aquí se agrupara la arquitectura de la información. (UCE, 2016)
- **Arquitectura Física:** Se trata de la forma en que se distribuye nuestra aplicación a los usuarios finales en el cual se denotan los actores y el medio a través del cual llega la aplicación al computador y/o dispositivo del usuario. (UCE, 2016)

El modelo de N-Capas es la arquitectura más utilizada para construir aplicaciones. Este tipo de modelo se basa en la separación de la lógica de negocio de la lógica de diseño, separando la Capa de Negocio, la Capa de Presentación y la Capa de Datos. (UCE, 2016).

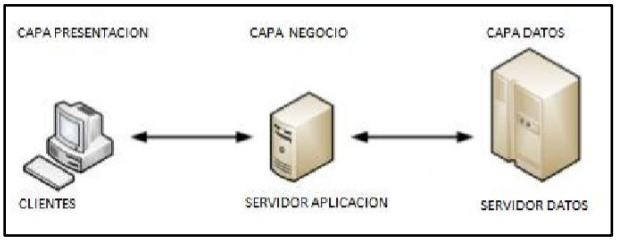


Ilustración 21 Arquitectura N-Capas
Fuente: (UCE, 2016)

Capa de presentación: Es la interfaz gráfica la cual es entendible y fácil de utilizar para el usuario, muestra las funcionalidades del sistema, y le permite la comunicación con la aplicación tanto en presentación y captura de la información. Esta capa se comunica con la capa de negocio. (UCE, 2016)

Capa de negocio: esta capa contiene la lógica principal de procesamiento de datos dentro de nuestra aplicación Web. Se comunica con la capa de presentación para obtener las entradas del usuario y presentar la información resultante, así como la capa de acceso a datos o directamente con servicios para realizar sus operaciones. (UCE, 2016)

Capa de datos: En esta capa es en donde se maneja los procedimientos para almacenar u obtener información de la base de datos a partir la aplicación web. Esta capa será la intermediaria entre la aplicación web y la base de datos respondiendo a todas las formas, cálculos complejos y accesos recurrentes que se utilizan para acceder a la información la cual es uno de los recursos de suma importancia a nivel empresarial . (UCE, 2016)

2.4.1 JAVA DATABASE CONNECTIVITY

2.4.1.1 INTRODUCCIÓN A JDBC

Java Database Connectivity (JDBC) es una interface de acceso a bases de datos estándar SQL que proporciona un acceso uniforme a una gran variedad de bases de datos relacionales. JDBC también proporciona una base común para la construcción de herramientas y utilidades de alto nivel. (Unam, 2016).

El paquete actual de JDK incluye JDBC y el puente JDBC-ODBC. Estos paquetes son para su uso con JDK 1.0. (Unam, 2016)

Drivers JDBC

Para usar JDBC con un sistema gestor de base de datos en particular, es necesario disponer del driver JDBC apropiado que haga de intermediario entre ésta y JDBC. Dependiendo de varios factores, este driver puede estar escrito en Java puro, o ser una mezcla de Java y métodos nativos JNI. (Unam, 2016).

2.4.1.2 ¿QUÉ ES JDBC?

JDBC es el API para la ejecución de sentencias SQL. (Como punto de interés JDBC es una marca registrada y no un acrónimo, no obstante a menudo es conocido como "Java Database Connectivity"). Consiste en un conjunto de clases e interfaces escritas en el lenguaje de programación Java. JDBC suministra un API estándar para los desarrolladores y hace posible escribir aplicaciones de base de datos usando un API puro Java. (Unam, 2016)

Usando JDBC es fácil enviar sentencias SQL virtualmente a cualquier sistema de base de datos. En otras palabras, con el API JDBC, no es necesario escribir un programa que acceda a una base de datos Sybase, otro para acceder a Oracle y otro para acceder a Informix. Un único programa escrito usando el API JDBC y el programa será capaz de enviar sentencias SQL a la base de datos apropiada. Y, con una aplicación escrita en el lenguaje de programación Java, tampoco es necesario escribir diferentes aplicaciones para ejecutar en diferentes plataformas. La combinación de Java y JDBC permite al programador escribir una sola vez y ejecutarlo en cualquier entorno. (Unam, 2016).

Java, siendo robusto, seguro, fácil de usar, fácil de entender, y descargable automáticamente desde la red, es un lenguaje base excelente para aplicaciones de base de datos. (Unam, 2016).

2.4.1.3 ¿QUÉ HACE JDBC?

Simplemente JDBC hace posible estas tres cosas. (Unam, 2016)

- Establece una conexión con la base de datos.
- Envía sentencias SQL
- Procesa los resultados.

```
Connection con = DriverManager.getConnection (
            "jdbc:odbc:wombat", "login", "password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next()) {
  int x = rs.getInt("a");
  String s = rs.getString("b");
  float f = rs.getFloat("c");
  }
```

Ilustración 22 Conexión JDBC N-Capas
Fuente: (Unam, 2016)

2.4.1.4 JDBC ES UN API DE BAJO NIVEL Y UNA BASE PARA API'S DE ALTO NIVEL

JDBC es una interface de bajo nivel, lo que quiere decir que se usa para 'invocar' o llamar a comandos SQL directamente. En esta función trabaja muy bien y es más fácil de usar que otros API's de conexión a bases de datos, pero está diseñado de forma que también sea la base sobre la cual construir interfaces y herramientas de alto nivel. Una interface de alto nivel es 'amigable', usa un API más entendible o más conveniente que luego se traduce en la interface de bajo nivel tal como JDBC. (Unam, 2016).

2.4.1.5 TIPOS DE DRIVERS JDBC

Los drivers que son susceptibles de clasificarse en una de estas cuatro categorías.

1. **Puente JDBC-ODBC más driver ODBC:** El producto de JavaSoft suministra acceso vía drivers ODBC. Nótese que el código binario ODBC, y en muchos casos el código cliente de base de datos, debe cargarse en cada máquina cliente que use este driver. Como resultado, este tipo de driver es el más apropiado en una red corporativa donde las instalaciones clientes no son un problema mayor, o para una aplicación en el servidor escrito en Java en una arquitectura en tres-niveles. (Unam, 2016).

2. **Driver Java parcialmente Nativo.** Este tipo de driver convierte llamadas JDBC en llamadas del API cliente para Oracle, Sybase, Informix, DB2 y otros DBMS. Nótese que como el driver puente, este estilo de driver requiere que cierto código binario sea cargado en cada máquina cliente. (Unam, 2016).

3. **Driver Java native JDBC-Net.** Este driver traduce llamadas JDBC al protocolo de red independiente del DBMS que después es traducido en el protocolo DBMS por el servidor. Este middleware en el servidor de red es capaz de conectar a los clientes puros Java a muchas bases de datos diferentes. El protocolo específico usado dependerá del vendedor. En general esta es la alternativa más flexible. (Unam, 2016).

4. **Driver puro Java y nativo-protocolo.** Este tipo de driver convierte llamadas JDBC en el protocolo de la red usado por DBMS directamente. Esto permite llamadas directas desde la máquina cliente al servidor DBMS y es la solución más práctica para accesos en intranets. Dado que muchos de estos protocolos son propietarios, los fabricantes de bases de datos serán los principales suministradores. (Unam, 2016).

2.4.1.6 APERTURA DE UNA CONEXIÓN

La forma estándar de establecer una conexión a la base de datos es mediante la llamada al método DriverManager.getConnection. Este método toma una cadena que contiene una URL. La clase DriverManager, referida como la capa de gestión JDBC, intenta localizar un driver que pueda conectar con la base de datos representada por la URL. La clase DriverManager mantiene una lista de clases Driver registradas y cuando se llama al método getConnection, se chequea con cada driver de la lista hasta que encuentra uno que pueda conectar con la base de datos especificada en la URL. El método connect de Driver usa esta URL para establecer la conexión. (Unam, 2016)

2.5 PRIMEFACES

Ilustración 23 Logo Primefaces
Fuente: http://www.primefaces.org/

Es una librería de componentes para JavaServer Faces (JSF) de código abierto que cuenta con un conjunto de componentes enriquecidos que facilitan la creación de las aplicaciones web. Primefaces está bajo la licencia de Apache License V2. Una de las ventajas de utilizar Primefaces, es que permite la integración con otros componentes como por ejemplo RichFaces. (Primefaces, 2016)

2.5.1 PROPIEDADES

- Conjunto de componentes ricos. Editor de HTML, autocompletar, cartas, gráficas o paneles, entre otros. (Primefaces, 2016).
- Soporte de ajax con despliegue parcial, lo que permite controlar qué componentes de la página actual se actualizarán y cuáles no. (Primefaces, 2016)
- Componente para desarrollar aplicaciones web para teléfonos móviles, especiales para iPhones, Palm, Android y teléfonos móviles Nokia. (Primefaces, 2016)

2.6 METODOLOGÍA XP

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (programacionextrema, 2016)

Ilustración 24 Fases de la Metodología XP
Fuente: (programacionextrema, 2016)

Características XP

- Metodología basada en prueba y error.
- Fundamentada en Valores y Prácticas.
- Expresada en forma de 12 Prácticas–Conjunto completo–Se soportan unas a otras–Son conocidas desde hace tiempo. La novedad es juntarlas

(programacionextrema, 2016)

Valores XP

- **Simplicidad XP.-** propone el principio de hacer la cosa más simple que pueda funcionar, en relación al proceso y la codificación. Es mejor hacer hoy algo simple, que hacerlo complicado y probablemente nunca usarlo mañana. (programacionextrema, 2016)
- **Comunicación.-** Algunos problemas en los proyectos tienen origen en que alguien no dijo algo importante en algún momento. XP hace casi imposible la falta de comunicación. (programacionextrema, 2016)
- **Realimentación.-** Retroalimentación concreta y frecuente del cliente, del equipo y de los usuarios finales da una mayor oportunidad de dirigir el esfuerzo eficientemente. (programacionextrema, 2016)

CAPÍTULO III

COMPARACIÓN DE LA ARQUITECTURA DE DESARROLLO N-CAPAS VERSUS LA TECNOLOGÍA ORM-HIBERNATE CON RESPECTO A LA PRODUCTIVIDAD EN EL DESARROLLO DE APLICACIONES WEB

Con el estudio realizado a la tecnología ORM-Hibernate y la arquitectura de desarrollo N-Capas se procede a realizar un análisis comparativo, para determinar cuál de ellas es la mejor opción con respecto a la productividad enfocando al acceso de dato en el desarrollo del Sistema de Gestión de Documentación.

El proceso aplicado para el análisis comparativo es el siguiente:

- Construcción de los Prototipos
- Análisis de los Resultados
- Recolección de encuestas
- Comprobación de la hipótesis

3.1 CONSTRUCCIÓN DE PROTOTIPOS

3.1.1 ESCENARIO DE PRUEBA

Se realizó dos aplicaciones web que hacen referencia a la gestión de acceso de los usuarios, la primera utilizando la arquitectura de desarrollo N-Capas y la segunda la tecnología ORM-Hibernate, para medir los indicadores de la variable dependiente de Productividad que son: el número de líneas de código y número de horas empleadas, se utiliza la herramienta LinesOfCodeWichtel y el registros manual correspondientemente. Para más información de la aplicación web. (**Ver Anexo 6**).

3.1.2 PROCESO DE PRUEBA

Teniendo en cuenta las dos aplicaciones, se elaboró los prototipos en donde se utilizó para el acceso de datos la arquitectura de desarrollo N-Capas y la tecnología ORM-Hibernate. Se tomó como referencia la gestión de acceso de los usuarios.

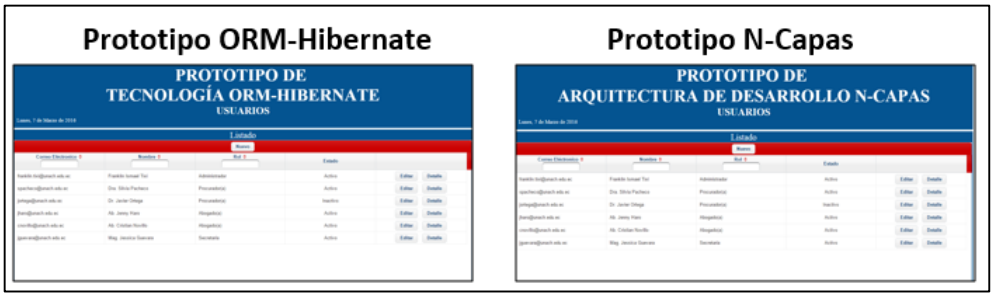


Ilustración 25 Prototipos para proceso de pruebas
Fuente: Autor

Al momento de realizar las pruebas con la herramienta LinesOfCodeWichtel, se identifica los siguientes parámetros:

- **Processed lines.-** total de número de líneas de código del archivo .java.
- **Code lines.-** número de líneas de código empleadas.
- **Blank lines.-** número de líneas en blanco.
- **Comment lines.-** número de líneas utilizadas para hacer comentarios.

Con esta herramienta se evaluó cada prototipo. Cabe mencionar que para la prueba tanto de la tecnología ORM-Hibernate y la arquitectura de desarrollo N-Capas se utilizaron el archivo DaoUsuario.java para medir el número de líneas de código.

A continuación, se muestra los resultados en tiempo real de cada prototipo.

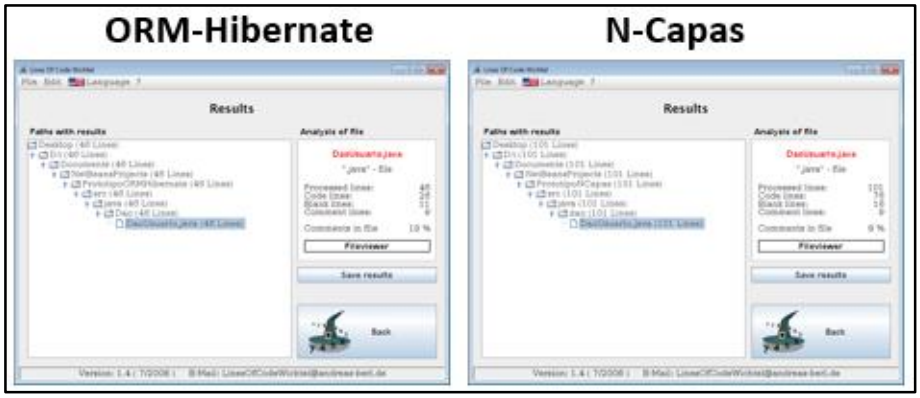


Ilustración 26 LinesOfCodeWichtel ORM-Hibernate y N-Capas
Fuente: Autor

Número de líneas de código empleadas:

LinesOfCodeWichtel – Tecnología ORM-Hibernate: 26 líneas

LinesOfCodeWichtel – Arquitectura de desarrollo N-Capas: 76 líneas

Para contabilizar y comparar el número de horas empleadas con la tecnología ORM-Hibernate y la arquitectura de desarrollo N-Capas se procedió a llevar un registro manual.

A continuación, se muestra los resultados.

Tabla 5 Horas empleadas N-Capas

REGISTRO DE LAS HORAS EMPLEADAS EN LA ARQUITECTURA DE DESARROLLO N-CAPAS	
Fecha	**Número de horas**
19/10/2015	8
20/10/2015	4

Fuente: Autor

Tabla 6 Horas empleadas ORM-Hibernate

REGISTRO DE LAS HORAS EMPLEADAS EN LA TECNOLOGÍA ORM-HIBERNATE	
Fecha	**Número de horas**
21/10/2015	5

Fuente: Autor

Tabla 7 Horas empleadas ORM-Hibernate y N-Capas

RESUMEN DE LAS HORAS EMPLEADAS CON LA TECNOLOGÍA ORM-HIBERNATE Y LA ARQUITECTURA DE DESARROLLO N-CAPAS	
	Número de horas
Arquitectura N-Capas	12
ORM-Hibernate	5

Fuente: Autor

3.2 ANÁLISIS DE RESULTADOS

3.2.1 ANALIZAR RESULTADOS

A partir de realizar las pruebas con la tecnología ORM-Hibernate y la arquitectura de desarrollo N-Capas se obtuvo los siguientes resultados:

Tabla 8 Resumen de líneas de código

	ORM-HIBERNATE	N-CAPAS
Número de líneas de código	26	76

Fuente: Autor

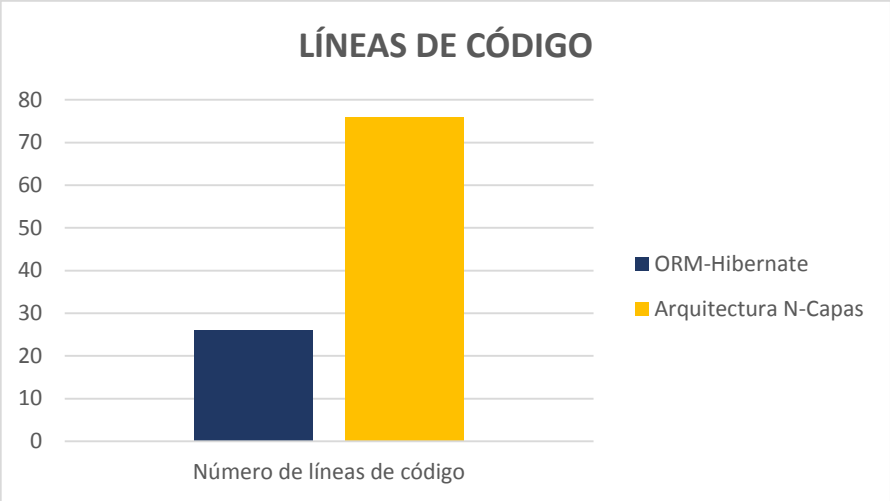


Ilustración 27 Resumen líneas de código
Fuente: Autor

Se demuestra el número de líneas de código empleadas para el desarrollo de los prototipos tanta para la tecnología ORM-Hibernate y como para la arquitectura de desarrollo N-Capas.

