UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Departamento de Computación

TRABAJO DE DIPLOMA

Título: Editor Web de Diagramas Entidad Relación Extendido

Autor: Yandy Rodríguez Rodríguez

Tutores: MSc. Alain Pereira Toledo

Dr. Abel Rodríguez Morffi

Santa Clara, Junio 2019

UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Computing Department

DIPLOMA THESIS

Title: Web Editor for Extended Entity Relationship Diagrams

Author: Yandy Rodríguez Rodríguez

Thesis Director: MSc. Alain Pereira Toledo

PhD. Abel Rodríguez Morffi

Santa Clara, June 2019

"Lo que convierte la vida en una bendición no es hacer lo que nos gusta, sino que nos guste lo que hacemos"

Goethe

A toda mi familia, con mucho amor y cariño, especialmente a mi madre por quererme y apoyarme tanto.

A mi abuela que ha estado en todo momento cerca de mí apoyándome.

A mi hermana por estar siempre pendiente de mí en todo momento y ser mi inspiración.

De Yandy

A toda mi familia, en especial a mi madre y a mi padre.

A mi hermana Yaily y a mi cuñado Sandy.

A mi abuela.

A mi tía Martha, mis primos jimaguas Lester y Leiser, prima Lumey y Angeline.

A mi madrina Gilda.

A mis amistades que siempre están cuando las necesito especialmente a Amanda y a Carlos.

A mi tutores Alain y Abel por su apoyo incondicional en todo momento.

En fin a todas aquellas personas que ayudaron a formar en mí una persona de bien.

RESUMEN

Los editores de diagramas son herramientas de software que permiten al diseñador añadir, a partir de cero, los objetos y conectores, reubicarlos, modificarlos o borrarlos, además de comprobar la corrección sintáctica de los diagramas. Actualmente existen alternativas tanto de escritorio como Web. Sin embargo muchos son de pago, no explotan las posibilidades de colaboración para el trabajo en equipo, o se reducen a herramientas de dibujo con pobre o ninguna comprobación de las restricciones sintáctica de los diagramas. En consecuencia, en el presente trabajo de diploma se describe el desarrollo de una aplicación Web que lleva por nombre EDiCASE, la cual permite dibujar y manipular diagramas Entidad Relación Extendido, un popular enfoque para el diseño conceptual de bases de datos. Con EDiCASE no solo es posible dibujar y manipular los elementos del diagrama, sino que permite crear proyectos y compartirlos con otros usuarios, crear versiones de los modelos, exportar el modelo a un documento XML, así como personalizar los componentes del modelo con colores. Puede además ser extendida para agregar funcionalidades, lo que la convierte en una herramienta para el trabajo grupal que permitirá avanzar en soluciones cada vez más complejas.

ABSTRACT

Diagram editors are software tools that allow the designer to add, from scratch, the objects and connectors, relocate, modify or delete them, as well as checking the syntactic correction of the diagrams. Currently, there are alternatives like desktop or Web diagram editors. However, many of them are paid, do not exploit the possibilities of collaboration for teamwork, or are merely drawing tools with poor or no checking of the syntactic constraints of the diagrams. Consequently, current work describes the development of a Web application called EDiCASE, which allows drawing and manipulating Extended Relationship Entity diagrams, a popular approach to the conceptual design of databases. With EDiCASE it is not only possible to draw and manipulate the elements of the diagram, but also to create projects and share them with other users, create versions of the models, export the model to an XML document, and customize the components of the model with colors. It can also be extended to add functionalities, which makes it a tool for teamwork that will allow scaling to solutions that are more complex.

INTRODUCCIÓN **1**

1. FUNDAMENTACIÓN TEÓRICA **4**

1.1. INTRODUCCIÓN 4
1.2. MODELO ENTIDAD-RELACIÓN 4
1.2.1. Modelo Entidad-Relación básico *4*
1.2.2. Modelo Entidad Relación Extendido *5*
1.2.3. Notaciones *6*
1.3. HERRAMIENTAS CASE 7
1.3.1. Herramientas CASE Web *8*
1.3.2. Resumen *9*
1.4. CARACTERIZACIÓN DE LAS TECNOLOGÍAS Y LENGUAJES 11
1.4.1. Tecnologías del Front-end *11*
1.4.1.1. HTML5 12
1.4.1.2. CSS3 12
1.4.1.2.1. Bootstrap 13
1.4.1.3. JavaScript 13
1.4.1.3.1. JQuery 14
1.4.1.3.2. AJAX 14
1.4.2. Tecnologías del Back-end *16*
1.4.2.1. Java 16
1.4.2.2. Servicios Web 17
1.4.2.3. Sistema Gestor de Base de Datos 19
1.4.2.4. Servidor Web 20
1.5. LENGUAJE DE MODELADO Y METODOLOGÍA DE DESARROLLO 20
1.5.1. Lenguaje de modelado UML *20*
1.5.2. Metodología de desarrollo AUP *20*
1.5.3. Visual Paradigm *22*
1.5.4. PGAdmin III *22*
1.5.5. IntelliJ IDEA *22*
1.5.6. POSTMAN *23*
1.6. CONCLUSIONES DEL CAPÍTULO 24

2. MODELO DEL NEGOCIO Y REQUISITOS **25**

2.1. INTRODUCCIÓN 25
2.2. ACTORES DEL SISTEMA 25
2.3. ESPECIFICACIÓN DE LOS REQUISITOS FUNCIONALES 25
2.4. ESPECIFICACIÓN DE LOS REQUISITOS NO FUNCIONALES 26
2.5. PAQUETES Y SUS RELACIONES 28
2.6. MODELO DEL DOMINIO O CONCEPTUAL 29
2.7. DIAGRAMA DE CASOS DE USO DEL SISTEMA 30
2.8. DESCRIPCIÓN DE LOS CASOS DE USO DEL SISTEMA. 31
2.9. CONCLUSIONES 44

3. DESCRIPCIÓN DE LA PROPUESTA DE SOLUCIÓN **45**

3.1. INTRODUCCIÓN 45
3.2. ARQUITECTURA DEL SISTEMA 45
3.3. PLUGIN JQUERY: DIAGRAMA 46
3.4. DIAGRAMA DE CLASES DE DISEÑO 47
3.5. DIAGRAMA DE SECUENCIA 52
3.6. DISEÑO DE LA BASE DE DATOS 55
3.6.1. Modelo conceptual de datos 55
3.6.2. Modelo lógico de datos 56
3.6.3. Modelo físico de datos 57
3.7. MODELO DE COMPONENTES 57
3.8. DIAGRAMA DE DESPLIEGUE 57
3.9. CONCLUSIONES 58

4. PRUEBAS Y ANÁLISIS DE FACTIBILIDAD 59

4.1. INTRODUCCIÓN 59
4.2. PLANIFICACIÓN 59
4.2.1. Puntos de casos de uso 59
4.2.1.1. Cálculo de puntos de casos de uso sin ajustar 59
4.2.1.2. Cálculo de puntos de casos de uso ajustados 61
4.2.2. Estimación del esfuerzo 63
4.3. PRUEBAS DE CAJA NEGRA 65
4.4. PRUEBAS UNITARIAS Y DOBLES 67
4.5. PRUEBAS DE RENDIMIENTO. 70
4.5.1. Pruebas de Carga 71
4.5.2. Pruebas de Stress 72
4.5.3. Pruebas de Resistencia 73
4.6. CONCLUSIONES 74