Tabla 9 Resumen de horas empleadas

	ORM-HIBERNATE	N-CAPAS
Número de horas empleadas	5	12

Fuente: Autor

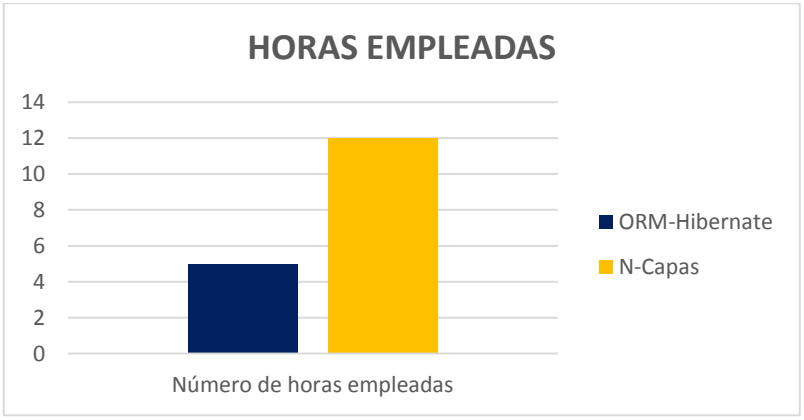


Ilustración 28 Resumen de horas empleadas
Fuente: Autor

Se demuestra el número de horas empleadas para el desarrollo de los prototipos tanta para la tecnología ORM-Hibernate y la arquitectura de desarrollo N-Capas.

Se realizó un cuadro comparativo entre la tecnología ORM-Hibernate y la arquitectura de desarrollo N-Capas para el número de líneas de código y el número de horas empleado para el desarrollo de cada prototipo.

Tabla 10 Resumen de los resultados de análisis

	N-Capas	ORM-Hibernate	Diferencia
Número de líneas de código	76	26	50
Número de horas empleadas	12	5	7
Porcentaje de líneas de código	100%	34%	66%
Porcentaje de horas empleadas	100%	42%	58%
		Promedio	62%

Fuente: Autor

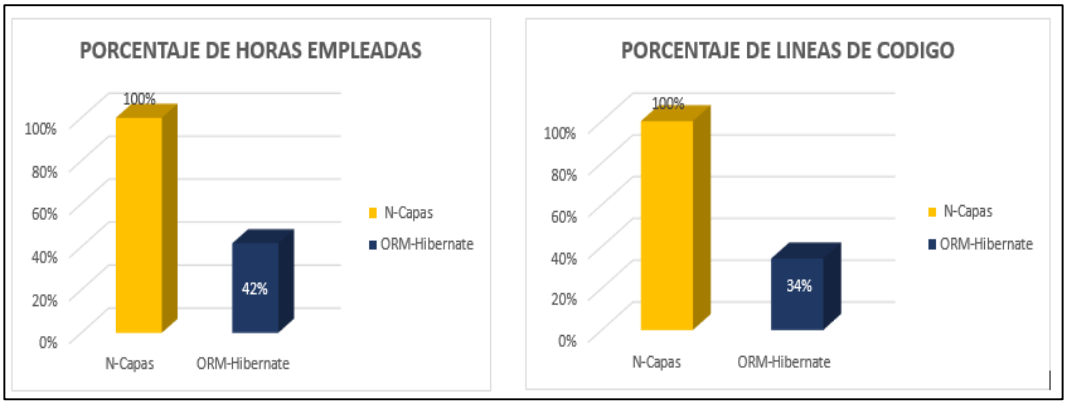


Ilustración 29 Resumen de resultado de análisis
Fuente: Autor

Después de realizar las comparaciones de los valores obtenidos, se muestran en la tabla donde porcentualmente ORM-Hibernate obtiene una mayor Productividad enfocada al acceso de datos con 62% versus a la arquitectura de desarrollo N-Capas.

3.3 RECOLECCION DE ENCUESTAS

Para ampliar la fundamentación de la presente investigación se consideró realizar encuestas a expertos en el desarrollo de las aplicaciones web (Población no Probabilístico).

Se realizó las encuestas en la compañía CityTech porque es una empresa relevante en el despliego de aplicaciones web en la ciudad de Riobamba, ya que tienen proyectos de relevancia en el desarrollo de software para Instituciones del sector Público y Privando utilizando la Arquitectura de Desarrollo N-Capas y Tecnología ORM-Hibernate; los programadores que laboran en la empresa anteriormente mencionada tienen experiencia de 5 años utilizando la Arquitectura de Desarrollo N-Capas y la Tecnología ORM-Hibernate por tal motivo se realizó las encuestas a 10 personas.

Los resultados de las encuestas son:

1. **¿Cuál de las siguientes tecnologías considera que utiliza menos tiempo en el desarrollo de aplicaciones web enfocado al acceso de datos?**

   ( ) Arquitectura N-Capas

   ( ) ORM-Hibernate

Tabla 11 Porcentaje Pregunta 1

ORM-HIBERNATE	N-CAPAS
9	1
90%	10%

Fuente: Autor

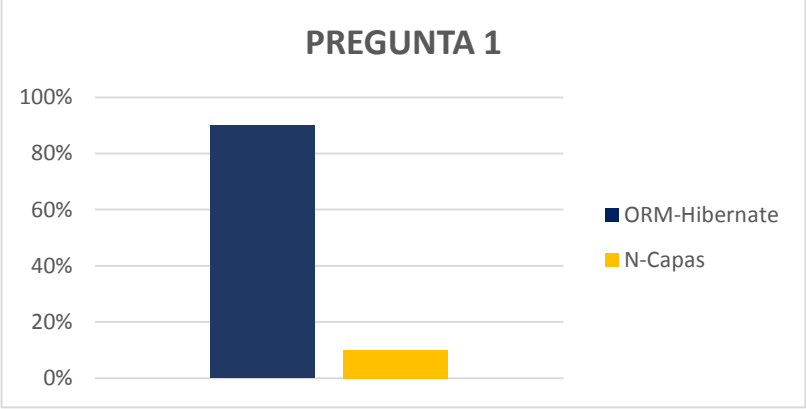


Ilustración 30 Porcentaje Pregunta 1
Fuente: Autor

Se demuestra que el 90% de los encuestados utilizan la tecnología ORM-Hibernate versus la arquitectura de desarrollo N-Capas cuando se trata de utilizar menor número de horas empleadas en el desarrollo de las aplicaciones web.

2. **¿Cuál de las siguientes tecnologías considera que utiliza menos líneas de código en el desarrollo de aplicaciones web enfocado al acceso de datos?**

( ) Arquitectura N-Capas

( ) ORM-Hibernate

Tabla 12 Porcentaje Pregunta 2

ORM-HIBERNATE	N-CAPAS
9	1
90%	10%

Fuente: Autor

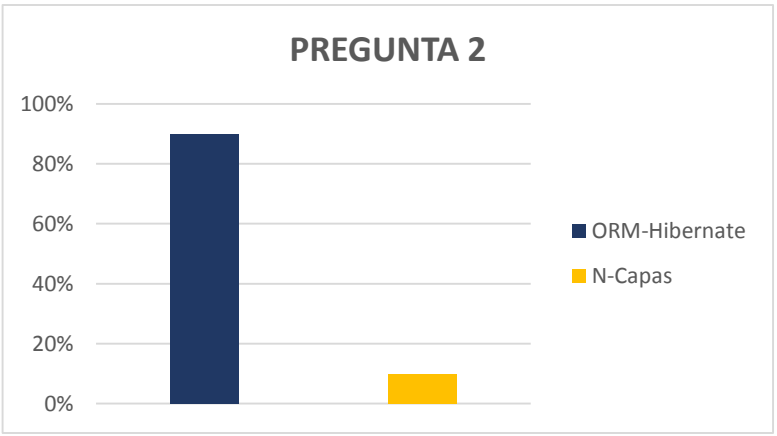


Ilustración 31 Porcentaje pregunta 2
Fuente: Autor

Se demuestra que el 90% de los encuestados utilizan la tecnología ORM-Hibernate versus la arquitectura de desarrollo N-Capas cuando se trata de utilizar menor número de líneas de código en el desarrollo de las aplicaciones web.

3. **¿Cuál de las siguientes tecnologías considera que ofrece más documentación de ayuda?**

( ) Arquitectura N-Capas

( ) ORM-Hibernate

Tabla 13 Porcentaje pregunta 3

ORM-Hibernate	N-Capas
2	8
20%	80%

Fuente: Autor

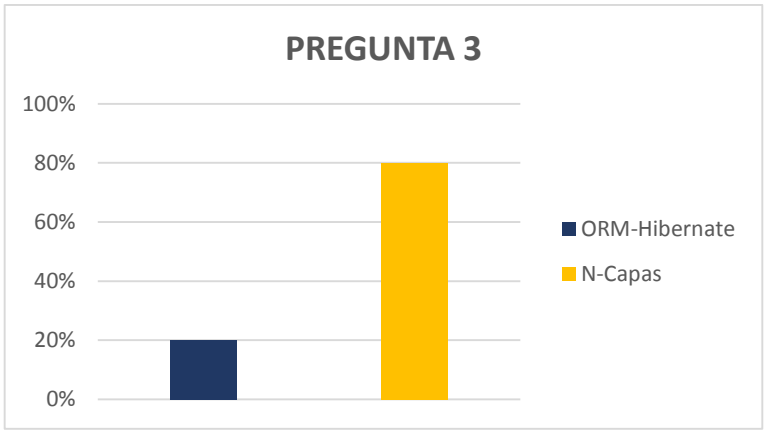


Ilustración 32 Porcentaje pregunta 3
Fuente: Autor

El grafico demuestra que existe más documentación de ayuda para la arquitectura de desarrollo N-Capas que para la tecnología ORM-Hibernate.

Resumen de los resultados de las encuestas realizadas

Tabla 14 Resumen encuestas realizadas

Preguntas	ORM-Hibernate	N-Capas
1	9	1
2	9	1
3	2	8

Fuente: Autor

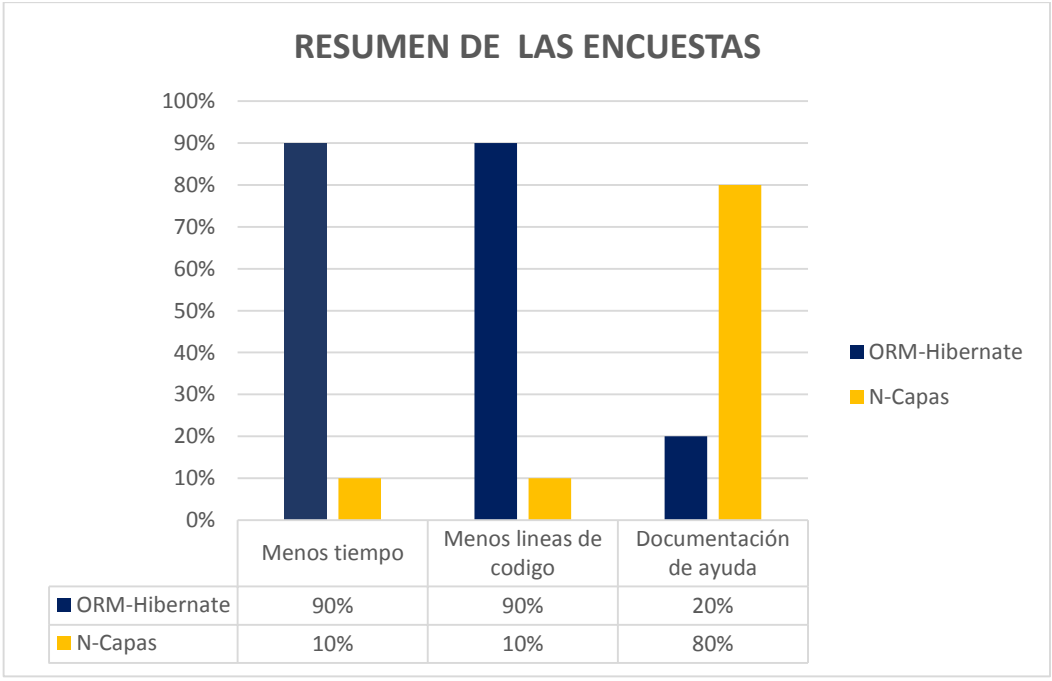


Ilustración 33 Resumen de encuestas realizadas
Fuente: Autor

La gráfica representa el resultado final de las encuestas realizadas a los expertos en el desarrollo de aplicaciones web, se determinó que ORM-Hibernate es más productivo enfocado al acceso de datos con respecto a la arquitectura de desarrollo N-Capas en líneas de código y tiempo empleado, además se determinó que ORM-Hibernate no cuenta con la suficiente información de ayuda.

3.4 COMPROBACIÓN DE LA HIPÓTESIS

Para la comprobación de la hipótesis se utiliza la estadística descriptiva, lo cual recoge datos para agruparlos y posteriormente analizarlos por medio de tablas o gráficos y decidir la mejor tecnología. Los resultados obtenidos con la ayuda de la herramienta LinesOfCodeWichtel para el número de líneas de código, el registro del número de horas empleadas en el desarrollo de los prototipos y en base al criterio de los expertos demostrado en las encuestas realizadas, se puede determinar que la tecnología ORM-Hibernate es un 62% más productivo enfocado a acceso de datos versus a la arquitectura de desarrollo N-Capas. Por lo que se establece que la tecnología ORM-Hibernate es la mejor opción a utilizar en el desarrollo del Sistema de Gestión de Documentación Fénix ya que ayuda al programador por medio del mapeo objeto/relacional, lenguaje de consultas HQL y SessionFactory.

CAPÍTULO IV

DESARROLLO DEL SISTEMA DE GESTIÓN DE DOCUMENTACIÓN PARA EL DEPARTAMENTO DE PROCURADURÍA DE LA UNIVERSIDAD NACIONAL DE CHIMBORAZO

Para el desarrollo del Sistema de Gestión de Documentación con respecto a la productividad enfocado al acceso de datos se utiliza la tecnología ORM-Hibernate. Se utilizara la metodología de desarrollo de software denominado eXtreme Programming (XP) porque se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios.

El sistema se desarrolló en la Universidad Nacional de Chimborazo y lo utilizara el Departamento de Procuraduría General, el mismo que ayudara a llevar un seguimiento de los procesos planificados y realizados por el personal administrativo.

4.1 DESARROLLO DEL SISTEMA

4.1.1 HERRAMIENTAS DE DESARROLLO

Para la implementación del Sistema de Gestión de Documentación se utilizará las siguientes tecnologías y herramientas.

Tabla 15: Herramientas de desarrollo para Fénix

HERRAMIENTA	CONCEPTO	VERSIÓN UTILIZADA
✓ **MySQL** 	Es un sistema de gestión de bases de datos relacional, multi-hilo y multiusuario. Por un lado se ofrece bajo la GNU GPL	MySQL 5.6.20

✓ **NETBEANS** 	NetBeans IDE le permite desarrollar rápida y fácilmente aplicaciones de escritorio, móviles y aplicaciones web, así como aplicaciones HTML5 con HTML, JavaScript y CSS.	NetbeanIDE 8.0
✓ **LIBRERÍA PRIMEFACES** 	PrimeFaces es una librería de componentes visuales para JSF que posee un conjunto de componentes ricos facilitando la creación de las aplicaciones web	PrimeFaces 5.0
✓ **HIBERNATE** 	Un mapeo de objeto O/R (ORM), Hibernate tiene que ver con la persistencia de datos que se aplica a las bases de datos relacionales (vía JDBC).	Hibernate 4.2.6
✓ **SERVIDOR GLASSFISH** 	GlassFish es un servidor de aplicaciones que implementa la plataforma JavaEE5, por lo que soporta las últimas versiones de tecnologías como: JSP, JSF, Servlets, Servicios Web (JAXWS), Metadatos de Servicios Web.	GlassFish 4.0

Fuente: Autor

4.1.2 GESTIÓN DEL PROYECTO

4.1.2.1 PLANIFICACIÓN DEL PROYECTO

Esta planificación del proyecto se realizó tras el estudio del problema y los requerimientos, mediante la representación de las historias se efectuó la planificación inicial la cual fue variando en el transcurso de la misma cambiando y mejorando las historias en base a concepción del problema.

4.1.2.2 INTEGRANTES Y ROLES

Con la participación del Director del proyecto, los miembros, los usuarios y desarrolladores, se formara el equipo encargado de la implementación del sistema. Esto implicara que los diseños deberán ser sencillos y claros, los usuarios dispondrán de versiones de prueba del software para que puedan participar en el proceso de desarrollo mediante sugerencias y aportaciones, dicho equipo de trabajo se ve ilustrado en la Tabla 15 definiendo Integrantes y Roles.

Tabla 16 Integrantes y Roles

Miembro	Roles XP	Metodología
Ismael Tixi	Rastreador, Testeador, Programador	XP
Ing. Diego Palacios	Consultor	

Fuente: Autor

4.1.2.3 PROTOTIPOS

Las interfaces de usuario son las más importantes ya que de esto dependerá el entendimiento fácil y rápido por parte del usuario al comenzar a manejar el sistema.

Se pretende que la interfaz del usuario sea amigable, sencilla y funcional con un alto grado de comprensión, por tal razón se crearon los prototipos generales del sistema. A continuación se realizara una breve descripción del proceso principal.

Inicio de sesión de los usuarios

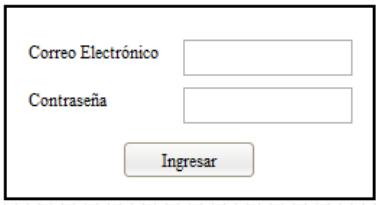


Ilustración 34 Inicio de Sesión
Fuente: Autor

En la figura se muestra la página principal del sistema

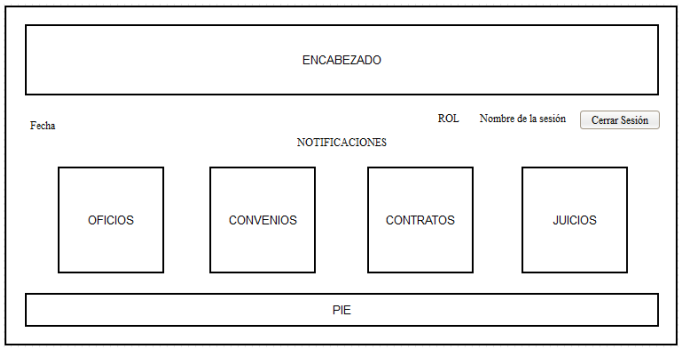


Ilustración 35 Página principal del sistema
Fuente: Autor

En la figura se muestra la página de inicio del módulo de oficios.

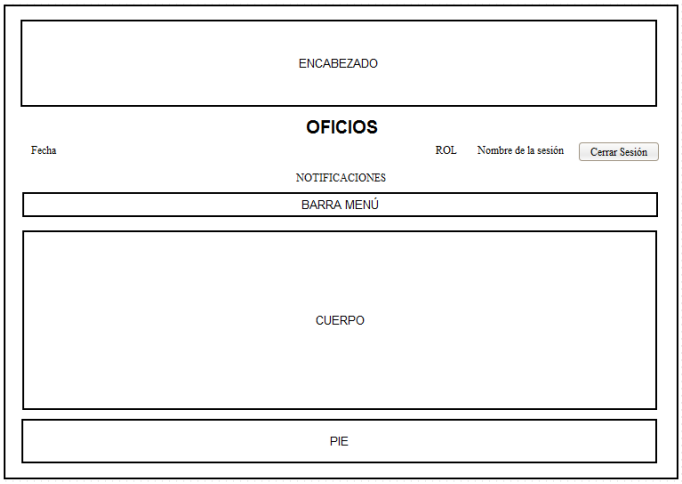


Ilustración 36 Inicio del módulo de oficios
Fuente: Autor

En la figura se muestra el registro del oficio recibido en el módulo de oficios cuando el rol sea secretaria.

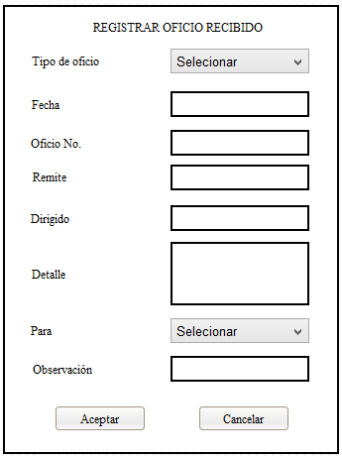


Ilustración 37 Registrar oficio recibido en el módulo de oficios
Fuente: Autor

Una pantalla para la designación del oficio en el módulo de oficios cuando el rol sea procurador.

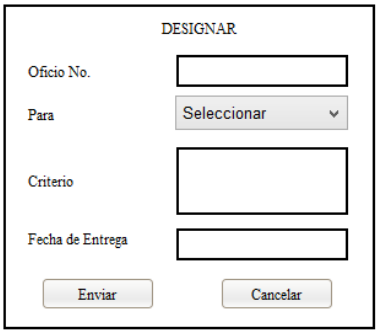


Ilustración 38 Designar oficio en el módulo de oficios
Fuente: Autor

Una pantalla para devolver el oficio en el módulo de oficios cuando el rol sea procurador.

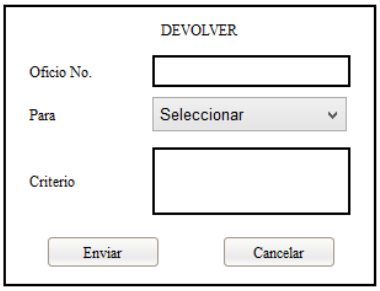


Ilustración 39 Devolver oficio en el módulo de oficios
Fuente: Autor

En la figura se registrar el informe del oficio en el módulo de oficios cuando el rol sea abogado.

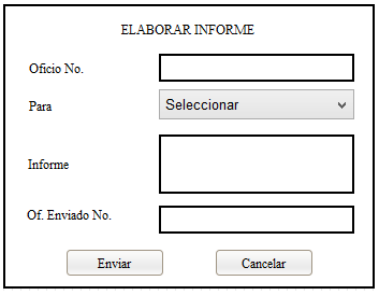


Ilustración 40 Registrar informe de oficio en el módulo de oficios
Fuente: Autor

Una pantalla donde despachar el oficio en el módulo de oficios cuando el rol sea procurador.

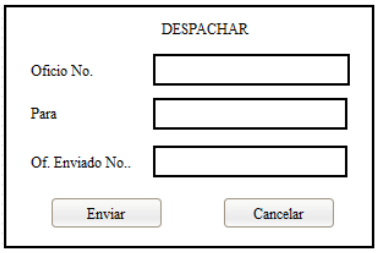


Ilustración 41 Despachar oficio
Fuente: Autor

Una pantalla para registrar el oficio enviado en el módulo de oficios cuando el rol sea secretaria.

Ilustración 42 Registrar oficio enviado
Fuente: Autor

En la figura se muestra la página de inicio en el módulo de convenios.

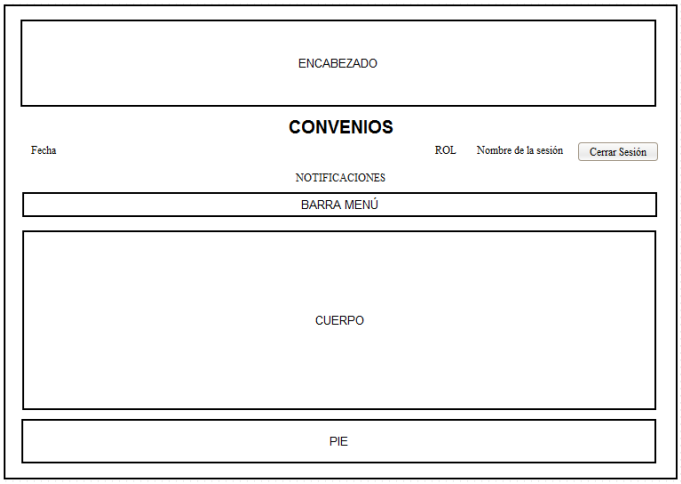


Ilustración 43 Inicio en el módulo de convenios
Fuente: Autor

En la figura se muestra el registro del oficio recibido en el módulo de convenios cuando el rol sea secretaria.

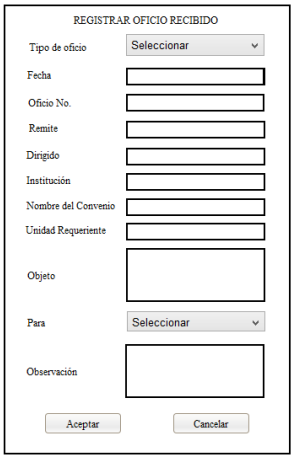


Ilustración 44 Registrar oficio recibido en el módulo de convenios
Fuente: Autor

Una pantalla para designar el oficio en el módulo de convenios cuando el rol sea procurador.

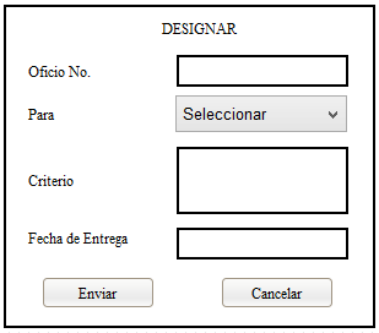


Ilustración 45 Designar oficio en el módulo de convenios
Fuente: Autor

Una pantalla para devolver el oficio en el módulo de convenios cuando el rol sea procurador.

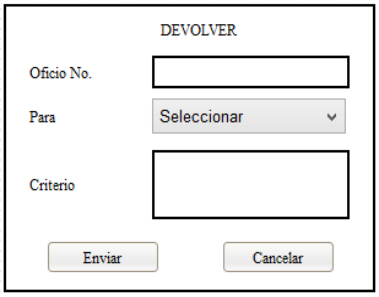


Ilustración 46 Devolver oficio en el módulo de convenios
Fuente: Autor

En la figura se muestra el registro del número de oficio en el módulo convenios cuando el rol sea abogado.

Ilustración 47 Número oficio en el módulo de convenios
Fuente: Autor

En la figura se muestra el registro del informe del oficio en el módulo de convenios cuando el rol sea abogado.

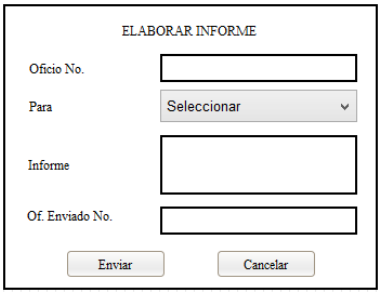


Ilustración 48 Informe oficio en el módulo de convenios
Fuente: Autor

Una pantalla para despachar el oficio en el módulo de convenios cuando el rol sea procurador.

Ilustración 49 Despachar oficio del módulo de convenios
Fuente: Autor

Una pantalla registrar el oficio enviado en el módulo de convenios cuando el rol sea abogado.

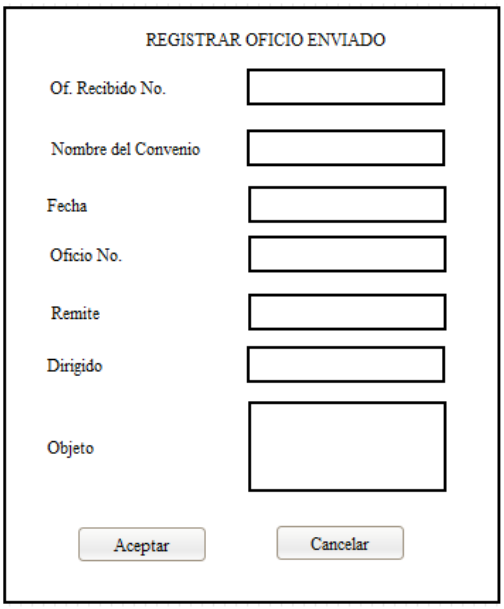


Ilustración 50 Registrar oficio enviado del módulo de convenios
Fuente: Autor

En la figura se muestra la página de inicio en el módulo de contratos.

Ilustración 51 Inicio en el módulo de contratos
Fuente: Autor

En la figura se muestra el registro del oficio recibido en el módulo de contratos cuando el rol sea secretaria.

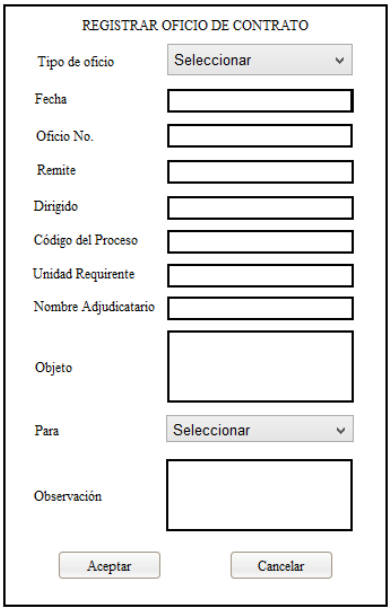


Ilustración 52 Registrar oficio recibido en el módulo de contratos
Fuente: Autor

Una pantalla para designar el oficio en el módulo de contratos cuando el rol sea procurador.

Ilustración 53 Designar oficio en el módulo de contratos
Fuente: Autor

Una pantalla devolver el oficio en el módulo de contratos cuando el rol sea procurador.

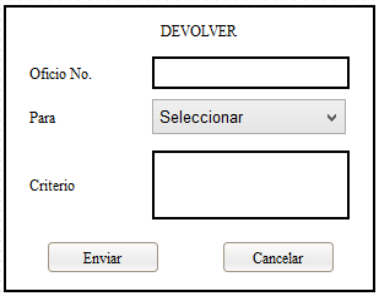


Ilustración 54 Devolver oficio en el módulo de contratos
Fuente: Autor

En la figura se muestra el registro del número de contrato en el módulo contratos cuando el rol sea abogado.

Ilustración 55 Número de contrato en el módulo de contratos
Fuente: Autor

Una pantalla para registrar el oficio de requisitos en el módulo de contratos cuando el rol sea abogado.

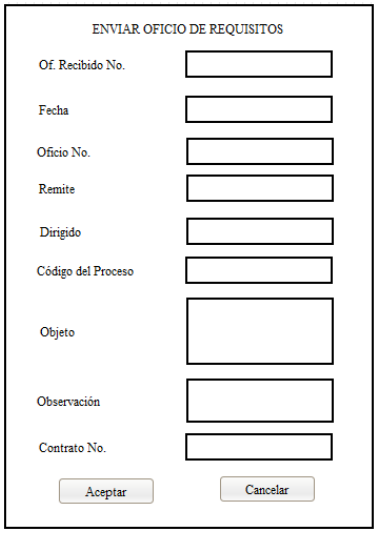


Ilustración 56 Registrar oficio de requisitos en el módulo de contratos
Fuente: Autor

Una pantalla para registrar el informe del oficio en el módulo de contratos cuando el rol sea abogado.

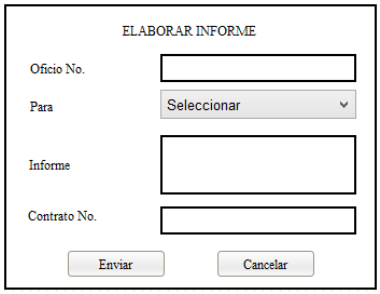


Ilustración 57 Informe oficio del módulo de contratos
Fuente: Autor

En la pantalla se muestra como despachar el oficio en el módulo de contratos cuando el rol sea procurador.

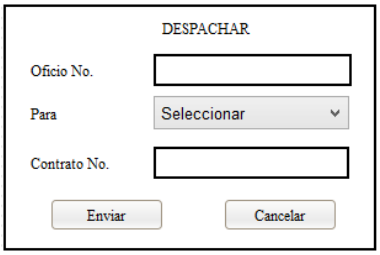


Ilustración 58 Despachar oficio en el módulo de contratos
Fuente: Autor

Una pantalla para registrar el contrato en el módulo de contratos cuando el rol sea abogado.

Ilustración 59 Registrar contrato en el módulo de contratos
Fuente: Autor

En la pantalla muestra el registro del oficio enviado en el módulo de contratos cuando el rol sea abogado.

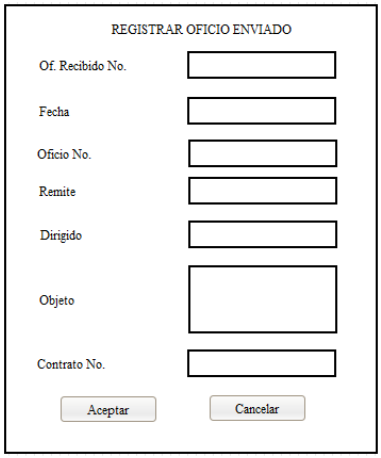


Ilustración 60 Registrar oficio enviado en el módulo de contratos
Fuente: Autor

En la figura se muestra la página de inicio en el módulo de juicios.

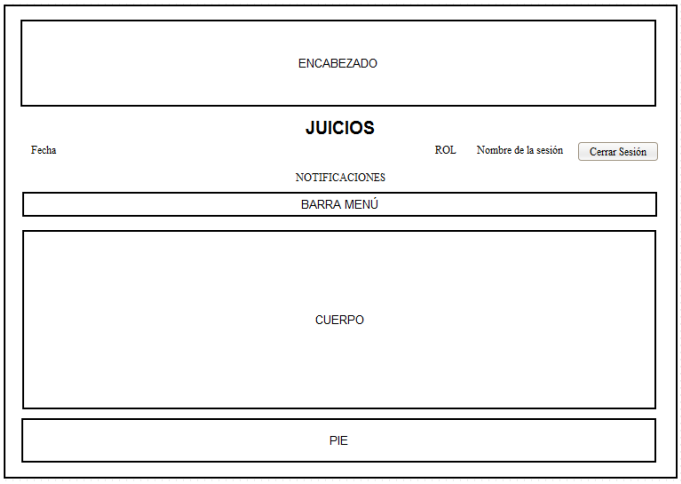


Ilustración 61 Inicio módulo de juicios
Fuente: Autor

En la figura se muestra el registro del juicio en el módulo de juicios cuando el rol sea abogado.

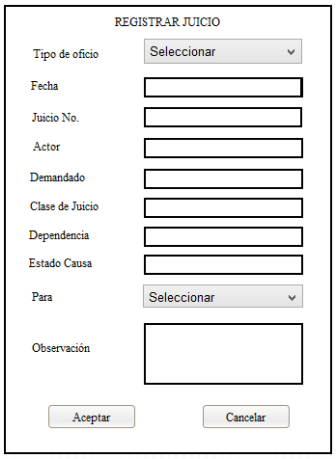


Ilustración 62 Registrar juicio en el módulo de juicios
Fuente: Autor

Una pantalla para designar el juicio en el módulo de juicios cuando el rol sea procurador.

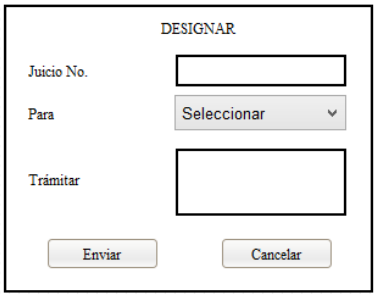


Ilustración 63 Designar juicio en el módulo de juicios
Fuente: Autor

En la figura se muestra el registro de la boleta en el módulo de juicios cuando el rol sea abogado.

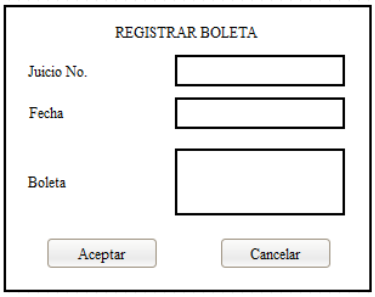


Ilustración 64 Registrar boleta en el módulo de juicios
Fuente: Autor

Una pantalla para finalizar el juicio en el módulo de juicios cuando el rol sea abogado.