CONCLUSIONES 75

RECOMENDACIONES 76

REFERENCIAS BIBLIOGRÁFICAS 77

ANEXOS 79

Tabla 1 Resumen de las herramientas analizadas 9
Tabla 2 Sitio oficial de la herramienta Web analizadas 11
Tabla 3 Actores del sistema 25
Tabla 4 Especificación de requisitos funcionales del sistema 26
Tabla 5 Caso de usos del sistema Gestionar proyecto 31
Tabla 6 Caso de usos del sistema Gestionar versión 38
Tabla 7 Cálculo de puntos de casos de uso sin ajustar 59
Tabla 8 Factor de peso de los actores sin ajustar 60
Tabla 9 Factor de peso de casos de uso sin ajustar 60
Tabla 10 Cálculo de puntos de casos de uso ajustados 61
Tabla 11 Valores del factor de complejidad técnica 61
Tabla 12 Valores del factor ambiente 62
Tabla 13 Cálculo del esfuerzo 63
Tabla 14 Esfuerzo del proyecto 64
Tabla 15 Condiciones de entrada 66
Tabla 16 Casos de prueba para Gestionar Proyecto: sección Insertar Proyecto 66
Tabla 17 Anexos: Construcciones de la notación Esmari & Navathe 79
Tabla 18 Anexos: Caso de usos del sistema Gestionar usuario 82
Tabla 19 Anexos: Caso de usos del sistema Exportar modelo. 87

Figura 1 Diagrama de paquetes 28
Figura 2 Modelo del dominio o conceptual 30
Figura 3 Diagrama de casos de uso del sistema 31
Figura 4 Prototipo Listar proyecto 34
Figura 5 Prototipo Agregar proyecto 34
Figura 6 Prototipo Eliminar proyecto 35
Figura 7 Prototipo Diálogo Eliminar proyecto 36
Figura 8 Prototipo Editar proyecto 37
Figura 9 Prototipo Diálogo Editar proyecto 38
Figura 10 Prototipo Cargar versión 40
Figura 11 Prototipo Insertar versión 40
Figura 12 Prototipo Salvar versión 41
Figura 13 Prototipo Nueva versión 41
Figura 14 Prototipo Actualizar versión. 42
Figura 15 Prototipo Volver a proyecto. 43
Figura 16 Prototipo Eliminar versión. 44
Figura 17 Arquitectura del sistema (Fuente: elaboración propia) 45
Figura 18 Diagrama de clase de diseño: Insertar proyecto 48
Figura 19 Diagrama de clase de diseño: Editar proyecto 49
Figura 20 Diagrama de clase de diseño: Eliminar proyecto 50
Figura 21 Diagrama de clase de diseño: Insertar versión 51
Figura 22 Diagrama de clase de diseño: Editar versión 51
Figura 23 Diagrama de clase de diseño: Eliminar versión 52
Figura 24 Diagrama de secuencia sección: Insertar proyecto 53
Figura 25 Diagrama de secuencia sección: Editar proyecto 54
Figura 26 Diagrama de secuencia sección: Eliminar proyecto 54
Figura 27 Diagrama de secuencia sección: Insertar versión 54
Figura 28 Diagrama de secuencia sección: Editar versión 55
Figura 29 Diagrama de secuencia sección: Eliminar versión 55
Figura 30 Modelo Conceptual de datos 56
Figura 31 Modelo lógico de datos 56
Figura 32 Diagrama de despliegue 58
Figura 33 Interfaz gráfica para introducir los datos 66
Figura 34 Fragmento del código de la clase diseñada para la realización de las pruebas unitarias y dobles para la operación actualizar modelo 69
Figura 35 Resultado satisfactorio para la operación actualizar modelo 69
Figura 36 Fragmento del código de la clase diseñada para la realización de las pruebas unitarias y dobles para la operación insertar usuario 69
Figura 37 Resultado satisfactorio para la operación insertar usuario 69
Figura 38 Fragmento del código de la clase diseñada para la realización de las pruebas unitarias y dobles para la operación actualizar proyecto 70

Figura 39 Resultado satisfactorio para la operación actualizar proyecto 70
Figura 40 *Script* para la ejecución de las pruebas de rendimiento 71
Figura 41 Resultado de las pruebas de Carga 72
Figura 42 Resultado de las pruebas de Stress 73
Figura 43 Resultado de las pruebas de Resistencia 74
Figura 44 Prototipo Registrar cuenta 83
Figura 45 Prototipo Cambiar contraseña 85
Figura 46 Prototipo Autenticar usuario 86
Figura 47 Prototipo Interfaz Exportar Modelo a XML. 88
Figura 48 Prototipo Exportar Modelo a XML. 88
Figura 49 Diagrama de clase de diseño: Autenticar usuario 89
Figura 50 Diagrama de clase de diseño: Registrar usuario 90
Figura 51 Diagrama de clase de diseño: Cambiar contraseña 90
Figura 52 Diagrama de clase de diseño: Exportar modelo 91
Figura 53 Diagrama de secuencia sección: Autenticar usuario 91
Figura 54 Diagrama de secuencia sección: Registrar usuario 92
Figura 55 Diagrama de secuencia sección: Cambiar contraseña 92
Figura 56 Diagrama de secuencia sección: Exportar modelo 92
Figura 57 Diagrama de componentes general. 93
Figura 58 interfaz del POSTMAN. Escenario de prueba para autenticar usuario. 94
Figura 59 interfaz del POSTMAN. Respuesta del autenticar usuario. 94

Introducción

Actualmente la tendencia del mundo del software a la creación de sistemas, aplicaciones y base de datos con diversas herramientas de desarrollo y arquitecturas hace que las soluciones necesiten ser genéricas y adaptables a cada negocio en particular. El uso de herramientas CASE (acrónimo del inglés *Computer Aided Software Engineering)* por entidades productoras de software permite automatizar aspectos claves en el proceso de desarrollo, tales como el análisis y el diseño, permitiendo aumentar la productividad y la calidad de manera competitiva. Cuba se encuentra en amplio proceso de informatización de la sociedad, por lo que el empleo y desarrollo de herramientas nacionales constituye un punto estratégico para su consolidación.

La creación de esquemas conceptuales constituye una etapa importante en el proceso de diseño de base de datos. Permite describir las propiedades del negocio y representar elementos del espacio del problema, además de ser la base para las siguientes fases del diseño. El modelo Entidad Relación (ER) ha estado presente como modelo conceptual desde la década de los 70 cuando Peter Chen (1976) lo propusiera. Durante todos estos años se le han introducido nuevos conceptos, y el resultado es conocido como modelo Entidad Relación Extendido (ERE). También existen diversas notaciones que representan el modelo.

Entre las herramientas nacionales poseedoras de varias de las construcciones del modelo ERE para la modelación conceptual se encuentra el ERECASE (Álvarez Martínez de la Cotera, 2006; Morffi, 2007). Fue desarrollada en la Universidad Central “Marta Abreu” de las Villas por el grupo de Base de Datos de la Facultad de Matemática, Física y Computación, sobre la plataforma .NET y con el lenguaje de programación C#. La herramienta presenta una interfaz intuitiva y un fuerte proceso de validación de esquemas. Según García (2010), la herramienta desarrollada permite al diseñador construir diagramas Entidad Relación Extendido que pueden ser transformados automáticamente en esquemas relacionales y luego en sentencias SQL (acrónimo del inglés *Structured Query Language*).

En los últimos años un número creciente de herramientas CASE se han desarrollado en el ámbito internacional con el objetivo de facilitar el proceso de diseño conceptual de base de datos, basadas en el modelo ER, las cuales implementan distintas notaciones de este modelo. Estas herramientas actualmente son variadas y han migrado hacia la Web por ser un ambiente colaborativo, permitiendo que los esquemas que se producen puedan ser visualizados y editados por otros usuarios. La mayoría de estas herramientas, presentes en Internet, tienen

deficiencias. Muchas exigen el pago de una licencia para ser usadas y aunque permiten la modelación de varias notaciones del modelo ERE, no incluyen varias de las construcciones propias del modelo como las interrelaciones ternarias y los atributos compuestos o multivaluados. Otras no permiten la colaboración, o al usuario se le asigna la responsabilidad de velar por el cumplimiento de las restricciones propias del modelo, una labor sumamente compleja, principalmente para quienes están comenzando su aprendizaje y no cuentan con la experiencia suficiente en el campo profesional. Se suma a ello, la necesidad creciente de trabajo grupal para avanzar con la resolución de problemas que plantean los sistemas cada vez más complejos.