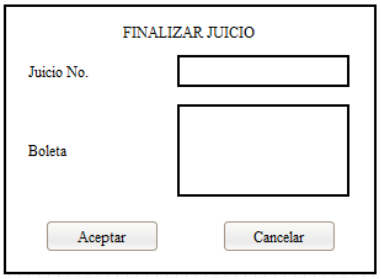


Ilustración 65 Finalizar juicio en el módulo de juicios
Fuente: Autor

4.1.2.4 HISTORIAS DE USUARIOS

Las historias de los usuarios tienen como finalidad ver las necesidades del sistema por lo tanto se realizaran descripciones cortas y escritas en el lenguaje del usuario sin terminología, detallando el tiempo que conllevara la implementación así como la estimación del riesgo de dicha historia de usuario.

Cada historia de usuario se divide en actividades planificarles y medibles para su realización, estas forman una interacción, este plan nos indicara diferentes interacciones del sistema. La realización de este plan debe tener muy en cuenta la prioridad de los usuarios para satisfacerles en mayor medida como se muestra en la tabla.

Tabla 17 Historias de Usuarios

N°	NOMBRE	PRIORIDAD	RIESGO	ESFUERZO	INTERACCIÓN
1	Gestión de Acceso de Usuarios	Alta	Alta	Medio	1
2	Gestión de Proceso de Oficios	Medio	Medio	Medio	1
3	Gestión de Proceso de Convenios	Alto	Alto	Alto	1
4	Gestión de Proceso de Contratos	Alto	Alto	Alto	1
5	Gestión de Proceso de Juicios	Medio	Medio	Medio	1
6	Emisión de Reportes	Medio	Medio	Medio	2

Fuente: Autor

4.1.2.5 PLAN DE ENTREGAS

Con cada historia de usuario previamente evaluada en tiempo de desarrollo ideal, el usuario agrupara en orden de importancia.

Para realizar el plan de entregas se hará en función de dos parámetros: tiempo de desarrollo y grado de importancia para el usuario. Las iteraciones individuales son planificadas en detalle antes de que comience cada iteración como se puede apreciar en la tabla y figuras.

Iteración 1

Tabla 18 Plan de Entrega Iteración 1

Historia de Usuario	Duración en semanas
Gestión de Acceso de Usuarios	1
Gestión de Oficios	2
Gestión de Convenios	2
Gestión de Contratos	3
Gestión de Juicios	1

Fuente: Autor

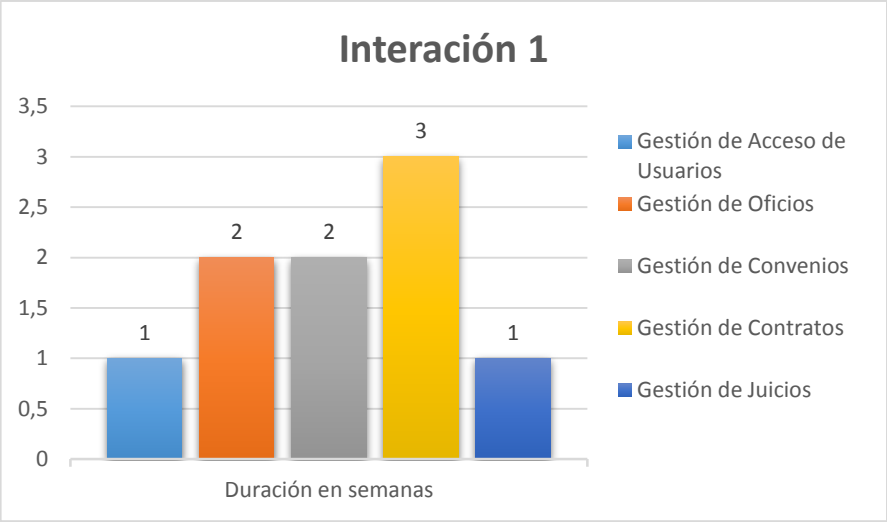


Ilustración 66 Plan de Entrega Iteración 1
Fuente: Autor

Interacción 2

Tabla 19 Plan de Entrega Iteración 2

Historia de Usuario	Duración en semanas
Emisión de Reportes	1

Fuente: Autor

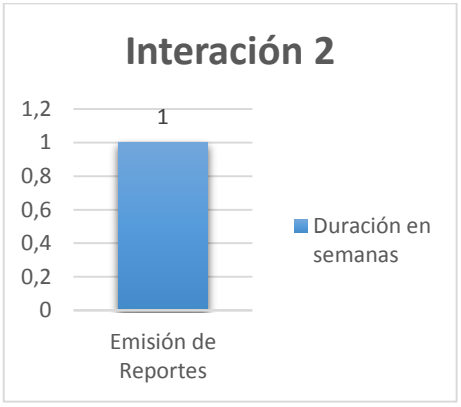


Ilustración 67 Plan de Entregas Iteración 2
Fuente: Autor

4.1.2.6 INCIDENCIA

Iteración primera: se tratara de tener preparado las funcionalidades básicas de los usuarios, las herramientas para el desarrollo del sistema. Se realizó un prototipo de la base de datos acorde a la necesidad del Departamento de Procuraduría General de la Universidad Nacional de Chimborazo (Ver Anexo 6) aprobado por las autoridades para luego implementar en el motor de base de datos MySQL, como

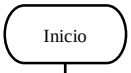
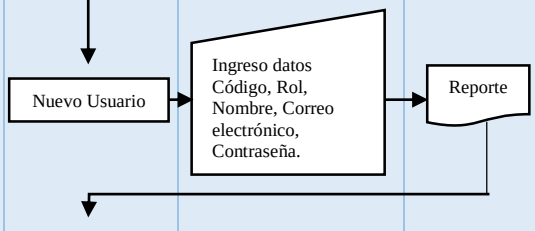
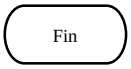
estamos utilizando la tecnología ORM-Hibernate no se crean las función CRUD en la base de datos. Una vez creadas la base de datos se procedió a crear un package denominado default package donde están la configuración de conexión y la ingeniería inversa. Después se procedió a crear un package Pojo donde va a estar las clases de las tablas que fueron mapeados. Posteriormente se creó package con el nombre de Interface donde se define el comportamiento de las clases. Después se creó una package Dao para la implementación de los interfaces. En el package BeanView que contiene los métodos que se requiere implementar para mostrar en las vistas.

Iteración segunda: Con esta iteración se pretende generar los respectivos reportes para las diferentes actividades que requiere cada usuario. Por requerimiento de la Procuradora General los reportes deben tener las actividades realizadas por cada usuario dado la fase, estado y fecha de la actividad.

4.1.2.7 ACTIVIDADES

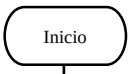
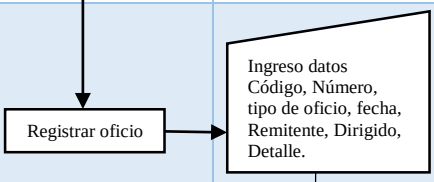
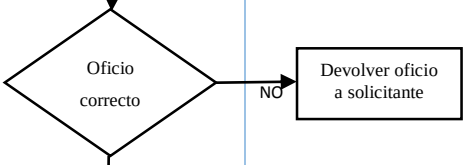
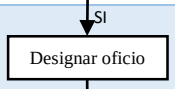
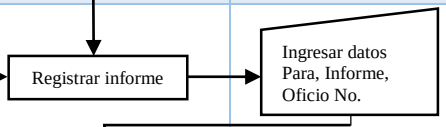
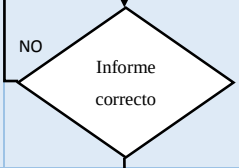
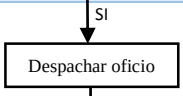
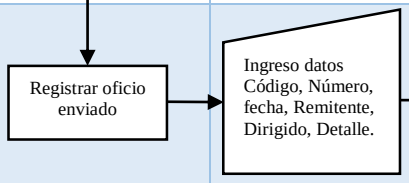
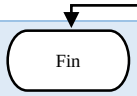
Las actividades del sistema fueron divididas en varios procesos que serán reflejados mediante flujos de procesos.

Tabla 20: Proceso nuevo usuario

PROCESO NUEVO USUARIO						
	Actividad	**Flujograma**	**IN**	**OUT**	**Responsable**	**Observación**
	Inicio					
1	Actividad				Administrador Master	
	Fin					

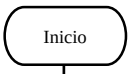
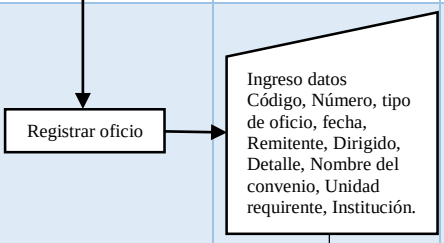
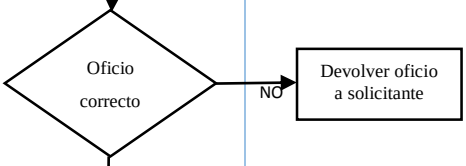
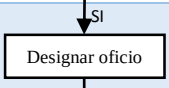
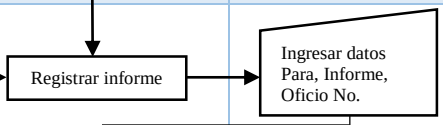
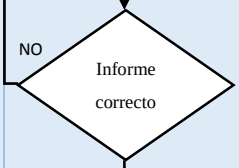
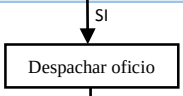
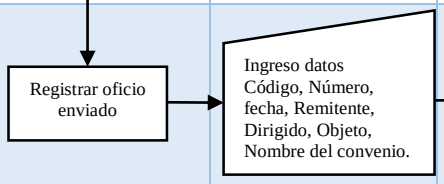
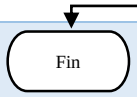
Fuente: Autor

Tabla 21 Gestión Proceso Oficio

GESTIONAR PROCESO OFICIO						
	Actividad	**Flujograma**	**IN**	**OUT**	**Responsable**	**Observación**
	Inicio					
1					Secretaria	
2					Procurador	
3					Procurador	
4					Abogado	
5					Procurador	
6					Procurador	
7					Secretaria	
8					Secretaria	
	Fin					

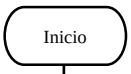
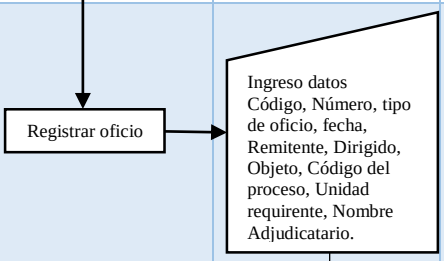
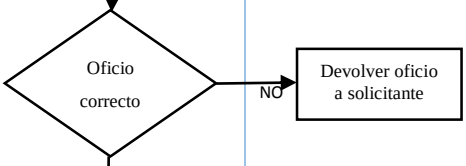
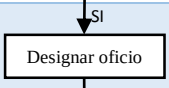
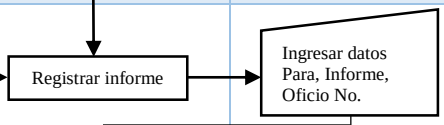
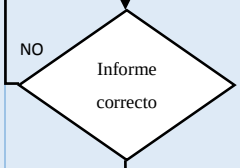
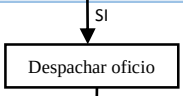
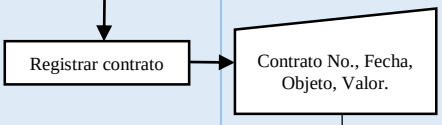
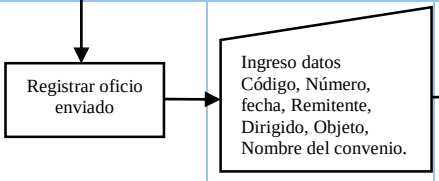
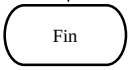
Fuente: Autor

Tabla 22 Gestionar Proceso Convenio

GESTIONAR PROCESO CONVENIO						
	Actividad	**Flujograma**	**IN**	**OUT**	**Responsable**	**Observación**
	Inicio					
1					Secretaria	
2					Procurador	
3					Procurador	
4					Abogado	
5					Procurador	
6					Procurador	
7					Abogado	
8					Abogado	
	Fin					

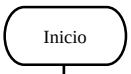
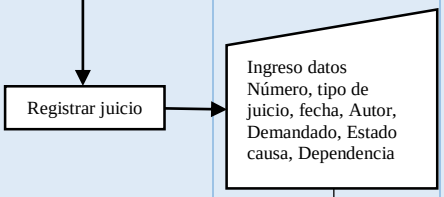
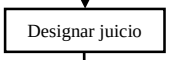
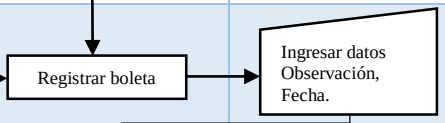
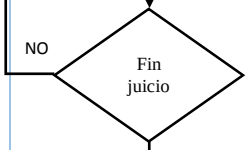
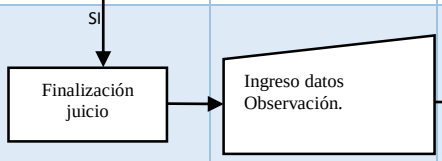
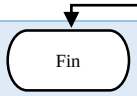
Fuente: Autor

Tabla 23 Gestionar Proceso Contrato

GESTIONAR PROCESO CONTRATO						
	Actividad	**Flujograma**	**IN**	**OUT**	**Responsable**	**Observación**
	Inicio					
1					Secretaria	
2					Procurador	
3					Procurador	
4					Abogado	
5					Procurador	
6					Procurador	
7					Abogado	
8					Abogado	
9					Abogado	
	Fin					

Fuente: Autor

Tabla 24 Gestionar Proceso Juicio

GESTIONAR PROCESO JUICIO						
	Actividad	**Flujograma**	**IN**	**OUT**	**Responsable**	**Observación**
	Inicio					
1					Abogado	
2					Procurador	
3					Abogado	
4					Abogado	
5					Abogado	
6					Secretaria	
	Fin					

Fuente: Autor

4.1.3 IMPLEMENTACIÓN

4.1.3.1 BASE DE DATOS

En la base de datos está conformado por 25 tablas para la realización de los módulos de oficios, convenios, contratos y juicios.

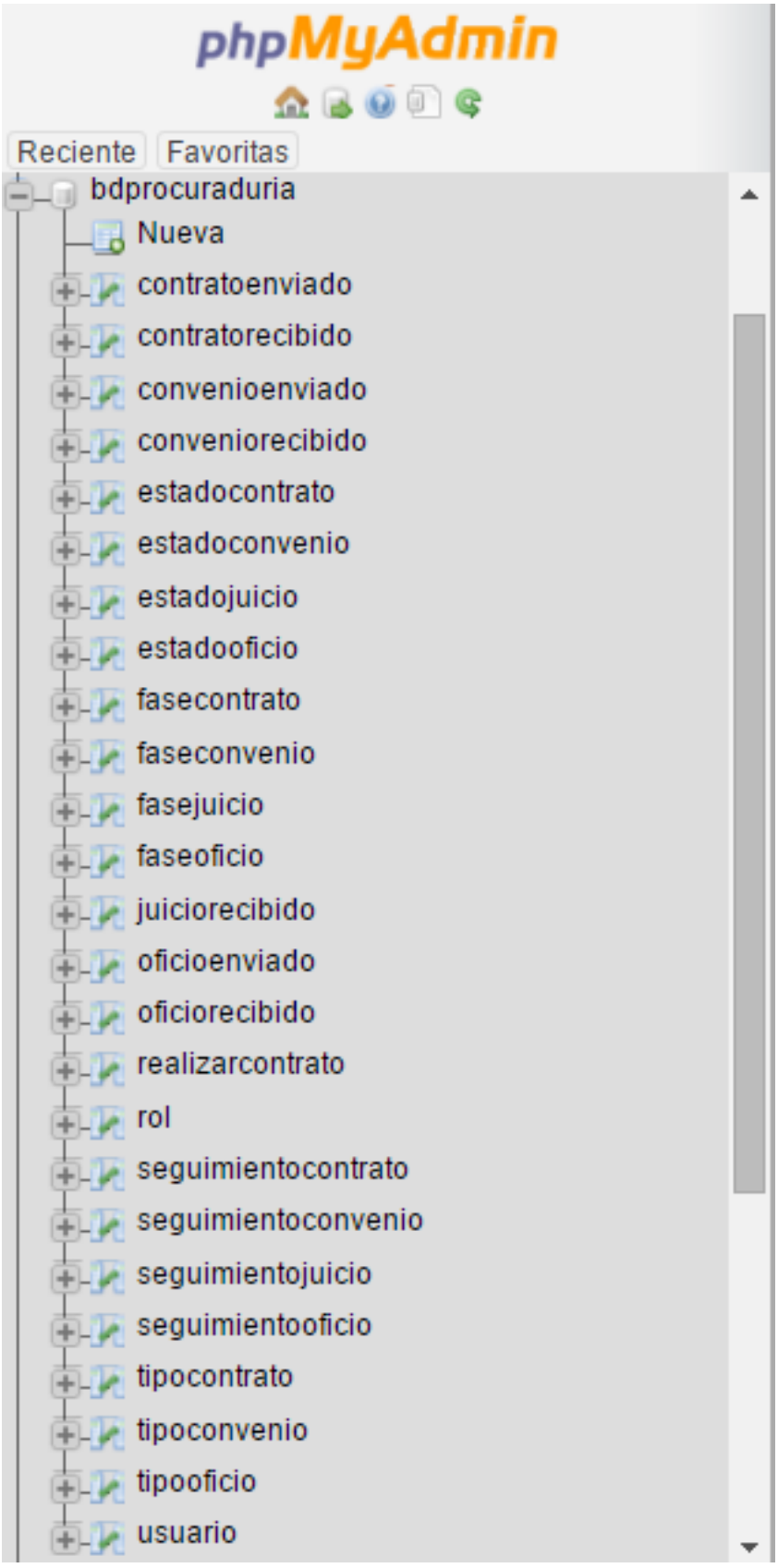


Ilustración 68 Esquema Base de Datos - MySQL
Fuente: Autor

DIAGRAMA ENTIDAD RELACION DE LA BASE DE DATOS FÉNIX

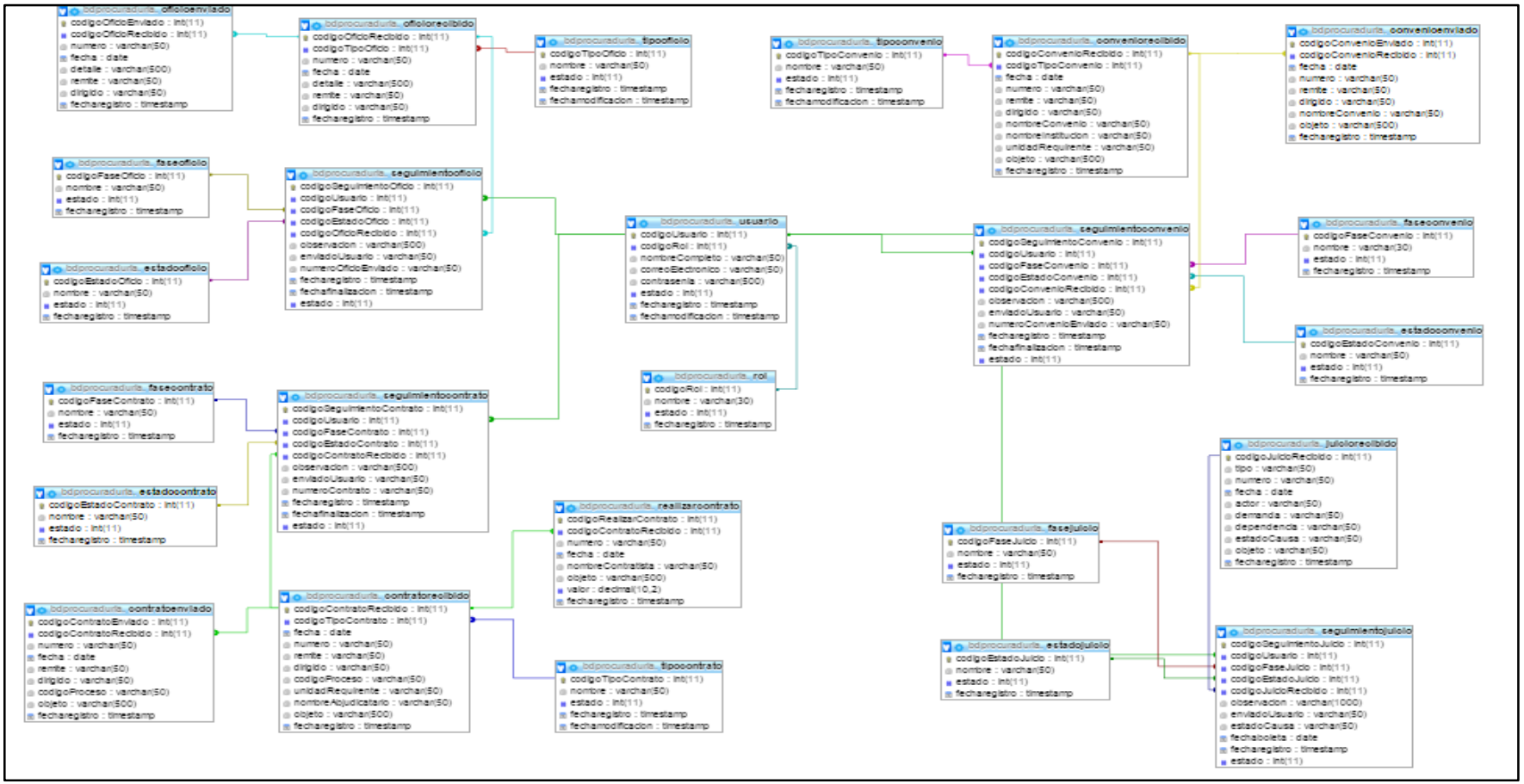


Ilustración 69 Diagrama Entidad Relacional BD Fénix

Fuente: Autor

4.1.3.2 DICCIONARIO DE DATOS

El Diccionario de datos permite guardar la estructura de la base de datos, es decir se define como se almacena y accede a la información.

- **NOMBRE DE LA TABLA: rol,** es la tabla que permite crear nuevos roles.

Tabla 25 Rol

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoRol	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: usuario**, es la tabla que permite registrar datos de un nuevo usuario.

Tabla 26 Usuario

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoUsuario	int	SI	NO	SI
codigoRol	int	NO	NO	NO
nombreCompleto	varchar	NO	NO	NO
correoElectronico	varchar	NO	NO	NO
contrasenia	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO
fechamodificacion	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: estadooficio**, es la tabla que permite registrar un nuevo estado en el módulo de oficios, para saber el estado del oficio recibido durante la gestión del mismo.

Tabla 27 EstadoOficio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoEstadoOficio	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: faseoficio**, es la tabla que permite registrar nuevas fases en el módulo de oficios, para las diferentes fase que debe seguir el oficio durante el proceso.

Tabla 28 FaseOficio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoFaseOficio	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: tipooficio**, es la tabla que permite registrar datos de un nuevo tipo de oficio en el módulo de oficios, para clasificar los oficios registrados.

Tabla 29 TipoOficio

Nombre de la columna	Tipo de dato	Clave primaria	Valores nulos	Auto incremental
codigoTipoOficio	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO
fechamodificacion	Timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: oficiorecibido**, es la tabla que permite registrar datos de un nuevo oficio recibido en el módulo de oficios.

Tabla 30 OficioRecibido

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoOficioRecibido	int	SI	NO	SI
codigoTipoOficio	int	NO	NO	NO
numero	varchar	NO	NO	NO
fecha	date	NO	NO	NO
detalle	varchar	NO	NO	NO
remite	varchar	NO	NO	NO
dirigido	varchar	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: oficioenviado**, es la tabla que permite registrar datos de un nuevo oficio enviado en el módulo de oficios, como respuesta de un oficio recibido.

Tabla 31 OficioEnviado

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoOficioEnviado	int	SI	NO	SI
codigoOficioRecibido	int	NO	NO	NO
numero	varchar	NO	NO	NO
fecha	date	NO	NO	NO
detalle	varchar	NO	NO	NO
remite	varchar	NO	NO	NO
dirigido	varchar	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: seguimientooficio**, es la tabla que permite registrar los estados, fases y el usuario que posee el oficio para ser resulto.

Tabla 32 SeguimientoOficio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoSeguimientoOficio	int	SI	NO	SI
codigoUsuario	int	NO	NO	NO
codigoFaseOficio	int	NO	NO	NO
codigoEstadoOficio	int	NO	NO	NO
codigoOficioRecibido	int	NO	NO	NO
observacion	varchar	NO	NO	NO
enviadoUsuario	varchar	NO	SI	NO
numeroOficioEnviado	varchar	NO	SI	NO
fecharegistro	timestamp	NO	SI	NO
fechafinalizacion	timestamp	NO	SI	NO
estado	int	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: estadoconvenio**, es la tabla que permite registrar nuevo estado para gestionar los oficios en el módulo de convenios.

Tabla 33 EstadoConvenio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoEstadoConvenio	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: faseconvenio**, es la tabla que permite registrar una nueva fase para gestionar los oficios en el módulo de convenios.

Tabla 34 FaseConvenio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoFaseConvenio	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: tipoconvenio**, es la tabla que permite registrar datos de un nuevo tipo de oficio en el módulo de convenios, para clasificar los oficios recibidos.

Tabla 35 TipoConvenio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoTipoConvenio	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO
fechamodificacion	Timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: conveniorecibido**, es la tabla que permite registrar datos de un nuevo oficio recibido en el módulo de convenios, para gestionar el proceso.

Tabla 36 ConvenioRecibido

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoConvenioRecibido	int	SI	NO	SI
codigoTipoConvenio	int	NO	NO	NO
fecha	date	NO	NO	NO
numero	varchar	NO	NO	NO
remite	varchar	NO	SI	NO
dirigido	varchar	NO	SI	NO
nombreConvenio	varchar	NO	SI	NO
nombreInstitucion	varchar	NO	SI	NO
unidadRequirente	varchar	NO	SI	NO
objeto	varchar	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: convenioenviado**, es la tabla que permite registrar datos de un nuevo oficio enviado en el módulo de convenios, como respuesta para la aprobación de convenio o no.

Tabla 37 ConvenioEnviado

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoConvenioEnviado	int	SI	NO	SI
codigoConvenioRecibido	int	NO	NO	NO
fecha	date	NO	NO	NO
numero	varchar	NO	NO	NO
remite	varchar	NO	SI	NO
dirigido	varchar	NO	SI	NO
nombreConvenio	varchar	NO	SI	NO
objeto	varchar	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: seguimientoconvenio**, es la tabla que permite registrar los proceso del oficio de convenio desde su inicio hasta el final en el módulo de convenios.

Tabla 38 SeguimientoConvenio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoSeguimientoConvenio	int	SI	NO	SI
codigoUsuario	int	NO	NO	NO
codigoFaseConvenio	int	NO	NO	NO
codigoEstadoConvenio	int	NO	NO	NO
codigoOficioRecibido	int	NO	NO	NO
observacion	varchar	NO	NO	NO
enviadoUsuario	varchar	NO	SI	NO
numeroOficioEnviado	varchar	NO	SI	NO
fecharegistro	timestamp	NO	SI	NO
fechafinalizacion	timestamp	NO	SI	NO
estado	int	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: estadocontrato**, es la tabla que permite registrar datos de un nuevo estado en el módulo de contratos.

Tabla 39 EstadoContrato

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoEstadoContrato	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: fasecontrato**, es la tabla que permite registrar datos de la nueva fase en el módulo de contratos.

Tabla 40 FaseContrato

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoFaseContrato	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: tipocontrato**, es la tabla que permite registrar datos de un nuevo tipo de oficio en el módulo de contratos, para clasificar los oficios de contratos.

Tabla 41 TipoContrato

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoTipoContrato	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO
fechamodificacion	Timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: contratorecibido**, es la tabla que permite registrar los datos de un oficio recibido para el proceso de contrato.

Tabla 42 ContratoRecibido

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoContratoRecibido	int	SI	NO	SI
codigoTipoContrato	int	NO	NO	NO
fecha	date	NO	NO	NO
numero	varchar	NO	SI	NO
remite	varchar	NO	SI	NO
dirigido	varchar	NO	SI	NO
codigoProceso	varchar	NO	SI	NO
unidadRequirente	varchar	NO	SI	NO
nombreAbjudicatario	varchar	NO	SI	NO
objeto	varchar	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: contratoenviado**, es la tabla que permite registrar los oficio enviado para la confirmación el registro del contrato realizado.

Tabla 43 ContratoEnviado

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoContratoEnviado	int	SI	NO	SI
codigoContratoRecibido	int	NO	NO	NO
numero	date	NO	NO	NO
fecha	varchar	NO	NO	NO
remite	varchar	NO	SI	NO
dirigido	varchar	NO	SI	NO
codigoProceso	varchar	NO	SI	NO
objeto	varchar	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: realizarcontrato**, es la tabla que permite registrar un nuevo contrato en el módulo de contratos.

Tabla 44 RealizarContrato

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoRealizarContrato	int	SI	NO	SI
codigoContratoRecibido	int	NO	NO	NO
numero	varchar	NO	NO	NO
fecha	date	NO	NO	NO
nombreContratista	varchar	NO	SI	NO
objeto	varchar	NO	NO	NO
valor	decimal	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: seguimientocontrato**, es la tabla que permite registrar los procesos que cumple el oficio de un contrato desde su registro hasta su envió de oficios para confirmar la realización del mismo.

Tabla 45 SeguimientoContrato

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoSeguimientoContrato	int	SI	NO	SI
codigoUsuario	int	NO	NO	NO
codigoFaseContrato	int	NO	NO	NO
codigoEstadoContrato	int	NO	NO	NO
codigoContratoRecibido	int	NO	NO	NO
observacion	varchar	NO	NO	NO
enviadoUsuario	varchar	NO	SI	NO
numeroContrato	varchar	NO	SI	NO
fecharegistro	timestamp	NO	SI	NO
fechafinalizacion	timestamp	NO	SI	NO
estado	int	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: estadojuicio**, es la tabla que permite registrar un nuevo estado en el módulo de juicios

.

Tabla 46 EstadoJuicio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoEstadoJuicio	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: fasejuicio**, es la tabla que permite registrar una nueva fase en el módulo de juicios.

Tabla 47 FaseJuicio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoFaseJuicio	int	SI	NO	SI
nombre	varchar	NO	NO	NO
estado	int	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: juiciorecibido**, es la tabla que permite registrar datos de un nuevo juicio en el módulo de juicios.

Tabla 48 JuicioRecibido

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoJuicioRecibido	int	SI	NO	SI
tipo	varchar	NO	NO	NO
numero	int	NO	NO	NO
fecha	date	NO	NO	NO
actor	varchar	NO	NO	NO
demanda	varchar	NO	NO	NO
dependencia	varchar	NO	NO	NO
estadoCausa	varchar	NO	NO	NO
objeto	varchar	NO	NO	NO
fecharegistro	timestamp	NO	NO	NO

Fuente: Autor

- **NOMBRE DE LA TABLA: seguimientojuicio**, es la tabla que permite registrar los movimientos de un juicios como el registro de boletas y la finalización del juicio.

Tabla 49 SeguimientoJuicio

Nombre de la Columna	Tipo de Dato	Clave Primaria	Valores Nulos	Auto Incremental
codigoSeguimientoJuicio	int	SI	NO	SI
codigoUsuario	int	NO	NO	NO
codigoFaseJuicio	int	NO	NO	NO
codigoEstadoJuicio	int	NO	NO	NO
codigoJuicioRecibido	int	NO	NO	NO
observacion	varchar	NO	NO	NO
enviadoUsuario	varchar	NO	SI	NO
estadoCausa	varchar	NO	SI	NO
fechaboleta	date	NO	SI	NO
fecharegistro	timestamp	NO	NO	NO
estado	int	NO	NO	NO

Fuente: Autor

4.1.3.3 PROTOTIPOS INTERFACES DE USUARIO FINALES

Con la descripción detallada de las historias de los proceso de gestión de oficios, convenios, contratos y juicios además con la información de los diagramas de procesos podemos definir las interfaces de usuario finales, las cuales serán implantadas en el sistema.

Ilustración 70 Control de Acceso de Usuarios
Fuente: autor

Ilustración 71 Registro de oficio recibido - módulo oficios
Fuente: autor

Ilustración 72 Consultar procesos - módulo oficios
Fuente: autor

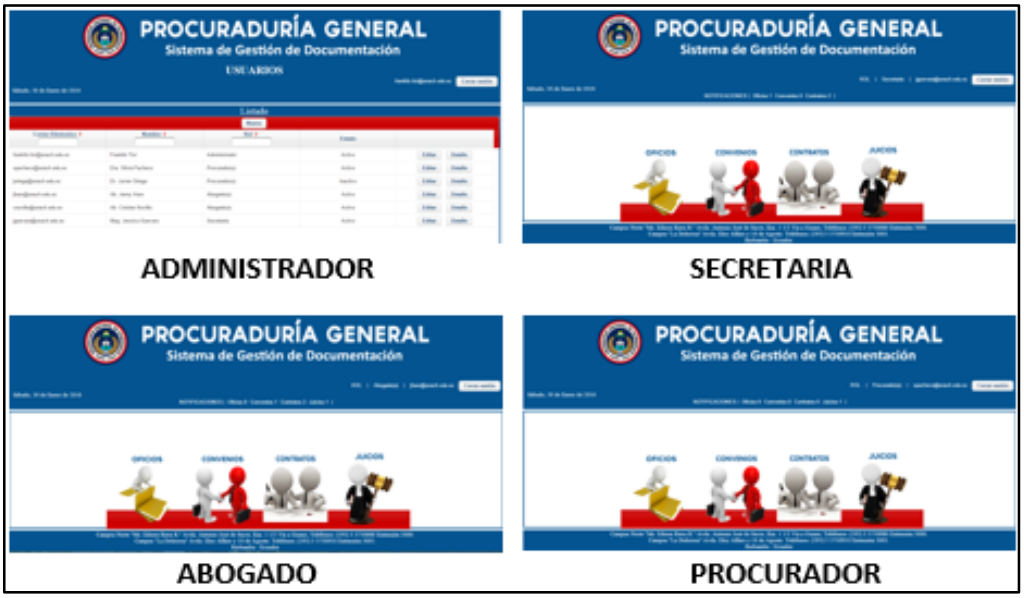
Iteración 1

Tabla 50 Iteración 1 - Historia 1

HISTORIA DE USUARIO	
Numero:1	**Usuario:** Administrador, Procurador, Abogado y Secretaria.
Nombre historia: Gestión de Acceso de Usuarios.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Esfuerzo: Medio	**Iteración asignada:** 1
Programador responsable: Ismael Franklin Tixi Cuzco	
Descripción: Antes de iniciar el sistema se requiere el correo electrónico y la contraseña para poder acceder a los módulos de acuerdo al rol de usuario.	

Observaciones: Hay cuatro roles de usuarios: Administrador, Procurador, Abogado y Secretaria, con distintos menús y permisos de acceso dependiendo de las funciones de los usuarios.