Este planteamiento conlleva a que se formule el siguiente **problema de investigación**: limitaciones de las herramientas CASE para promover el trabajo colaborativo en el diseño conceptual de bases de datos.

A partir del problema enunciado, emergen las siguientes **preguntas de investigación**:

1. ¿Cuáles son las tecnologías Web apropiadas para la creación de esquemas visuales y el control de versiones?
2. ¿Cuál notación permite construir diagramas Entidad Relación Extendido con alto nivel de expresividad?
3. ¿Cómo supervisar las historias de diseño y el acceso compartido de los esquemas producidos durante el diseño conceptual de una base de datos?

El **Objetivo General** del presente trabajo es: desarrollar un editor de diagramas Entidad-Relación Extendido mediante las tecnologías Web apropiadas, con vistas a promover la colaboración entre los diseñadores de bases de datos que trabajan en un mismo proyecto durante la etapa de modelación conceptual, la capacidad de supervisión de la historia de diseño y el acceso compartido de los esquemas producidos.

Para dar cumplimiento al objetivo mencionado se trazaron las siguientes **objetivos específicos** de la investigación:

1. Identificar la plataforma Web apropiada para la creación de esquemas visuales y el control de versiones.
2. Determinar la notación apropiada para construir diagramas Entidad Relación Extendido, con atención a su nivel de expresividad.
3. Implementar un editor de diagramas que permita el diseño conceptual mediante un diagrama Entidad Relación Extendido.

Con esta investigación se pretende proveer un editor de diagramas Entidad Relación Extendido para la impartición en clases de los contenidos de diseño conceptual de bases de datos, pero desde un ambiente Web, el cual se ha convertido en el entorno ideal para el despliegue de aplicaciones que promueven la participación activa de usuarios. Además, con el uso de una notación que ayude a lograr una mayor expresividad de los esquemas obtenidos, el diseñador puede representar un mayor número de tipos de restricciones identificadas en el negocio.

El contenido del presente Trabajo de Diploma está estructurado de la siguiente manera:

Capítulo I Fundamentación Teórica: analiza las principales características del modelo ER y ERE, principales notaciones en el modelo ERE, CASE actuales Web, así como las tecnologías, lenguajes y herramientas adecuadas que serán empleadas en el desarrollo del sistema.

Capítulo II Modelo de negocio y requisitos: describe detalladamente los principales procesos del negocio y profundiza con el desarrollo del análisis y el diseño, los elementos y funcionalidades que tendrá el sistema, siguiendo la metodología AUP (acrónimo del inglés de *Agile Unified Process*), reconocida ampliamente para el desarrollo ágil de proyectos de calidad.

Capítulo III Descripción de la propuesta de Solución: muestra la Arquitectura del Sistema, Diagramas de Clases de Diseño y de Secuencia de los Casos de Usos del Negocio más significativos. Así como los elementos fundamentales del Modelo de Implementación que son: el Diagrama de Despliegue y el Diagrama de Componentes.

Capítulo IV Pruebas y análisis de factibilidad: muestra la estimación y cálculo de tamaño del software, así como el esfuerzo, los costos que reportará el sistema, las pruebas realizadas al software y la visualización de los resultados de las mismas.

1. Fundamentación Teórica

1.1. Introducción

El presente capítulo tiene por objetivo el análisis de las principales características de las actuales herramientas CASE (acrónimo del inglés *Computer Aided Software Engineering*) para la representación de diversos diagramas, entre ellos el modelo Entidad Relación básico y el extendido. Además, se proporcionará una caracterización de las tecnologías y herramientas que serán utilizados para el desarrollo de la aplicación que permitan representar modelos para un mejor entendimiento de los datos que serán manejados en el desarrollo de aplicaciones.

1.2. Modelo Entidad-Relación

El modelo de datos Entidad Relación (ER) está basado en una percepción del mundo real. Se desarrolló para facilitar el diseño conceptual de bases de datos y provee una abstracción de la realidad en la que se representan los conceptos como entidades, atributos y las interrelaciones entre ellos. Actualmente no se ha estandarizado y se le considera una herramienta para el modelado de datos. El modelo ER fue presentado por Chen (1976).

1.2.1. Modelo Entidad-Relación básico

El modelo Entidad Relación es simple y sus conceptos básicos para la construcción de diagramas son (Elmasri and Navathe, 2007):

Entidades: se refieren a los conceptos, personas, objetos o cosas de los cuales el sistema necesite guardar información. Una entidad representa un tipo de objeto distinguible. Constituye un objeto de negocio importante que contiene más de una propiedad.

Atributos: constituyen cada una de las características asociadas a una entidad o relación. Son propiedades que se utilizan para describir una entidad o una relación.

Relaciones: muestran la forma en que dos o más entidades se asocian. Un tipo de relación representa una asociación entre entidades. Un tipo de relación puede ser unario, binario o n-ario, dependiendo de si el número de entidades involucradas en la relación es 1, 2 o más de 2.

1.2.2. Modelo Entidad Relación Extendido

Desde mediados de los 80 el incremento en el desarrollo de sistemas de bases de datos muchos se dieron a la tarea de introducir conceptos adicionales que ayudasen a captar la semántica del dominio del discurso; esta acometida por mejorar el modelo básico es conocida como Modelo Entidad Relación Extendido (ERE). Las extensiones en este modelo hacen referencias fundamentalmente a las interrelaciones y reglas de transformación hacia el modelo lógico. Suple las deficiencias desde el punto de vista semántico del modelo básico y lo enriquece con nuevos mecanismos de abstracción y representación de la realidad. En el modelo ERE, destacan las siguientes extensiones:

Atributos Multivalor: La mayoría de los atributos tienen solo un valor para una entidad particular, pero cuando una entidad puede tener varios valores para un atributo entonces dicho atributo se denomina multivalor (Elmasri and Navathe, 2007).

Atributos Derivados: En algunos casos, dos o más valores de atributo pueden estar relacionados, entonces a dicho atributo se le denomina derivado (Elmasri and Navathe, 2007).

Restricciones de Participación y Cardinalidad: La restricción de cardinalidad representa el número máximo de instancias de entidad que pueden o deben ocurrir para participar en la relación. La restricción de participación representa el número mínimo de instancias de entidad que deben ocurrir para participar en la relación. Por lo tanto, la restricción de participación representa la existencia total (obligatoria) o parcial (opcional) de una instancia de entidad en lo que se relaciona con su relación con otra entidad (Evans et al., 1995).

Generalización: en el modelo ERE es posible establecer jerarquías de generalización entre los conjuntos de entidades (Batini et al., 1992).

- Cubrimiento Total o Parcial. El cubrimiento de una generalización es total si cada elemento del conjunto de entidades genérico es transformado en al menos un elemento de los conjuntos de entidades de nivel más bajo o específicos; es parcial si existe algún elemento del conjunto de entidades genérico que no es transformado a algún elemento de los conjuntos de entidades específicos.
- Cubrimiento Disjunto o Solapado. El cubrimiento de una generalización es disjunto si cada elemento del conjunto de entidades genérico es transformado a lo sumo a

un elemento de los conjuntos de entidades específicos; es solapado si existe algún elemento del conjunto de entidades genérico que es transformado a elementos de dos o más conjuntos de entidades diferentes.