Gestión de Acceso de Usuarios



Fuente: Autor

Tabla 51 Iteración 1 - Historia 2

HISTORIA DE USUARIO	
Numero:2	**Usuario:** Procurador, Abogado y Secretaria.
Nombre historia: Gestión de Proceso de Oficios	
Prioridad en negocio: Medio	**Riesgo en desarrollo:** Medio
Esfuerzo: Medio	**Iteración asignada:** 1
Programador responsable: Ismael Franklin Tixi Cuzco	
Descripción: En este módulo se gestiona los oficios recibido por los diferentes departamentos dentro y fuera de la Universidad Nacional de Chimborazo	
Observaciones: Para cada oficio registrado en el sistema hay la opción de enviar un oficio como respuesta. La secretaria es la única que puede registrar un oficio recibido.	
 **Gestión de Proceso de Oficios**	

Fuente: Autor

Tabla 52 Iteración 1 - Historia 3

HISTORIA DE USUARIO	
Numero:3	**Usuario:** Procurador, Abogado y Secretaria.
Nombre historia: Gestión de Proceso de Convenios	
Prioridad en negocio: Alto	**Riesgo en desarrollo:** Alto
Esfuerzo: Alto	**Iteración asignada:** 1
Programador responsable: Ismael Franklin Tixi Cuzco	

Descripción: En este módulo se gestiona los oficios de convenios recibido por las diferentes instituciones de la ciudad o país.
Observaciones: En el momento en se aprueba el convenio se envía un oficio para registrar el convenio entre la universidad y la institución solicitante.
 **Gestión de Proceso de Convenios**

Fuente: Autor

Tabla 53 Iteración 1 - Historia 4

HISTORIA DE USUARIO	
Numero:4	**Usuario:** Procurador, Abogado y Secretaria.
Nombre historia: Gestión de Proceso de Contratos	
Prioridad en negocio: Alto	**Riesgo en desarrollo:** Alto
Esfuerzo: Alto	**Iteración asignada:** 1
Programador responsable: Ismael Franklin Tixi Cuzco	
Descripción: En este módulo se gestiona los oficios de contratos recibido para adquisición de equipos o servicios para la universidad.	
Observaciones: se debe cumplir los requisitos para firma el contrato.	
 **Gestión de Proceso de Contratos**	

Fuente: Autor

Tabla 54 Iteración 1 - Historia 5

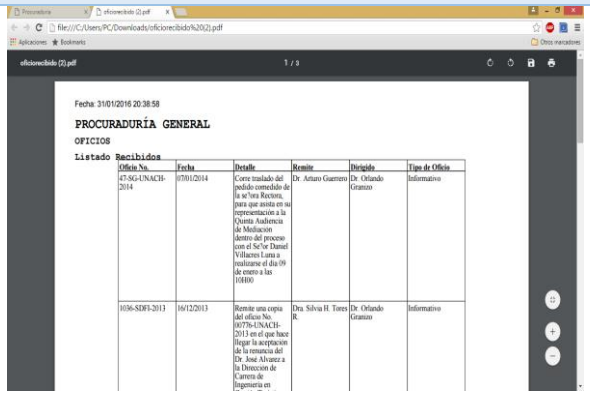
HISTORIA DE USUARIO	
Numero:5	**Usuario:** Procurador, Abogado y Secretaria.
Nombre historia: Gestión de Proceso de Juicios	
Prioridad en negocio: Medio	**Riesgo en desarrollo:** Medio
Esfuerzo: Medio	**Iteración asignada:** 1
Programador responsable: Ismael Franklin Tixi Cuzco	
Descripción: En este módulo se registra los juicios en que está inmersa la universidad.	
Observaciones: el abogado registra el seguimiento de los juicios.	
 **Gestión de Proceso de Juicios**	

Fuente: Autor

Iteración 2

Tabla 55 Interacción 2 - Historia 1

HISTORIA DE USUARIO	
Numero:6	**Usuario:** El Procurador, Abogado y Secretaria.
Nombre historia: Emisión de Reportes.	
Prioridad en negocio: Medio	**Riesgo en desarrollo:** Medio
Esfuerzo: Medio	**Iteración asignada:** 2
Programador responsable: Ismael Franklin Tixi Cuzco	
Descripción: El Procurador, Abogado y Secretaria podrán exportar los reportes en pdf.	

Observaciones: una vez exportado los reportes en pdf se pueden imprimir.
 **Emisión de Reportes**

Fuente: Autor

4.1.3.4 CÓDIGO FUENTE

El código fuente se lo ha tomado de gestionar oficios, puesto que todas y cada una de las clases, interface, dao, controlador y paginas xhtml son similares en estructura. **(Ver anexo 1, 2, 3, 4, 5)**

4.1.4 PRUEBAS

Las pruebas son muy importantes y son creadas a partir de las historias de los usuarios, el usuario debe especificar los aspectos que se van a probar cuando una historia de usuario ha sido correctamente realizada, esta prueba de usuario puede tener una o más pruebas de aceptación, las que sean necesarias para garantizar el correcto funcionamiento.

Cada prueba que se realice representa una salida esperada del sistema. Una historia de usuario no se considera completa hasta que se realicen todas las pruebas de aceptación necesarias.

A continuación se muestra las pruebas realizadas a cada una de las historias de los usuarios del sistema con su respectiva tabla de pruebas.

Historia 1

Tabla 56 Pruebas Historia 1

Fecha	Descripción	Autor
15/01/2016	Pruebas	Ismael Franklin Tixi
Gestión de Acceso a Usuarios		
Descripción	Hay cuatro tipos de usuarios: Administrador, Procurador, Abogado y Secretaria.	
Condiciones de Ejecución.	Cada uno de los usuarios mencionados debe constar en la base de datos y tener asignado un rol y permisos de acceso a los módulos y menús dependiendo de las funciones que le corresponden.	
Entrada	• El usuario ingresa su correo electrónico y contraseña • El proceso de control de acceso a Usuarios finaliza.	
Resultado Esperado	Después de ingresar su correo electrónico y su contraseña, debe mostrarse automáticamente la página de inicio del sistema con los menús asignados para cada tipo de usuario.	
Evaluación de la Prueba	**Exitosa** 	**Fallida** 

Fuente: Autor

Historia 2

Tabla 57 Pruebas Historia 2

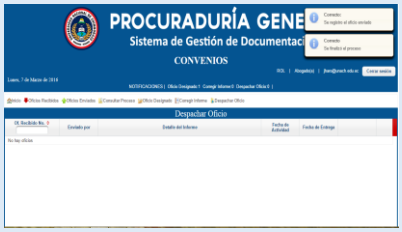
Fecha	Descripción	Autor
15/01/2016	Pruebas	Ismael Franklin Tixi
Gestión de Proceso de Oficios		
Descripción	Hay cuatro tipos de usuarios: Procurador, Abogado y Secretaria.	
Condiciones de Ejecución.	Cada uno de los usuarios mencionados debe constar en la base de datos y tener asignado un rol y permisos de acceso a los módulos y menús dependiendo de las funciones que le corresponden.	

Entrada	• Secretaria ingresa el oficio para ser resuelto • Procurador designa el oficio a Abogado • Abogado resuelve el oficio en proceso	
Resultado Esperado	Secretaria registra un oficio enviado como respuesta de la gestión del oficio recibido.	
Evaluación de la Prueba	**Exitosa** 	**Fallida** 

Fuente: Autor

Historia 3

Tabla 58 Pruebas Historia 3

Fecha	Descripción	Autor
15/01/2016	Pruebas	Ismael Franklin Tixi
Gestión de Proceso de Convenios		
Descripción	Hay tres tipos de usuarios: Procurador, Abogado y Secretaria.	
Condiciones de Ejecución.	Cada uno de los usuarios mencionados debe constar en la base de datos y tener asignado un rol y permisos de acceso a los módulos y menús dependiendo de las funciones que le corresponden.	
Entrada	• Secretaria ingresa el oficio de convenio para ser resuelto • Procurador designa el oficio de convenio a Abogado • Abogado resuelve el oficio de convenio en proceso	
Resultado Esperado	Abogado registra un oficio enviado como respuesta de la gestión del oficio de convenio recibido.	
Evaluación de la Prueba	**Exitosa** 	**Fallida** 

Fuente: Autor

Historia 4

Tabla 59 Pruebas Historia 4

Fecha	Descripción	Autor
15/01/2016	Pruebas	Ismael Franklin Tixi
Gestión de Proceso de Contratos		
Descripción	Hay tres tipos de usuarios: Procurador, Abogado y Secretaria.	
Condiciones de Ejecución.	Cada uno de los usuarios mencionados debe constar en la base de datos y tener asignado un rol y permisos de acceso a los módulos y menús dependiendo de las funciones que le corresponden.	
Entrada	• Secretaria ingresa el oficio de contrato para ser resuelto • Procurador designa el oficio a Abogado • Abogado resuelve el oficio de contrato en proceso	
Resultado Esperado	Abogado registra el contrato realizado durante el proceso y registra dos oficios enviados como respuesta de la gestión del oficio de contrato recibido.	
Evaluación de la Prueba	**Exitosa** 	**Fallida** 

Fuente: Autor

Historia 5

Tabla 60 Pruebas Historia 5

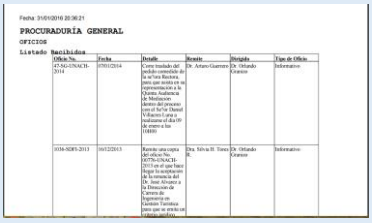
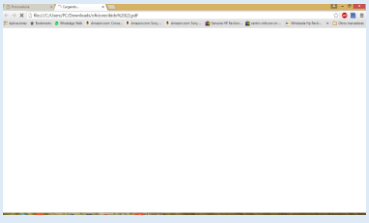
Fecha	Descripción	Autor
15/01/2016	Pruebas	Ismael Franklin Tixi
Gestión de Proceso de Juicios		
Descripción	Hay tres tipos de usuarios: Procurador, Abogado y Secretaria.	
Condiciones de Ejecución.	Cada uno de los usuarios mencionados debe constar en la base de datos y tener asignado un rol y permisos de acceso a los módulos y menús dependiendo de las funciones que le corresponden.	

Entrada	• Abogado registra el juicio para su seguimiento • Procurador designa el juicio a Abogado • Abogado da seguimiento al juicio para hasta su finalización	
Resultado Esperado	Abogado registra las boletas y la finalización del juicio	
Evaluación de la Prueba	**Exitosa** 	**Fallida** 

Fuente: Autor

Historia 6

Tabla 61 Pruebas Historia 6

Fecha	Descripción	Autor
15/01/2016	Pruebas	Ismael Franklin Tixi
Emisión de Reportes		
Descripción	El Procurador, Abogado o Secretaria dependiendo de la actividad que requiera podrán exportar un reporte.	
Condiciones de Ejecución	El Procurador, Abogado o Secretaria debe constar en la base de datos del sistema para poder generar un reporte.	
Entrada	• El Procurador, Abogado o Secretaria una vez que ingresa al sistema escoge la opción de descargar pdf.	
Resultado Esperado	Tras seleccionar la acción que requieren se exporta exitosamente el reporte.	
Evaluación de la Prueba	**Exitosa** 	**Fallida** 

Fuente: Autor

CONCLUSIONES

- La tecnología ORM-Hibernate utiliza el mapeo objeto/relacional como su nombre lo dice, esto permite crear las clases o pojo de cada tabla de la base de datos en poco tiempo. Esta tecnología usa SessionFactory la cual se encarga de indicar al sistema, donde se encuentran todos los archivos de mapeo de Hibernate, además emplea session que no son más que objetos de corta vida que se utilizan para realizar conexiones a la base de datos y posee su propio lenguaje de consulta de datos denominado HQL; todo esto permite obtener extensibilidad, escalabilidad y eficiencia en el desarrollo y mantenimiento de aplicaciones web.

- Durante el estudio se determinó que la tecnología ORM-Hibernate mejora el desarrollo de aplicaciones web con respecto a la productividad enfocado al acceso de datos en tiempo empleado y líneas de código utilizadas gracias a la herramienta LinesOfCodeWichtel y al registro del número de horas empleadas.

- Con el desarrollo del Sistema de Gestión de Documentación Fénix se ha logrado automatizar los procesos de gestión de documentación que anteriormente se hacían de forma semiautomática (Microsoft Excel), además permite a los usuarios la adecuada organización de la información de las actividades realizadas durante los procesos.

- La mayor parte de los problemas que se suscitaron en el transcurso de este proyecto investigativo, se relacionaron con la falta de fuentes de información o documentación en español.

- Para ampliar la fundamentación de la presente investigación se consideró realizar encuestas a expertos en el desarrollo de aplicaciones web permitió determinar que utilizan la tecnología ORM-Hibernate por su mayor productividad.

RECOMENDACIONES

- Para el desarrollo de las aplicaciones web de forma eficiente se recomienda utilizar la tecnología ORM-Hibernate porque permite una mayor productividad enfocado al acceso de datos, gracias a la tecnología mapeo objeto/relacional este permite la creación de las clases de las tablas de la base de datos en segundos y el tratamiento de los objetos con los métodos save, update, delete y el lenguaje de consulta HQL.

- Se sugiere un análisis de las diversas versiones de ORM-Hibernate con respecto a la compatibilidad de las mismas, ya que algunos componentes no funciones correctamente en versiones superiores o viceversa en la presente investigación se utilizó framework Hibernate 4.2.6.

- Se recomienda la utilización de la herramienta LinesOfCodeWichtel para cuantificar las líneas de código empleadas para el desarrollo de las aplicaciones web por su facilidad del manejo y su claridad de la información.

- Se propone el análisis comparativo entre las tecnologías Mapeo Objeto/Relacional (ORM) versus Java Persistence AP (JPA) con respectó al rendimiento.

BIBLIOGRAFÍA

Cando Cando, O. A. (26 de 01 de 2016). *http://www.espoch.edu.ec/*. Obtenido de http://dspace.espoch.edu.ec/bitstream/123456789/2427/1/18T00501.pdf

Carmen Gonzales. (21 de 01 de 2016). *http://java.sys-con.com/*. Obtenido de http://java.sys-con.com/node/140123

Hibernate. (21 de 01 de 2016). *Hibernate.org*. Obtenido de http://docs.jboss.org/hibernate/orm/4.2/manual/en-US/html_single/

Java. (28 de 01 de 2016). *https://www.java.com*. Obtenido de https://www.java.com/es/about/

Payá Martín, A. (26 de 01 de 2016). *http://www.comillas.edu/*. Obtenido de http://www.iit.upcomillas.es/pfc/resumenes/450955e7368ca.pdf

Primefaces. (28 de 01 de 2016). *http://www.primefaces.org/*. Obtenido de http://www.primefaces.org/

programacionextrema. (28 de 01 de 2016). *http://programacionextrema.tripod.com*. Obtenido de http://programacionextrema.tripod.com/fases.htm

Reyes Freire, T. A. (26 de 01 de 2016). *http://www.utn.edu.ec/*. Obtenido de http://repositorio.utn.edu.ec/bitstream/123456789/571/1/Tesis.pdf

UCE. (28 de 01 de 2016). *http://www.uce.edu.ec/*. Obtenido de http://www.dspace.uce.edu.ec/bitstream/25000/5177/1/T-UCE-0011-214.pdf

Unam. (20 de 02 de 2016). *www.unam.mx*. Obtenido de http://profesores.fi-b.unam.mx/sun/Downloads/Java/jdbc.pdf

ANEXOS

ANEXO 1: CLASE

```
package Pojo;
// Generated 09/01/2016 0:08:28 by Hibernate Tools 3.6.0
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

/**
 * Oficiorecibido generated by hbm2java
 */
public class Oficiorecibido  implements java.io.Serializable {
     private Integer codigoOficioRecibido;
     private Tipooficio tipooficio;
     private String numero;
     private Date fecha;
     private String detalle;
     private String remite;
     private String dirigido;
     private Date fecharegistro;
     private Set seguimientooficios = new HashSet(0);
     private Set oficioenviados = new HashSet(0);

    public Oficiorecibido() {
    }

    public Oficiorecibido(Tipooficio tipooficio, String numero, Date fecha, String
detalle, String remite, String dirigido, Date fecharegistro) {
        this.tipooficio = tipooficio;
        this.numero = numero;
        this.fecha = fecha;
        this.detalle = detalle;
        this.remite = remite;
        this.dirigido = dirigido;
        this.fecharegistro = fecharegistro;
    }
    public Oficiorecibido(Tipooficio tipooficio, String numero, Date fecha, String
detalle, String remite, String dirigido, Date fecharegistro, Set seguimientooficios,
Set oficioenviados) {
       this.tipooficio = tipooficio;
       this.numero = numero;
       this.fecha = fecha;
       this.detalle = detalle;
       this.remite = remite;
       this.dirigido = dirigido;
       this.fecharegistro = fecharegistro;
       this.seguimientooficios = seguimientooficios;
       this.oficioenviados = oficioenviados;
```

```
}

public Integer getCodigoOficioRecibido() {
   return this.codigoOficioRecibido;
}

public void setCodigoOficioRecibido(Integer codigoOficioRecibido) {
   this.codigoOficioRecibido = codigoOficioRecibido;
}
public Tipooficio getTipooficio() {
   return this.tipooficio;
}

public void setTipooficio(Tipooficio tipooficio) {
   this.tipooficio = tipooficio;
}
public String getNumero() {
   return this.numero;
}

public void setNumero(String numero) {
   this.numero = numero;
}
public Date getFecha() {
   return this.fecha;
}

public void setFecha(Date fecha) {
   this.fecha = fecha;
}
public String getDetalle() {
   return this.detalle;
}

public void setDetalle(String detalle) {
   this.detalle = detalle;
}
public String getRemite() {
   return this.remite;
}

public void setRemite(String remite) {
   this.remite = remite;
}
public String getDirigido() {
   return this.dirigido;
}
```

```
    public void setDirigido(String dirigido) {
        this.dirigido = dirigido;
    }
    public Date getFecharegistro() {
        return this.fecharegistro;
    }

    public void setFecharegistro(Date fecharegistro) {
        this.fecharegistro = fecharegistro;
    }
    public Set getSeguimientooficios() {
        return this.seguimientooficios;
    }

    public void setSeguimientooficios(Set seguimientooficios) {
        this.seguimientooficios = seguimientooficios;
    }
    public Set getOficioenviados() {
        return this.oficioenviados;
    }

    public void setOficioenviados(Set oficioenviados) {
        this.oficioenviados = oficioenviados;
    }

}
```