Agregaciones: una limitación del modelo ER básico es que no facilita expresar interrelaciones entre interrelaciones. Una interrelación con cardinalidad máxima y los conjuntos de entidades que relaciona, pueden ser manejados como un conjunto de entidades en un nivel de abstracción mayor, lo que posibilita que se pueda asociar con otros conjuntos de entidades (Batini et al., 1992).

1.2.3. Notaciones

Según Quintero et al. (2008) "*Para construir modelos de datos existen diversas notaciones, las cuales le dan diferente relevancia a cada uno de los elementos que se pueden representar en los modelos*". La existencia de variadas notaciones con construcciones muy diferenciadas implica cambios en la representación del modelo ERE, es decir en los conceptos y extensiones que se modelan. Razón por la cual se analizaron algunas de las características más significativas en las notaciones utilizadas por herramientas CASE, para encontrar la de mayor nivel de expresividad.

Las construcciones de estas notaciones representan la generalización y la agregación por lo que no serán analizados estos conceptos y extensiones, concentrándose en las particularidades más significativas encontradas:

- Notación IE (*Information Engineering*): desarrollada inicialmente por Clive Finkelstein quien luego la refinó con el apoyo de James Martin; aunque es clara e intuitiva, sirve solo para modelos de alto nivel de abstracción (modelos lógicos). Su modelo es binario no permitiendo relaciones n-arias, no modela atributos y tampoco atributos relacionados con una relación. La relación se muestra como una línea que conecta dos entidades asociadas y recibe un nombre. El tipo de entidad se representa como un cuadro. Las relaciones están representada por una línea y su cardinalidad se indica con el pie de gallo al final de esta (Evans et al., 1995).
- Notación Barker: fue adoptada por Oracle Corporation en sus productos de modelado de datos; sirve de referente en la metodología planteada por Richard Barker. Entre los elementos más significativos se encuentran que su modelo es binario y no permite relaciones n-aria y atributos en las relaciones. Las relaciones están representada por una línea y su cardinalidad se indica con el pie de gallo al

final de la línea. En esta notación solo se representan atributos opcionales, requeridos y claves, el primero permite tomar un valor nulo, se ilustran con una pequeña 'o' delante del nombre del atributo. Los requeridos, cuyo valor siempre se requiere, se indican con un pequeño '*' y el atributo clave se representa con '#' (Evans et al., 1995).

- Notación IDEF1x: su modelo al ser binario no permite relaciones n-arias. Su representación constituye un cuadro cerrado con el nombre de la entidad en la parte superior. Los atributos de la entidad se enumeran dentro del cuadro. Las claves principales se enumeran en la sección superior del cuadro. Los atributos de datos (clave no primaria) se indican en la sección inferior del cuadro. Existen tipos de entidades, como las independientes las cuales constituyen el tipo de entidad regular, se representan con una caja cuadrada y las dependientes está representada por una caja de esquinas redondeadas. Las relaciones están representadas por una línea y su cardinalidad con un círculo oscuro al final de esta (Evans et al., 1995).
- Notación Elmasri&Navathe (2007): es una notación que posee gran expresividad por representar los diferentes tipos de datos, vínculos y restricciones. Su análisis conllevó a la decisión de utilizar esta notación para modelar los diagramas en el editor. Según Evans et al (1995), esta notación constituye la más rica semánticamente en términos de componentes de modelado y restricciones. En la Tabla 17 en los Anexos se representan las principales construcciones de esta notación.

1.3. Herramientas CASE

Las herramientas CASE son aplicaciones informáticas destinadas a aumentar la productividad y la calidad en el desarrollo del software reduciendo los costos del mismo en términos de tiempo y dinero. Estas herramientas permiten tener una mejor organización y control del desarrollo de un sistema informático (Chavarría Báez and Ocotilla Rojas, 2016).

A fines de la década de 1980 se introdujeron en el mercado las primeras herramientas CASE y es a mediados de los 90, con el aumento de la orientación a objetos, que se consideró que las herramientas CASE eran un fracaso. Las primeras herramientas OO CASE comenzaron la llamada “Guerra de la Notación”, lo que hizo que la comparación de

las herramientas fuera bastante difícil porque las personas comparaban más las notaciones de diagramación que las características de la herramienta en sí mismas. Dicha controversia culminó cuando Rumbaugh, Booch y Jacobson (2007) se reunieron y crearon UML (acrónimo del inglés de *Unified Modeling Language*)(DIAS, 2017).

1.3.1. Herramientas CASE Web

Lucidchart

Es una herramienta de diagramación basada en la Web, que permite a los usuarios colaborar y trabajar juntos en tiempo real, creando diagramas de flujo, organigramas, esquemas de sitios Web, diseños UML, mapas mentales, prototipos de software y muchos otros tipos de diagrama. Esta herramienta está construida con estándares Web, como HTML5 y JavaScript, además funciona en todos los navegadores Web modernos. También cuenta con una variedad de opciones de personalización. La importación de SVG, permite al usuario crear sus propias librerías de formas. También permite alterar libremente las formas, líneas y plantillas existentes, así como también subir imágenes propias a los diagramas.

Permite variadas notaciones para modelar un diagrama ERE, como la notación Chen, Information Engineering, Barker, IDEF1X. Posee variada documentación sobre el modelado.

DRAW.IO

Es una aplicación Web fácil de utilizar y sin muchas restricciones para acceder a ella. Incluye formas básicas para los elementos UML (acrónimo del inglés de *Unified Modeling Language*), Entidad Relación y BPMN (acrónimo del inglés de *Business Process Model and Notation*).

Sus características principales son:

- Cliente HTML 5 nativo.
- Gran biblioteca de plantillas incorporada.
- Interfaz intuitiva de arrastrar y soltar.
- Función de buscar y agregar imágenes.
- Exportar a PNG/JPG/XML/SVG/PDF.
- Soporte de dispositivos táctiles.

- Colaboración en tiempo real.

CREATELY

Es una herramienta de diagramación, reconocida por su facilidad de uso. Posee más de 50 tipos de diagramas y miles de ejemplos para modelar ágilmente. También permite trabajar en modo *offline* y sincronizar los modelos en modo *online*. Permite modelar diagramas ER con notación Chen.

El programa funciona correctamente en cualquier navegador y requiere de Flash para poder funcionar. Está muy completo y tiene una buena cantidad de librerías, además de contar con suficientes herramientas para lograr resultados completamente profesionales.

ERDplus

ERDplus es una sencilla plataforma Web que centra sus funciones en el modelado de bases de datos. Ofrece el modelado de diagramas ER y esquemas relacionales. La notación utilizada permite dibujar entidades regulares y débiles, varios tipos de atributos, y todas las posibles restricciones de cardinalidad de las relaciones .Además permite la conversión automática de diagramas ER a esquemas relacionales y estos a la vez en sentencias SQL.

SmartDraw

Sirve para hacer múltiples diagramas. Una de las ventajas con las que cuenta esta aplicación es que es posible compartir documentos, para que varias personas puedan trabajar sobre ellos. Posee una aplicación nativa para Windows. Permite crear diagramas ER de forma fácil y rápida, además soporta las notaciones para el modelo ER: *Information Engineering* y Chen.

1.3.2. Resumen

En la Tabla 1 se detalla un resumen de las herramientas analizadas y en la Tabla 2 los sitios oficiales de las herramientas CASE basadas en la Web.