ANEXO 2: MAPEOS DE XML

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<!-- Generated 09/01/2016 0:08:31 by Hibernate Tools 3.6.0 -->
<hibernate-mapping>
    <class name="Pojo.Oficiorecibido" table="oficiorecibido"
catalog="bdprocuraduria">
        <id name="codigoOficioRecibido" type="java.lang.Integer">
            <column name="codigoOficioRecibido" />
            <generator class="identity" />
        </id>
        <many-to-one name="tipooficio" class="Pojo.Tipooficio" fetch="select">
            <column name="codigoTipoOficio" not-null="true" />
        </many-to-one>
        <property name="numero" type="string">
            <column name="numero" length="50" not-null="true" />
```

```
        </property>
        <property name="fecha" type="date">
            <column name="fecha" length="10" not-null="true" />
        </property>
        <property name="detalle" type="string">
            <column name="detalle" length="500" not-null="true" />
        </property>
        <property name="remite" type="string">
            <column name="remite" length="50" not-null="true" />
        </property>
        <property name="dirigido" type="string">
            <column name="dirigido" length="50" not-null="true" />
        </property>
        <property name="fecharegistro" type="timestamp">
            <column name="fecharegistro" length="19" not-null="true" />
        </property>
        <set name="seguimientooficios" table="seguimientooficio" inverse="true"
lazy="true" fetch="select">
            <key>
                <column name="codigoOficioRecibido" not-null="true" />
            </key>
            <one-to-many class="Pojo.Seguimientooficio" />
        </set>
        <set name="oficioenviados" table="oficioenviado" inverse="true"
lazy="true" fetch="select">
            <key>
                <column name="codigoOficioRecibido" not-null="true" />
            </key>
            <one-to-many class="Pojo.Oficioenviado" />
        </set>
    </class>
</hibernate-mapping>
```

ANEXO 3: INTERFACE

```
package Interface;

import Pojo.Oficiorecibido;
import java.util.List;
import org.hibernate.Session;

/**
 *
 * @author PC
 */
public interface InterfaceOficioRecibido {
```

```
   public boolean register (Session session, Oficiorecibido ObjOficioRecibido) throws Exception;
   public List<Oficiorecibido> getAll(Session session)throws Exception;
   public Oficiorecibido getByCodigoOficioRecibido(Session session,Integer codigoOficioRecibido) throws Exception;
   public Oficiorecibido getByNumeroOficioRecibido(Session session,String numeroOficioRecibido) throws Exception;
   public boolean update(Session session,Oficiorecibido ObjOficioRecibido) throws Exception;
   public Oficiorecibido getUltimoRegistro(Session session) throws Exception;
   public List<Oficiorecibido> getByNumeroOficioRecibidoEnviado(Session session,String numeroOficioRecibidoEnviado) throws Exception;

}
```

ANEXO 4: DAO

```
package Dao;
import Interface.InterfaceOficioRecibido;
import Pojo.Oficiorecibido;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
/**
 *
 * @author PC
 */
public class DaoOficioRecibido implements InterfaceOficioRecibido {

   @Override
   public boolean register(Session session, Oficiorecibido ObjOficioRecibido) throws Exception{
      session.save(ObjOficioRecibido);
      return true;
   }

   @Override
   public List<Oficiorecibido> getAll(Session session) throws Exception {
      String hql = "from Oficiorecibido as oficiorecibido inner join fetch oficiorecibido.tipooficio "
            + "as Tipooficio order by codigoOficiorecibido desc";
      Query query = session.createQuery(hql);
      // va a retorda todos los registro
      List<Oficiorecibido> listaOficioRecibido = (List<Oficiorecibido>) query.list();

      return listaOficioRecibido;
```

```
    }

    @Override
    public Oficiorecibido getByCodigoOficioRecibido(Session session, Integer
codigoOficioRecibido) throws Exception {
        String hql = "from Oficiorecibido as oficiorecibido inner join fetch
oficiorecibido.tipooficio "
                + "as Tipooficio where codigoOficioRecibido=:codigoOficioRecibido";
        Query query = session.createQuery(hql);
        query.setParameter("codigoOficioRecibido", codigoOficioRecibido);
        return (Oficiorecibido) query.uniqueResult();
    }

    @Override
    public boolean update(Session session, Oficiorecibido ObjOficioRecibido)
throws Exception {
        session.update(ObjOficioRecibido);
        return true;
    }

    @Override
    public Oficiorecibido getByNumeroOficioRecibido(Session session, String
numeroOficioRecibido) throws Exception {
        String hql="from Oficiorecibido where numero=:numero";
        Query query=session.createQuery(hql);
        query.setParameter("numero", numeroOficioRecibido);
        Oficiorecibido ObjOficiorecibido=(Oficiorecibido) query.uniqueResult();
        return ObjOficiorecibido;
    }

    @Override
    public Oficiorecibido getUltimoRegistro(Session session) throws Exception {
        String hql="from Oficiorecibido order by codigoOficioRecibido desc";
        Query query=session.createQuery(hql).setMaxResults(1);
        return (Oficiorecibido) query.uniqueResult();
    }

    @Override
    public List<Oficiorecibido> getByNumeroOficioRecibidoEnviado(Session
session, String numeroOficioRecibidoEnviado) throws Exception {
        String hql = "from Oficiorecibido where numero=:numero";
        Query query = session.createQuery(hql);
        query.setParameter("numero", numeroOficioRecibidoEnviado);
        List<Oficiorecibido> listaOficioRecibido = (List<Oficiorecibido>)
query.list();
        return listaOficioRecibido;
    }
}
```

ANEXO 5: CONTROLADOR

```
package BeanView;
import BeanSession.MbSLogin;
import Dao.DaoEstadoOficio;
import Dao.DaoFaseOficio;
import Dao.DaoOficioRecibido;
import Dao.DaoSeguimientoOficio;
import Dao.DaoTipoOficio;
import Dao.DaoUsuario;
import Pojo.Estadooficio;
import Pojo.Faseoficio;
import Pojo.Oficiorecibido;
import Pojo.Seguimientooficio;
import Pojo.Tipooficio;
import Pojo.Usuario;
import Util.HibernateUtil;
import com.lowagie.text.BadElementException;
import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Font;
import com.lowagie.text.FontFactory;
import com.lowagie.text.PageSize;
import com.lowagie.text.Paragraph;
import com.lowagie.text.Phrase;
import java.awt.Color;
import java.io.IOException;
import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.ViewScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpSession;
import org.hibernate.Session;
import org.hibernate.Transaction;

/**
 *
 * @author PC
 */
@ManagedBean(name = "mbVOficioRecibido")
@ViewScoped
public class MbVOficioRecibido implements Serializable {
    private Seguimientooficio ObjSeguimientooficio;
```

```
private Oficiorecibido ObjOficiorecibido;
private Oficiorecibido OficiorecibidoSel;
private List<Oficiorecibido> listaOficioRecibido;
private List<Oficiorecibido> listaOficioRecibidos;
private List<Oficiorecibido> listaOficioRecibidoFiltrar;
private Tipooficio TipooficioSel;
private Faseoficio FaseoficioSel;
private Estadooficio EstadooficioSel;
private int ValorTipoOficio;
private List<Tipooficio> listaTipoOficio;
private int numeroOficioRecibido;
private int ValorUsuario;
private int ValorFaseOficio;
private int ValorEstadoOficio;
private int ValorOficioRecibido;
private List<Usuario> listaUsuario;
private Usuario UsuarioSel;
private List<Faseoficio> listaFaseOficio;
private List<Estadooficio> listaEstadoOficio;
private Session session;
private Transaction transaction;
@ManagedProperty("#{mbSLogin}")
private MbSLogin mbSLogin;

/**
 * Creates a new instance of MbVOficioRecibido
 */
public MbVOficioRecibido() {
   this.ObjOficiorecibido = new Oficiorecibido();
   this.ObjSeguimientooficio=new Seguimientooficio();
}

public List<Tipooficio> getAllcargarTipoOficioActivo() {
   this.session = null;
   this.transaction = null;

   try {
      DaoTipoOficio daoTipoOficio = new DaoTipoOficio();
      this.session = HibernateUtil.getSessionFactory().openSession();
      this.transaction = this.session.beginTransaction();
      this.listaTipoOficio = daoTipoOficio.getAllActivo(this.session);
      this.transaction.commit();
      return this.listaTipoOficio;
   } catch (Exception ex) {
      if (this.transaction != null) {
         this.transaction.rollback();
      }
```

```
            FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_FATAL, "Error Fatal", "Por favor
contante con su administrador" + ex.getMessage()));
            return null;
        } finally {
            if (this.session != null) {
                this.session.close();
            }
        }
    }

    /// Para registrar el oficio recibido
    public void register() {
        this.session = null;
        this.transaction = null;
        try {

            DaoOficioRecibido daoOficiorecibido = new DaoOficioRecibido();
            DaoTipoOficio daoTipoOficio = new DaoTipoOficio();
            DaoSeguimientoOficio daoSeguimientoOficio = new
DaoSeguimientoOficio();
            DaoUsuario daoUsuario = new DaoUsuario();
            DaoFaseOficio daoFaseOficio = new DaoFaseOficio();
            DaoEstadoOficio daoEstadoOficio = new DaoEstadoOficio();
            this.session = HibernateUtil.getSessionFactory().openSession();
            this.transaction = this.session.beginTransaction();
            //Para asignar el tipo oficio
            this.TipooficioSel = daoTipoOficio.getByCodigoTipoOficio(this.session,
ValorTipoOficio);
            this.ObjOficiorecibido.setTipooficio(this.TipooficioSel);
            this.ObjOficiorecibido.setFecharegistro(new Date());
            if (daoOficiorecibido.getByNumeroOficioRecibido(this.session,
this.ObjOficiorecibido.getNumero()) != null) {
                FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_ERROR, "Error:", "El número de
oficio ya se encuentra registrado, elija otro número por favor"));
                return;
            }

            daoOficiorecibido.register(this.session, this.ObjOficiorecibido);
            this.transaction.commit();
            FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Correcto:", "Se registro el
oficio"));

            //// ingresar el seguimiento de oficio secretaria
            ///es obligatorio el open session para la creacion de otro transaction
            this.session = HibernateUtil.getSessionFactory().openSession();
```

```
        this.transaction = session.beginTransaction();
        this.ObjSeguimientooficio.setFecharegistro(new Date());
        this.ObjSeguimientooficio.setEstado(1);
        /// Para ingresar usuario
        HttpSession sessionUsuario=(HttpSession)
FacesContext.getCurrentInstance().getExternalContext().getSession(true);

this.ObjSeguimientooficio.setUsuario(daoUsuario.getByCorreoElectronico(this.se
ssion, sessionUsuario.getAttribute("correoElectronico").toString()));
        //Para asignar la fase
        this.FaseoficioSel = daoFaseOficio.getByCodigoFaseOficio(this.session,
1);
        ObjSeguimientooficio.setFaseoficio(this.FaseoficioSel);
        //Para asignar el estado
        this.EstadooficioSel =
daoEstadoOficio.getByCodigoEstadoOficio(this.session, 1);
        ObjSeguimientooficio.setEstadooficio(this.EstadooficioSel);
        //Para asignar el oficio recibido
        ObjOficiorecibido = daoOficiorecibido.getUltimoRegistro(this.session);
        ObjSeguimientooficio.setOficiorecibido(this.ObjOficiorecibido);
        daoSeguimientoOficio.register(this.session, this.ObjSeguimientooficio);
        this.transaction.commit();
        //FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Correcto:", "El proceso del
oficio se inicio"));

        /// Seguimiento para enviar al Procurador(a)
        ///es obligatorio para crear otra transaccion --
transactionHibernateUtil.getSessionFactory().openSession();--
        this.session = HibernateUtil.getSessionFactory().openSession();
        this.transaction = session.beginTransaction();
        this.ObjSeguimientooficio.setFecharegistro(new Date());
        this.ObjSeguimientooficio.setEstado(0);

this.ObjSeguimientooficio.setEnviadoUsuario(this.ObjSeguimientooficio.getUsua
rio().getNombreCompleto());
        /// Para ingresar usuario
        this.UsuarioSel = daoUsuario.getByCodigoUsuario(this.session,
ValorUsuario);
        ObjSeguimientooficio.setUsuario(this.UsuarioSel);
        //Para asignar la fase
        this.FaseoficioSel = daoFaseOficio.getByCodigoFaseOficio(this.session,
2);
        ObjSeguimientooficio.setFaseoficio(this.FaseoficioSel);
        //Para asignar el estado
        this.EstadooficioSel =
daoEstadoOficio.getByCodigoEstadoOficio(this.session, 3);
        ObjSeguimientooficio.setEstadooficio(this.EstadooficioSel);
```

```
            //Para asignar el oficio recibido
            ObjOficiorecibido = daoOficiorecibido.getUltimoRegistro(this.session);
            ObjSeguimientooficio.setOficiorecibido(this.ObjOficiorecibido);
            daoSeguimientoOficio.register(this.session, this.ObjSeguimientooficio);
            this.transaction.commit();
            FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_INFO, "Correcto:", "Se envio el oficio
a Procurador(a)"));
            this.ObjOficiorecibido = new Oficiorecibido();
            this.ValorTipoOficio = 0;
            this.ValorUsuario = 0;
            this.ValorFaseOficio = 0;
            this.ValorOficioRecibido = 0;
            this.ValorEstadoOficio = 0;
            this.ObjSeguimientooficio.setObservacion("");
        } catch (Exception ex) {
            if (this.transaction != null) {
                transaction.rollback();
            }
            FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_FATAL, "Error fatal:", "Por favor
contacte con su administrador " + ex.getMessage()));
        } finally {
            if (this.session != null) {
                this.session.close();
            }
        }
    }

    public List<Oficiorecibido> getAll() {
        this.session = null;
        this.transaction = null;

        try {
            DaoOficioRecibido daoOficioRecibido = new DaoOficioRecibido();
            this.session = HibernateUtil.getSessionFactory().openSession();
            this.transaction = this.session.beginTransaction();
            this.listaOficioRecibido = daoOficioRecibido.getAll(this.session);
            this.transaction.commit();
            return this.listaOficioRecibido;
        } catch (Exception ex) {
            if (this.transaction != null) {
                this.transaction.rollback();
            }
            FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_FATAL, "Error Fatal", "Por favor
contante con su administrador" + ex.getMessage()));
            return null;
```

```
        } finally {
            if (this.session != null) {
                this.session.close();
            }
        }

    }


    public void preProcessPDF(Object document) throws IOException,
BadElementException, DocumentException {
        Document pdf = (Document) document;
        pdf.setPageSize(PageSize.A4.rotate());
        pdf.open();
        SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss");
        pdf.add(new Phrase("Fecha: " + formato.format(new Date())));
        pdf.add(new Paragraph("PROCURADURÍA GENERAL",
FontFactory.getFont(FontFactory.COURIER, 20, Font.BOLD, new Color(0, 0,
0))));
        pdf.add(new Paragraph("OFICIOS",
FontFactory.getFont(FontFactory.COURIER, 15, Font.BOLD, new Color(0, 0,
0))));
        pdf.add(new Paragraph("Listado Recibidos",
FontFactory.getFont(FontFactory.COURIER, 15, Font.BOLD, new Color(0, 0,
0))));
    }


     public List<Usuario> getAllcargarProcurador() {
        this.session = null;
        this.transaction = null;

        try {
            DaoUsuario daoUsuario = new DaoUsuario();
            this.session = HibernateUtil.getSessionFactory().openSession();
            this.transaction = this.session.beginTransaction();
            this.listaUsuario = daoUsuario.getAllUsuarioRol(this.session,2);
            this.transaction.commit();
            return this.listaUsuario;

        } catch (Exception ex) {
            if (this.transaction != null) {
                this.transaction.rollback();

            }
```

```
            FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_FATAL, "Error Fatal", "Por favor
contante con su administrador" + ex.getMessage()));
            return null;
        } finally {
            if (this.session != null) {
                this.session.close();
            }
        }

    }


    /// Metodos get y set
    public Seguimientooficio getObjSeguimientooficio() {
        return ObjSeguimientooficio;
    }

    public void setObjSeguimientooficio(Seguimientooficio ObjSeguimientooficio)
{
        this.ObjSeguimientooficio = ObjSeguimientooficio;
    }

    public Oficiorecibido getObjOficiorecibido() {
        return ObjOficiorecibido;
    }

    public void setObjOficiorecibido(Oficiorecibido ObjOficiorecibido) {
        this.ObjOficiorecibido = ObjOficiorecibido;
    }

    public Oficiorecibido getOficiorecibidoSel() {
        return OficiorecibidoSel;
    }

    public void setOficiorecibidoSel(Oficiorecibido OficiorecibidoSel) {
        this.OficiorecibidoSel = OficiorecibidoSel;
    }

    public List<Oficiorecibido> getListaOficioRecibido() {
        return listaOficioRecibido;
    }

    public void setListaOficioRecibido(List<Oficiorecibido> listaOficioRecibido) {
        this.listaOficioRecibido = listaOficioRecibido;
    }

    public List<Oficiorecibido> getListaOficioRecibidoFiltrar() {
```

```
        return listaOficioRecibidoFiltrar;
    }

    public void setListaOficioRecibidoFiltrar(List<Oficiorecibido>
listaOficioRecibidoFiltrar) {
        this.listaOficioRecibidoFiltrar = listaOficioRecibidoFiltrar;
    }

    public Tipooficio getTipooficioSel() {
        return TipooficioSel;
    }

    public void setTipooficioSel(Tipooficio TipooficioSel) {
        this.TipooficioSel = TipooficioSel;
    }

    public Faseoficio getFaseoficioSel() {
        return FaseoficioSel;
    }

    public void setFaseoficioSel(Faseoficio FaseoficioSel) {
        this.FaseoficioSel = FaseoficioSel;
    }

    public Estadooficio getEstadooficioSel() {
        return EstadooficioSel;
    }

    public void setEstadooficioSel(Estadooficio EstadooficioSel) {
        this.EstadooficioSel = EstadooficioSel;
    }

    public int getValorTipoOficio() {
        return ValorTipoOficio;
    }

    public void setValorTipoOficio(int ValorTipoOficio) {
        this.ValorTipoOficio = ValorTipoOficio;
    }

    public List<Tipooficio> getListaTipoOficio() {
        return listaTipoOficio;
    }

    public void setListaTipoOficio(List<Tipooficio> listaTipoOficio) {
        this.listaTipoOficio = listaTipoOficio;
    }
```