TABLA 1 RESUMEN DE LAS HERRAMIENTAS ANALIZADAS

Nombre	Tipo	Gratuita	Colaboración	Versionado	Observaciones
LucidChart	Web	No	Sí	Sí	Herramienta muy completa, sin

					embargo, es de pago. Está basada en la nube que, aunque es una tecnología muy novedosa requiere de conexión constante a internet.
DRAW.IO	Web	Sí	Sí	No	Es gratuita y de fácil uso, no obstante, es una herramienta que requiere de experiencia, ya que no tiene en cuenta las reglas para el modelado y permite dibujar los diagramas sin ninguna restricción.
CREATELY	Web	No	Sí	Sí	Herramienta de fácil uso y de pago, necesita tener instalado Flash lo cual es un inconveniente ya que hace su funcionamiento un poco más lento, además la compañía Adobe encargada de su mantenimiento no dará más soporte al *plugin* a partir del 2020.
ERDplus	Web	Sí	No	No	Es una herramienta muy básica. No permite la colaboración.
SmartDraw	Web	No	Sí	No	Esta herramienta no permite el versionado, pero si la colaboración, constituye un problema para los usuarios en el trabajo grupal dependen de comunicación constante para informar de cambios realizados

					en los diagramas compartidos.

TABLA 2 SITIO OFICIAL DE LA HERRAMIENTA WEB ANALIZADAS

Herramienta Web	**Sitio oficial**
Lucichard	https://www.lucidchart.com/
DRAW.IO	https://www.draw.io/
CREATELY	https://creately.com/
ERDplus	https://erdplus.com/
SmartDraw	https://www.smartdraw.com/

Después del análisis realizado a diferentes herramientas que existen en la actualidad, se puede determinar que estas, aunque permiten el modelado de diferentes diagramas útiles para el desarrollo de una aplicación, de manera general, algunas de ellas no brindan la posibilidad de lograr la colaboración entre usuario o son de pago.

1.4. Caracterización de las tecnologías y lenguajes

1.4.1. Tecnologías del Front-end

En la actualidad, con el avance de Internet, ha surgido diversidad de tecnologías y lenguajes para la creación de aplicaciones Web. Estas ofrecen grandes facilidades para establecer comunicación con el usuario mediante páginas y su visualización desde cualquier parte del mundo haciendo uso de un navegador.

Las tecnologías del *Front-end* se utilizan del lado del cliente, y gestionan la información que le es trasmitida al servidor. Son las encargadas de mostrar de manera cómoda e interactiva los datos al usuario. Actualmente existe una amplia variedad de estas tecnologías. A continuación se describen las tecnologías analizadas para la implementación del editor, por sus características y ventajas.

1.4.1.1. HTML5

En el desarrollo de la aplicación se utilizará la quinta revisión del lenguaje básico de desarrollo Web, HTML (acrónimo del inglés *Hypertext Markup Language*). HTML5 es interpretado y permite describir hipertexto, es decir, texto presentado de forma estructurada y agradable, con enlaces que conducen a otros documentos o fuentes de información relacionadas. Se le considera la estructura básica de toda página (Goldstein et al., 2015).

Ventajas de HTML5:

- Es simple, provocando que disminuyan los tiempos de carga de una página.
- No necesita compilación, solo es necesario un navegador para que sea interpretado.
- Es semántico, ya que al dotarse de un significado cada parte de la estructura de una página es mejor interpretado por el navegador, permitiendo que la indización de los documentos sea más certera.
- Posee estructura de etiquetas mejorada, permitiendo definir por separado el encabezado, la barra de navegación, las secciones de la página, los textos del sitio, los diálogos y el pié de página del sitio. Mejorando así la creación de la estructura del código Web y en el manejo óptimo de las etiquetas.
- Ofrece la posibilidad de obtener un código más limpio y fácil de depurar.
- Permite que las páginas Web almacenen localmente códigos y contenidos, a través de la caché.
- Los sitios desarrollados son más dinámicos.

1.4.1.2. CSS3

El CSS (acrónimo del inglés *Cascading Style Sheets*) en su tercera revisión permite controlar la presentación de documentos estructurados y escritos en HTML. El uso de CSS3 en el desarrollo del editor permitirá separar la estructura y el contenido de la presentación estética en un documento, con un control mayor del mismo y sus atributos (Goldstein et al., 2015).

Ventajas de CSS3:

- Permite vincular un solo archivo CSS a diversas páginas definiendo los estilos de todo un sitio.
- Mejora el tiempo de respuesta de un sitio, si los estilos se encuentran en un solo archivo CSS.
- Evita la repetición de código en los archivos HTML.
- Es compatible con múltiples navegadores.
- Ofrece la posibilidad de cambiar la apariencia y el comportamiento de un elemento añadiéndole animaciones y transiciones; por lo que provee de mayor interactividad al sitio.

1.4.1.2.1. Bootstrap

Entre los *framework* CSS existentes se eligió Bootstrap por ser *OpenSource***.** Además contener un conjunto de plantillas basadas en CSS y HTML para diseñar estilos, elementos, botones, navegación, tipografía y una gran variedad de componentes para la interfaz de usuario (Lambert, 2016). Por ello va a tener un papel importante en el desarrollo del editor, ya que la interfaz que será desarrollada proveerá usabilidad e interactividad a los usuarios.

Ventajas de Bootstrap:

- Permite el diseño adaptable, mejor conocido como RWD (acrónimo del inglés *Responsive Web Design*) permitiendo que se ajuste a cualquier dispositivo y tamaño de pantalla.
- Incluye una gran cantidad de complementos. Permite un ahorro sustancial de tiempo al no depender de la integración y el estudio de *plugins* para mejorar la interfaz visual.
- Constituye un marco simple de trabajo de uso ágil y sencillo, y facilita el diseño de interfaces a los programadores.

1.4.1.3. JavaScript

JavaScript es un lenguaje basado en objetos y guiado por eventos, diseñado específicamente para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de Internet (Bae, 2019).

Ventajas de JavaScript:

- Permite que se pueda probar directamente en cualquier navegador sin necesidad de procesos intermedios, convirtiéndolo en un lenguaje interpretado.
- Permite que *scripts* desarrollados no requieran de mucha memoria ni tiempo adicional de transmisión, por ser pequeños y compactos.
- No requiere de un tiempo de compilación; ya que los *scripts* se pueden desarrollar en un período de tiempo relativamente corto.
- Es independiente de la plataforma hardware o sistema operativo, funciona correctamente siempre y cuando exista un navegador con soporte JavaScript.

1.4.1.3.1. JQuery

El *framework* JQuery es una biblioteca *OpenSource* de JavaScript, rápida y compacta. Posee variadas funciones que permiten la manipulación de documentos HTML, el manejo de eventos y de animaciones. Su API (acrónimo del inglés *Application Programming Interface*) simple y completa tiene la capacidad de cambiar completamente la forma en que se escribe el JavaScript (Franklin and Ferguson, 2017).

Ventajas de JQuery (York, 2015) :

- Permite iterar y atravesar el DOM (acrónimo del inglés *Document Object Model*) a través de su sofisticada e incorporada capacidad para usar selectores, tal como se haría en CSS.
- Facilita la adición de métodos personalizados a través de su arquitectura de *plugins* fácil de entender, permitiendo un ahorro substancial de tiempo y esfuerzo.
- Reduce la redundancia en la navegación y la funcionalidad de la interfaz de usuario, como cuadros de diálogo emergentes, animaciones y transiciones.
- Es flexible y rápido para el desarrollo de aplicaciones.

1.4.1.3.2. AJAX

Con la utilización de AJAX (acrónimo del inglés *Asynchronous JavaScript And XML*) en el desarrollo en la Web se facilita la interactividad de la aplicación que se ejecuta en el navegador de los usuarios con una comunicación asíncrona con el servidor. De esta forma, es posible realizar cambios sobre una página sin necesidad de recargarla, aumentando la velocidad y usabilidad de la misma (Sandeep, 2014).

Ventajas de AJAX:

- Menor tiempo de transferencia de datos cliente/servidor.
- Optimización de recursos.
- Las aplicaciones desarrolladas son más rápidas e interactivas, al estilo de aplicaciones de escritorio, por lo que mejora la experiencia del usuario.
- Reduce de manera significativa tener que cargar información continuamente del servidor, actualizando solamente porciones de la página.
- Reduce de manera significativa los tiempos de carga inicial y el tiempo de espera para una petición.
- Así como AJAX funciona en cualquier navegador, es perfectamente compatible con cualquier tipo de servidor estándar y lenguaje de programación Web.

Entre las diferentes notaciones que existen para la comunicación servidor/cliente se seleccionó JSON (acrónimo del inglés *JavaScript Object Notation*), por ser un formato ligero para el intercambio de datos abierto y basado en texto. También por estar desarrollado principalmente para usarse en aplicaciones Web con el objetivo de trasmitir información estructurada de forma asíncrona, integrándose perfectamente con AJAX. Lo anterior se confirma con las siguientes palabras de Marrs (Marrs, 2017):

> *JSON es simple y está reemplazando gradualmente a XML como el formato principal de intercambio de datos en Internet. JSON es fácil de leer y sus estructuras se traducen fácilmente a conceptos bien entendidos por los desarrolladores de software: matrices, objetos y pares de nombre / valor.*

Además JSON constituye un subconjunto de la notación literal de objetos de JavaScript que se usa generalmente con ese lenguaje. Los tipos básicos y las estructuras de datos de la mayoría de los lenguajes de programación pueden ser representados en JSON, por lo que es posible usar el formato para intercambiar datos estructurados entre programas escritos en diferentes lenguajes como: Java y JavaScript.

Entre las principales características de JSON se encuentran:

- Su sintaxis es muy concisa por lo que requiere menos codificación y procesamiento que su homólogo XML.
- Soporta diversos lenguajes de programación.
- Es legible e independiente de la plataforma.
- No se necesita código de aplicación adicional para analizar texto.

1.4.2. Tecnologías del Back-end

Las tecnologías del *Back-end* se utilizan del lado del servidor, para gestionar las diferentes peticiones de información que le llegan desde el cliente. Una vez tratada la información le es devuelta al cliente para que sea visualizada a través de las tecnologías descritas anteriormente. Actualmente existe diversidad de estas tecnologías para la gestión de servicios, a continuación se enuncian las escogidas por su versatilidad en el desarrollo de aplicaciones Web y sus principales características.

1.4.2.1. Java

Entre los lenguajes de programación orientados a objetos (POO) u OOP (acrónimo del inglés *Object-Oriented Programming*), se encuentra Java. Lenguaje flexible y multiplataforma desarrollado por James Gosling de Sun Microsystems a principio de los 90. La intención de Sun Microsystems era crear un lenguaje con una estructura y una sintaxis similar a C y C++, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel. La mayor parte de las tecnologías Java están bajo la licencia GNU GPL. Las aplicaciones desarrolladas en Java funcionan en cualquier entorno, dado que no es el sistema quien las ejecuta, sino la JVM (acrónimo del inglés *Java Virtual Machine*)(Burd, 2017). Los servicios de la aplicación serán desarrollados en Java, permitiendo la implementación de clases para el procesamiento de las solicitudes enviadas desde el cliente y generar las respuestas.

Ventajas del desarrollo con Java (Saternos, 2014):

- Variedad de bibliotecas que pueden utilizarse para abordar la mayoría de las actividades de traducción o integración con otras tecnologías, como Jackson y Gson, dos bibliotecas populares que pueden convertir objetos Java en su representación JSON y viceversa.
- Java y la JVM permiten incluir la funcionalidad del servidor en un proyecto sin la necesidad de instalar, configurar y mantener una implementación completa del mismo.
- Lenguaje multiplataforma.

1.4.2.2. Servicios Web

Los servicios Web pueden definirse como el conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones con el objetivo de brindar servicios. Las aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios Web para intercambiar datos en redes de ordenadores, lo que contribuye a sistemas más flexibles y estables. Se escogió el estilo arquitectónico de desarrollo de software REST (acrónimo del inglés *Representational State Transfer*), por establecerse en los últimos años como modelo predominante en cuanto a la adopción de una API (acrónimo del inglés *Application Programming Interface*) a gran escala, desplazando a otras arquitecturas como SOAP (acrónimo del inglés *Simple Object Access Protocol*) orientadas mayormente a operaciones transaccionales (Patni, 2017).

Fielding (2000) define el término REST como estilo arquitectónico de desarrollo de software basado en recursos, que depende de protocolos de comunicaciones con caché cliente/servidor, sin estado, como el protocolo estándar HTTP (acrónimo del inglés *Hypertext Transfer Protocol*), para facilitar el desarrollo de las aplicaciones. Al aplicar los principios del diseño de REST a un protocolo como HTTP, los desarrolladores pueden crear interfaces para su uso en prácticamente cualquier dispositivo o sistema operativo. REST se describe mediante un conjunto de reglas arquitectónicas que tratan de minimizar la latencia y las comunicaciones de la red y, al mismo tiempo, maximizar la independencia y escalabilidad de las implementaciones de los componentes (bases de datos, código fuente).

Ventajas del desarrollo basado en REST:

- Separación cliente/servidor: permite que los componentes desarrollados puedan evolucionar de forma independiente, posibilitando que se puede escalar flexiblemente el producto sin excesivos problemas.
- Independencia de plataformas/lenguajes: ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo, en cualquier tecnología o lenguaje.
- Mejora la experiencia de usuario: teóricamente el desarrollo de sitios Web basados en un API (acrónimo del inglés *Application Programming Interface*) con estilo

arquitectónico REST; puede dar mejor desempeño y ser más fiable que uno tradicional.

Por todo lo descrito anteriormente, se requiere que la aplicación a desarrollar se implemente como un servicio Web con estilo arquitectónico REST. Además con la utilización del protocolo HTTPS (acrónimo del inglés *Hypertext Transport Protocol Secure)* versión segura de HTTP, garantiza que el usuario podrá intercambiar información codificada de manera segura con las páginas Web, sin que la información sea interceptada ni llegue a manos de terceros.

Ventajas del uso de HTTPS:

- Permite fomentar un estándar y un protocolo basado en texto.
- Permite la seguridad de los datos al ser transmitidos al servidor.
- Permite la integridad de los datos.
- Permite la verificación de identidad: garantiza que la información que un usuario envíe llegue al destino correcto y queda registrado tal y como ha escrito el usuario.
- Genera confianza a los usuarios con su uso.

Según Patni (2017) el API o especificación de Java para servicios Web, JAX-RS proporciona soporte para crear servicios Web de acuerdo con el patrón arquitectónico REST. JAX-RS es una colección de interfaces y anotaciones de Java que simplifica el desarrollo de aplicaciones del lado del servidor. Se optó por utilizar esta tecnología por permitir que los servicios sean más fáciles de desarrollar y de consumir, además de ser la especificación de referencia de REST (Patni, 2017).

Características sobresalientes de JAX-RS:

- Clases de recursos basadas en POJO (acrónimo del inglés *Plain Old Java Object*). Es decir un objeto java se convierte en un recurso Web añadiéndole anotaciones.
- Modelo de programación centrado en HTTP.
- Es una API estandarizada para construir y consumir servicios Web REST.

Entre las anotaciones esenciales que provee JAX-RS se encuentran:

- @Path indica la ruta relativa a añadir a la URI (acrónimo del inglés *Uniform Resource Identifier*) para acceder a una clase o método.
- @GET, @PUT, @POST, @DELETE, @HEAD hacen referencia al tipo de petición HTTP que satisface un método.

- @Produces especifica el tipo MIME (acrónimo del inglés *Multipurpose Internet Mail Extensions*) que retorna un método.
- @Consumes especifica el tipo MIME que requiere un método.