```
public int getNumeroOficioRecibido() {
   return numeroOficioRecibido;
}

public void setNumeroOficioRecibido(int numeroOficioRecibido) {
   this.numeroOficioRecibido = numeroOficioRecibido;
}

public int getValorUsuario() {
   return ValorUsuario;
}

public void setValorUsuario(int ValorUsuario) {
   this.ValorUsuario = ValorUsuario;
}

public int getValorFaseOficio() {
   return ValorFaseOficio;
}

public void setValorFaseOficio(int ValorFaseOficio) {
   this.ValorFaseOficio = ValorFaseOficio;
}

public int getValorEstadoOficio() {
   return ValorEstadoOficio;
}

public void setValorEstadoOficio(int ValorEstadoOficio) {
   this.ValorEstadoOficio = ValorEstadoOficio;
}

public int getValorOficioRecibido() {
   return ValorOficioRecibido;
}

public void setValorOficioRecibido(int ValorOficioRecibido) {
   this.ValorOficioRecibido = ValorOficioRecibido;
}

public List<Usuario> getListaUsuario() {
   return listaUsuario;
}

public void setListaUsuario(List<Usuario> listaUsuario) {
   this.listaUsuario = listaUsuario;
}
```

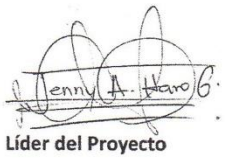
```
    public Usuario getUsuarioSel() {
        return UsuarioSel;
    }

    public void setUsuarioSel(Usuario UsuarioSel) {
        this.UsuarioSel = UsuarioSel;
    }

    public List<Faseoficio> getListaFaseOficio() {
        return listaFaseOficio;
    }

    public void setListaFaseOficio(List<Faseoficio> listaFaseOficio) {
        this.listaFaseOficio = listaFaseOficio;
    }

    public List<Estadooficio> getListaEstadoOficio() {
        return listaEstadoOficio;
    }

    public MbSLogin getMbSLogin() {
        return mbSLogin;
    }

    public void setMbSLogin(MbSLogin mbSLogin) {
        this.mbSLogin = mbSLogin;
    }

    public void setListaEstadoOficio(List<Estadooficio> listaEstadoOficio) {
        this.listaEstadoOficio = listaEstadoOficio;
    }

    public List<Oficiorecibido> getListaOficioRecibidos() {
        return listaOficioRecibidos;
    }

    public void setListaOficioRecibidos(List<Oficiorecibido>
listaOficioRecibidos) {
        this.listaOficioRecibidos = listaOficioRecibidos;
    }
}
```

ANEXO 6: ESPECIFICACIONES PARA EL SISTEMA DE GESTIÓN DE DOCUMENTACIÓN

ACTIVIDADES QUE SE DESARROLLAN EN LA PROCURADURÍA DE LA UNACH

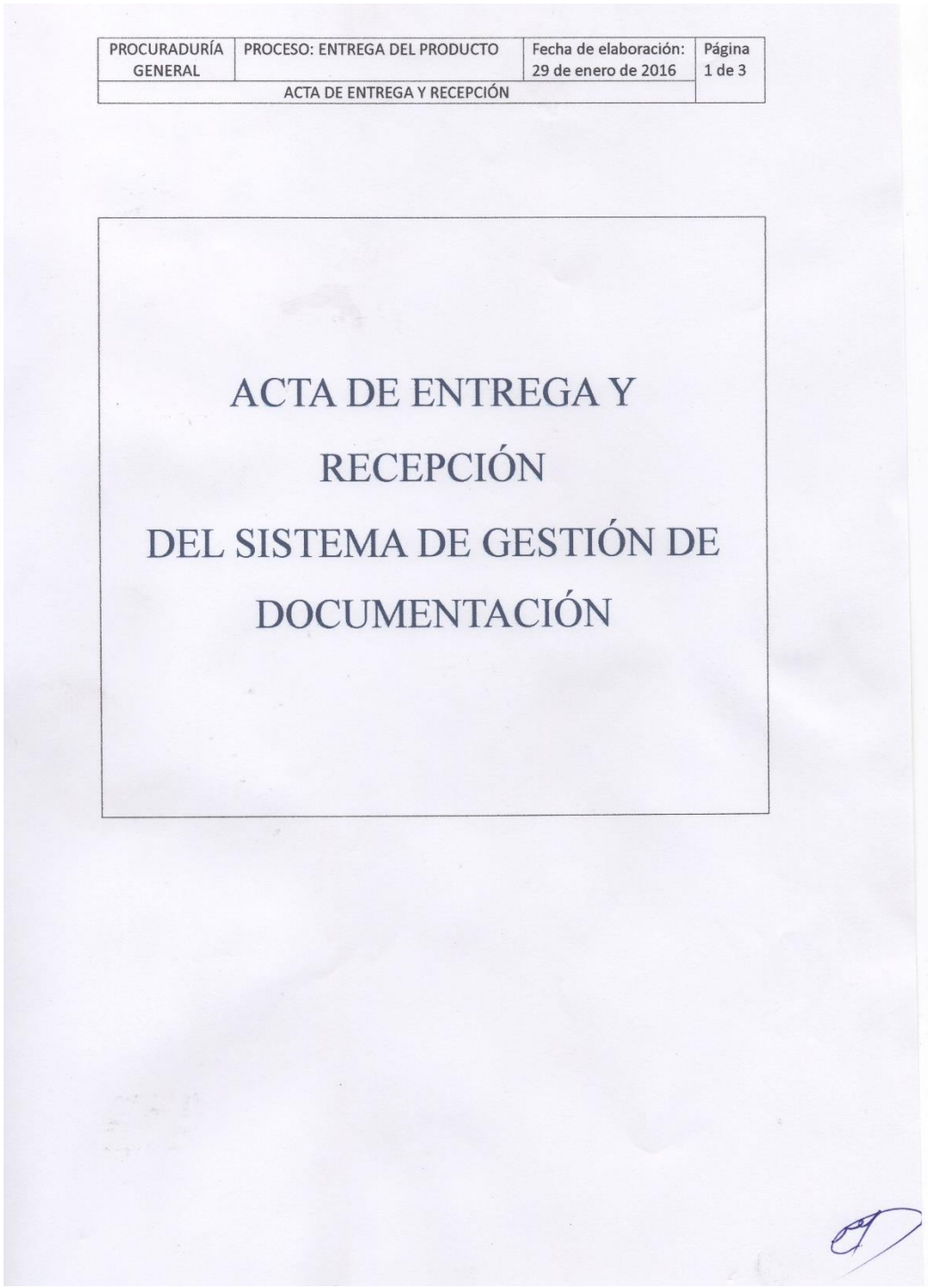
- Registro de las comunicaciones que ingresan a la Procuraduría.
- Registro de los oficios enviados desde la Procuraduría.
- Elaboración de criterios jurídicos en base a las propuestas de convenios.
- Criterios jurídicos en relación a consultas realizadas por las diferentes dependencias en los campos administrativo y académico.
- Seguimiento de los procesos judiciales en los que se encuentra inmersa la UNACH.
- Elaboración de contratos que son producto de procesos de contratación pública.
- Seguimiento post contractual.

Certifico,



Líder del Proyecto

ANEXO 7: ACTA ENTREGA-RECEPCIÓN DEL SISTEMA DE GESTIÓN DE DOCUMENTACIÓN



PROCURADURÍA GENERAL	PROCESO: ENTREGA DEL PRODUCTO	Fecha de elaboración: 29 de enero de 2016	Página 1 de 3
ACTA DE ENTREGA Y RECEPCIÓN			

ACTA DE ENTREGA Y
RECEPCIÓN
DEL SISTEMA DE GESTIÓN DE
DOCUMENTACIÓN

PROCURADURÍA GENERAL	PROCESO: ENTREGA DEL PRODUCTO	Fecha de elaboración: 29 de enero de 2016	Página 2 de 3
ACTA DE ENTREGA Y RECEPCIÓN			

Proyecto	Sistema de Gestión de Documentación
Cliente	Procuraduría General de la Universidad Nacional de Chimborazo
Representante del Cliente	Ab. Cristhian Novillo Jara
Fecha	29/01/2016

Por medio del presente le informo que el día de hoy se culmina con las siguientes fases del proyecto.

- Planificación
- Análisis
- Diseño
- Desarrollo
- Pruebas
- Puesta en Producción
- Cierre

Se realiza la entrega formal del Sistema de Gestión de Documentación, el cual contiene los siguientes módulos.

Previamente entregados:

- Oficios
- Convenios
- Contratos
- Juicios

Esta entrega se hace conforme a las especificaciones establecidas en el documento elaborado por la Ab. Jenny Haro.

Los módulos ya están instalados y funcionan en los ambientes solicitando. La revisión se efectuó en el sitio con los siguientes resultados.

MÓDULO	COD REQ	REQUERIMIENTO	ESTADO(No inicializado, En proceso, Ejecutado)
Oficios	REQ-001	Seguimiento de procesos de los oficios.	EJECUTADO
Convenios	REQ-002	Seguimiento de procesos de los convenios.	EJECUTADO
Contratos	REQ-003	Seguimiento de procesos de los contratos.	EJECUTADO
Juicios	REQ-004	Seguimientos de proceso de los juicios.	EJECUTADO

RESTRICCIONES DE USO DEL SISTEMA DE GESTIÓN DE DOCUMENTACIÓN

Este sistema solamente podrá ser utilizado en el departamento de la Procuraduría General.

Ab. Cristhian Novillo
Coordinador Procuraduría General

Ing. Danny Velasco
Director de ISYC

Ing. Diego Palacios
Docente de ISYC

Sr. Franklin Tixi
Estudiante de ISYC

ANEXO 8: HOJA DE ENCUESTA A LA EMPRESA DE DESARROLLO DE SOFTWARE CITYTECH

UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA EN SISTEMAS Y COMPUTACIÓN

Encuesta desarrollada por Franklin Tixi

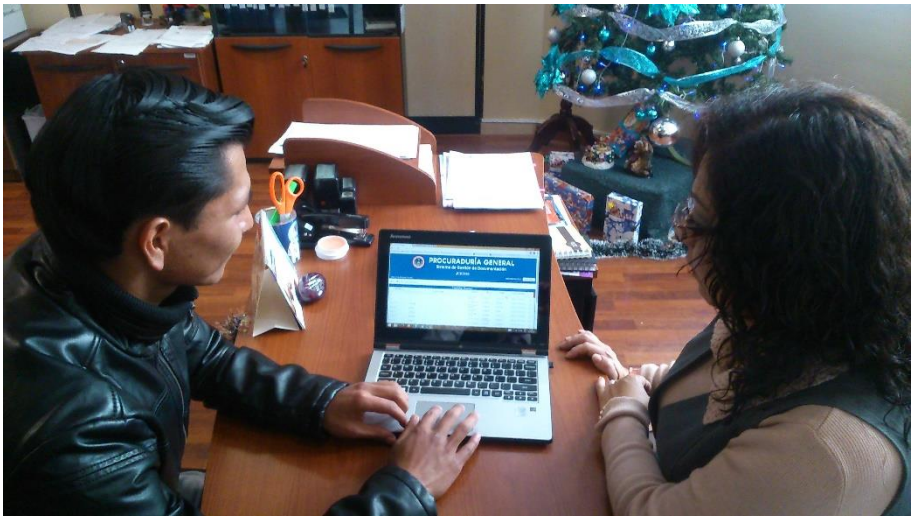
La siguiente encuesta está orientada a profesionales con experiencia en desarrollo de aplicaciones web en Java, utilizando Tecnología ORM-Hibernate y Arquitectura de desarrollo N-Capas. Tiene como intención recabar información que demuestre cuál de las dos tecnologías antes mencionadas ofrece más productividad en el ambiente de desarrollo.

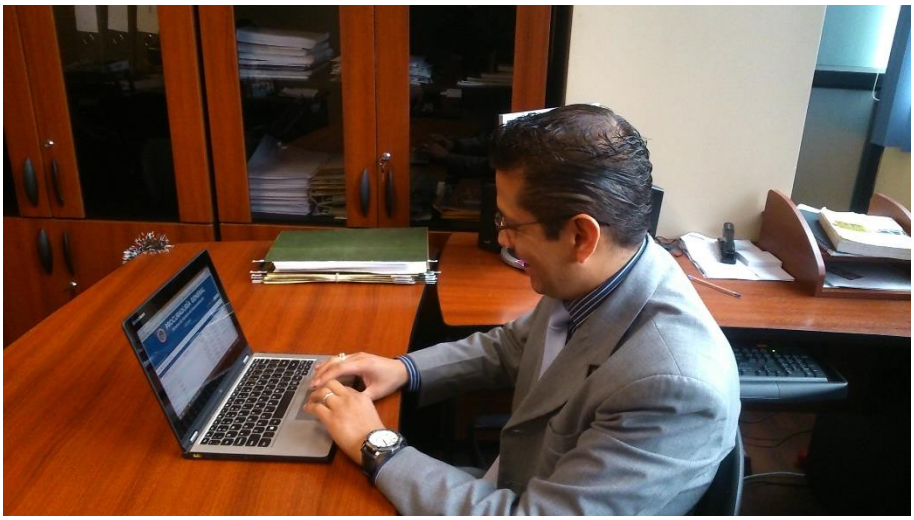
1. **¿Cuál de las siguientes tecnologías considera que utiliza menos tiempo en el desarrollo de aplicaciones web enfocado al acceso de datos?**
    - ORM-Hibernate
    - Arquitectura N-Capas

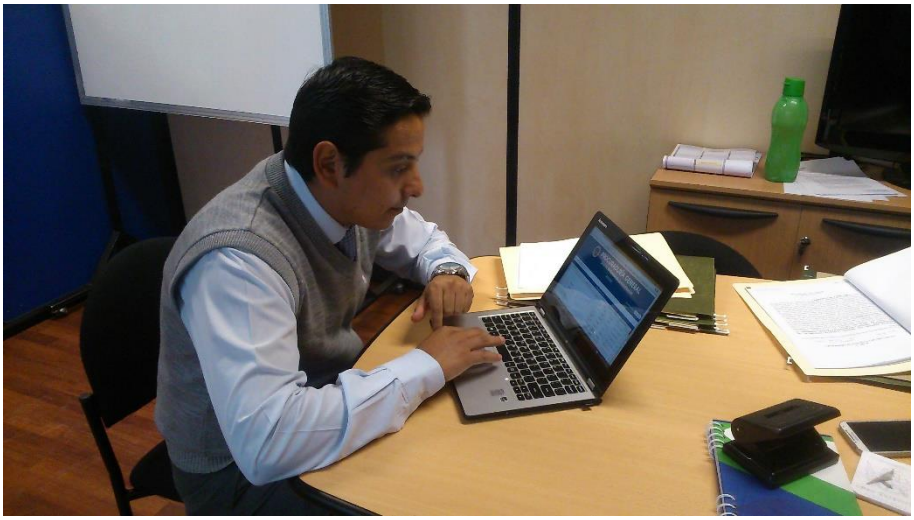
2. **¿Cuál de las siguientes tecnologías considera que utiliza menos líneas de código en el desarrollo de aplicaciones web enfocado al acceso de datos?**
    - ORM-Hibernate
    - Arquitectura N-Capas

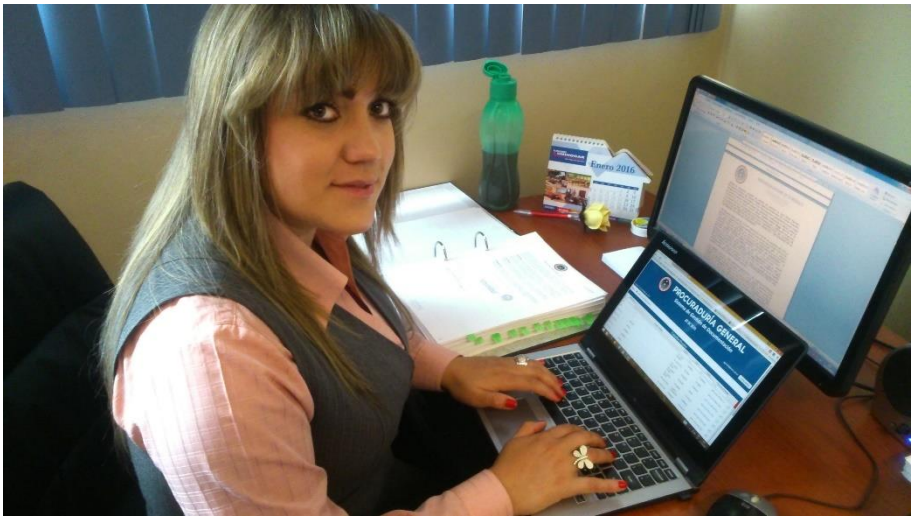
3. **¿Cuál de las siguientes tecnologías considera que ofrece más documentación de ayuda?**
    - ORM-Hibernate
    - Arquitectura N-Capas

ANEXO 9: DEPARTAMENTO PROCURADURÍA GENERAL DE LA UNIVERSIDAD NACIONAL DE CHIMBORAZO


PROCURADURÍA GENERAL






Enero 2016
PROCURADURÍA GENERAL