Además se eligió Jersey como biblioteca de desarrollo de la aplicación, por ser la implementación más estable y de referencia de JAX-RS según Sandoval (Sandoval, 2009), y proporcionar un conjunto de herramientas para Java necesarias para desarrollar el proyecto. Otros motivos que conllevaron a su elección fueron:

- La licencia de Jersey es *OpenSource*, por lo que permite su utilización sin necesidad de pagarla.
- Cuenta con una amplia documentación.
- Jersey es sencillo de utilizar una vez configurado, permite realizar de manera simple la comunicación cliente /servidor.
- La librería se encarga totalmente de la realización de la comunicación y las peticiones empleadas.

1.4.2.3. Sistema Gestor de Base de Datos

PostgreSQL es un gestor de bases de datos relacional orientado a objetos, libre y gratuito. Posee una gran escalabilidad, ya que es capaz de soportar una gran cantidad de peticiones simultáneas, ajustándose al número de CPUs y a la cantidad de memoria que posee el sistema de forma óptima (Smith, 2010).

PostgreSQL se basa en el modelo de transacciones ACID (acrónimo del inglés *Atomicity, Consistency, Isolation and Durability*). Además permite el control de la concurrencia mediante un sistema denominado MVCC (acrónimo del inglés *Multiversion Concurrency Control*) el cual asegura que una operación no pueda afectar a otras, de esta manera dos transacciones sobre la misma información no genera error.

PostgreSQL presenta las siguientes características:

- Un punto débil de muchos gestores de bases de datos relacionales es que no proveen de un tipo de datos integrado para manejar los documentos JSON. Sin embargo, PostgreSQL permite el trabajo con este formato.
- PostgreSQL es multiplataforma.
- Tiene la capacidad de comprobar la integridad referencial, equiparándolo con los gestores de bases de datos de alto nivel.

1.4.2.4. Servidor Web

Apache Tomcat Server es un servidor Web potente y flexible; disponible para distintas plataformas y entornos. Al estar escrito en Java solo necesita de su Máquina Virtual para su ejecución. Puede ser usado como servidor Web independiente en entornos con alto nivel de tráfico de datos y alta disponibilidad (Vukotic and Goodwill, 2011).

Características de Apache Tomcat:

- Se puede acceder a su código fuente en los términos establecidos en la *OpenSource Apache License*.
- Es prácticamente universal, por su disponibilidad en multitud de sistemas operativos.
- Es altamente configurable en cuanto a la creación y gestión de *logs*, de este modo es posible tener un mayor control sobre lo que sucede en el servidor.

1.5. Lenguaje de modelado y metodología de desarrollo

1.5.1. Lenguaje de modelado UML

UML (acrónimo del inglés de *Unified Modeling Language*) es un lenguaje de modelado visual de propósito general que se utiliza para especificar, visualizar, construir y documentar los artefactos de un sistema software. Los modelos de UML están pensados, tanto para el análisis lógico, como para el diseño destinado a la implementación. No exige un proceso de desarrollo concreto, pero fue diseñado para servir de apoyo a los procesos interactivos, incrementales, guiados por casos de uso y centrados en la arquitectura (Rumbaugh et al., 2007).

Ventajas de UML:

- UML estandariza los artefactos y la notación que se utilicen.
- Se basa en una notación gráfica concisa, fácil de aprender y utilizar.
- Se puede utilizar para modelar sistemas software en diversos dominios.

1.5.2. Metodología de desarrollo AUP

La metodología de desarrollo de software escogida fue AUP (acrónimo del *Agile Unified Process*) desarrollada por Scott Ambler (2002) es una versión simplificada de RUP (acrónimo del inglés de *Rational Unified Process*), y describe una aproximación al

desarrollo de aplicaciones que combina conceptos propios del proceso unificado tradicional con técnicas ágiles, con el objetivo de mejorar la productividad. AUP se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y ser iterativo e incremental.

Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Consta de cuatro fases, al igual que en RUP: Iniciación, Elaboración, Construcción y Transición. Sin embargo, a diferencia de RUP, AUP se guía por siete disciplinas (Stephens, 2015):

- Modelo: su objetivo es entender el negocio de la organización, el problema de dominio que se abordan en el proyecto, y determinar una solución viable para resolver el problema de dominio.
- Aplicación: su objetivo es transformar el modelo o modelos en código ejecutable y realizar un nivel básico de las pruebas, en particular, pruebas unitarias.
- Prueba: su objetivo consiste en realizar una evaluación objetiva para garantizar la calidad. Esto incluye la búsqueda de defectos, validar que el sistema funciona tal como está establecido, verificando que se cumplan los requisitos.
- Despliegue: el objetivo es la prestación y ejecución del sistema y que el mismo este a disposición de los usuarios finales.
- Gestión de configuración: el objetivo es la gestión de acceso a herramientas de un proyecto. Esto incluye no sólo el seguimiento de las versiones con el tiempo, sino también el control y gestión del cambio para ellos.
- Gestión de proyectos: el objetivo es dirigir las actividades que se lleva a cabo en el proyecto. Esto incluye la gestión de riesgos, la dirección de personas (la asignación de tareas, el seguimiento de los progresos), coordinación con el personal y los sistemas fuera del alcance del proyecto para asegurarse de que es entregado a tiempo y dentro del presupuesto.
- Entorno: el objetivo es apoyar el resto de los esfuerzos por garantizar que el proceso sea el adecuado, la orientación (normas y directrices), y herramientas (hardware, software) estén disponibles para el equipo según sea necesario.

Además AUP insiste en la necesidad de contar con un equipo pequeño y sólido, compuesto por miembros competentes, capaces de desempeñar, incluso varios roles (André and Baldoquín, 2008).

1.5.3. Visual Paradigm

Se optó por la utilización de la herramienta Visual Paradigm para UML por constituir una poderosa herramienta CASE que soporta el ciclo de vida completo del desarrollo de software. Está diseñado para un amplio rango de usuarios, incluyendo ingenieros de software, analista de sistema, analista de negocio, arquitectos de sistema y quienes estén interesados en la construcción de sistemas de software confiables mediante el uso de la Orientación a Objetos.

Entre las características principales se pueden citar:

- Facilita una rápida construcción de aplicaciones de calidad a un menor costo.
- Se encuentra disponible en múltiples versiones y plataformas.
- Posee una fuerte validación de la sintaxis y permite personalizar propiedades.
- Permite la reutilización de diagramas y elementos de UML.

1.5.4. PGAdmin III

Se utilizará en el desarrollo de la aplicación la herramienta administrativa PGAdmin III, aplicación gráfica usada para la gestión de PostgreSQL, siendo la más completa y popular con licencia *OpenSource*. PGAdmin está escrito en C++ y utiliza la biblioteca gráfica multiplataforma wxWidgets, permitiendo que se pueda usar en sistemas operativos como GNU/Linux, FreeBSD, Solaris, Mac OS y Windows. PGAdmin está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples, hasta desarrollar bases de datos complejas.

La aplicación incluye también:

- Editor SQL con resaltado de sintaxis.
- Amplia documentación para ayudar a los usuarios menos experimentados.

1.5.5. IntelliJ IDEA

IntelliJ IDEA es desarrollado por JetBrains, y está disponible en dos ediciones: *Community*, disponible bajo licencia *OpenSource* y la edición comercial llamada *Ultimate*. Es el IDE que se utilizara para desarrollar la aplicación, por ser multiplataforma, ampliamente usado en el desarrollo de aplicaciones Web cliente/servidor con estilos arquitectónicos REST (Krochmalski, 2014).

Entre las características principales se encuentran:

- Asistencia de codificación inteligente: posee un editor inteligente que reconoce Java, HTML, XML, CSS y JavaScript.
- Soporte de desarrollo Web: permite el rápido desarrollo de aplicaciones Web con soporte completo para las tecnologías y lenguajes como XML, HTML, CSS y servidores como Glassfish y Apache Tomcat. Además, posee soporte mejorado de JavaScript y AJAX para la construcción de interfaces Web estándar.
- Soporte de Pruebas Unitarias: incorpora un corredor de pruebas totalmente compatible con JUnit. Además, IntelliJ IDEA proporciona una API de inspección flexible la cual permite crear fácilmente complementos para ampliar la funcionalidad de análisis de código.

1.5.6. POSTMAN

Resulta imprescindible para el trabajo con la arquitectura REST, apoyarse en herramientas que faciliten el trabajo con peticiones Web. Una de las herramientas habituales es POSTMAN, la cual ha alcanzado popularidad entre desarrolladores Web. Funciona como una extensión del navegador Google Chrome, pero también se encuentra disponible como aplicación nativa para los sistemas operativos Mac, Windows y Linux. La gran mayoría de sus componentes básicos son *OpenSource,* aunque existe una versión *Premium* que añade funcionalidades adicionales. En este trabajo se utiliza la versión nativa para Windows de POSTMAN, pues permite crear distintos escenarios de pruebas y organizarlas. Además también ofrece la posibilidad de (Holmberg and Nyberg, 2018):

- Crear y enviar peticiones HTTP y HTTPS.
- Modificar todos los parámetros de una petición, así al editar el contenido, se puede añadir campos tanto en JSON, XML o texto plano.
- Almacenar las peticiones para poder reproducirlas posteriormente.
- Probar todas las funcionalidades de una aplicación Web sin tener que escribir líneas de código. Esto supone un gran ahorro de tiempo, en todas las etapas del desarrollo.

1.6. Conclusiones del capítulo

En este capítulo se ofreció una visión general de las características principales de modelo Entidad Relación Extendido y algunas de las notaciones que se utilizan para el modelado conceptual. Se tomó la notación de Elmasri&Navathe para la implementación del editor por sus características. También se describieron algunas características de las herramientas CASE actuales basadas en un entorno Web, las cuales propician la colaboración de usuarios y creación de esquemas conceptuales con algunas de las notaciones analizadas.

Teniendo en cuenta las ventajas y características de las tecnologías y lenguajes analizados, se decidió desarrollar el *Back-end* con el uso de Java como lenguaje de programación para la implementación de las clases que intervendrán en el procesamiento de las solicitudes enviadas desde el cliente y la generación de las respuestas; siguiendo el estilo arquitectónico REST mediante la especificación JAX-RS y el empleo de la biblioteca jersey. Y el *Front-end* con HTML5, CSS3, Bootstrap, JavaScript y JQuery. Además, se seleccionó AUP como metodología de desarrollo y UML como lenguaje de modelado.

2. Modelo del negocio y requisitos

2.1. Introducción

En el presente capítulo se describen los procesos del sistema correspondiente al software que se desea implementar contendiente a las fases de análisis y diseño. Exponiéndose los requerimientos y las funcionalidades que debe cumplir la aplicación para tener la confiabilidad y calidad esperada por los usuarios finales.

2.2. Actores del sistema

El actor es una idealización de un rol desempeñado por una persona externa. En un sistema juega un papel por cada caso de uso con el que colabora, representan terceros fuera del sistema que colaboran con el mismo (Rumbaugh et al., 2007). A continuación en la Tabla 3 se presenta el actor del sistema y su descripción.

TABLA 3 ACTORES DEL SISTEMA

Actor	Descripción
Usuario	El usuario es el responsable de crear, editar y eliminar un proyecto, además de ejecutar los mismos requisitos en las diferentes versiones de un diagrama, también puede invitar a otros usuarios para observar diagramas elaborados por este, los cuales solo pueden editar versiones existentes y crear nuevas.

2.3. Especificación de los requisitos funcionales

Los requerimientos son una descripción de las necesidades de un producto. El objetivo de la fase de requerimientos es identificar y documentar lo que en realidad se necesita. El reto consiste en definirlos de manera inequívoca, de modo que se detecten los riesgos en tiempo (Larman, 1999). A continuación en la Tabla 4 se presentan los requisitos funcionales identificados y definidos para el sistema.

TABLA 4 ESPECIFICACIÓN DE REQUISITOS FUNCIONALES DEL SISTEMA

#RF	Nombre	Descripción
RF1.	Gestionar usuario.	Funcionalidad que abarca el registro, cambio de contraseña y autenticación de un usuario.
RF1.1.	Registrar cuenta.	Funcionalidad que permite el registro de un usuario.
RF1.2.	Cambiar contraseña.	Funcionalidad que permite el cambio de contraseña de un usuario.
RF1.3.	Autenticar usuario.	Funcionalidad que permite la autenticación de un usuario en el sistema.
RF2.	Gestionar proyecto.	Funcionalidad que abarca la inserción, eliminación y edición de un proyecto de un usuario.
RF2.1.	Insertar proyecto.	Funcionalidad que permite la inserción de un proyecto.
RF2.2.	Eliminar proyecto.	Funcionalidad que permite la eliminación de un proyecto.
RF2.3.	Editar proyecto.	Funcionalidad que permite la edición de un proyecto.
RF3.	Gestionar versión.	Funcionalidad que abarca la inserción, eliminación y edición de una versión de un diagrama.
RF3.1.	Insertar versión.	Funcionalidad que permite la inserción de una versión.
RF3.2.	Eliminar versión.	Funcionalidad que permite la eliminación de una versión.
RF3.3.	Editar versión.	Funcionalidad que permite la edición de una versión.
RF4.	Exportar diagrama.	Funcionalidad que permite al usuario la exportación del modelo de un diagrama ERE a un XML.

2.4. Especificación de los requisitos no funcionales

A continuación se identifican los principales requisitos no funcionales del sistema:

RNF1.Seguridad:

- La aplicación estará funcionando a tiempo completo; de esta forma es posible que los usuarios tengan acceso, según sus permisos, en todo momento a la información solicitada.
- Se empleara un certificado SSL (acrónimo del inglés de *Secure Sockets Layer*) para asegurar la conexión, lo cual hace que todo el tráfico generado sea seguro de atacantes externos, previniendo especialmente el ataque MITM (acrónimo del inglés de *Man In The Middle*).

RNF2.Hardware:

El servidor Web y de base de datos deben tener como mínimo las siguientes características:

- Computador Core i3-4170.
- 4 GB de memoria RAM DDR-3.
- 10 GB de espacio libre en disco duro.

El navegador de las computadoras clientes requiere como mínimo en hardware.

- Computador Pentium.
- 2 GB de memoria RAM DDR-3.

RNF3.Software:

- Gestor de bases de datos PostgresSQL versiones superiores a la 9.5.15.
- Servidor Web Apache Tomcat 8.0.32.
- Las máquinas clientes deben contar con navegadores como Mozilla Firefox versiones superiores a la 65.0.1 y de Google Chrome versiones superiores a la 73.0.1.
- JDK 8.0 (acrónimo del inglés de *Java Development Kit*).
- JRE 8.0 (acrónimo del inglés de *Java Runtime Environment*).

RNF4.Rendimiento:

- El sistema debe tener un tiempo de respuesta rápido y eficiente, inferior a 15 segundos.
- La aplicación debe de ser capaz de soportar gran cantidad de usuarios conectados simultáneamente.

RNF5.Apariencia o interfaz externa:

- El diseño de la interfaz debe simular una aplicación de escritorio.

- De manera general se pretende que la aplicación muestre una interfaz agradable, sencilla, sugerente e intuitiva de modo que los usuarios puedan interactuar fácilmente con el sistema.

RNF6.Usabilidad:

- La interfaz será fácil de usar para los usuarios que interactúen con ella.
- El sistema estará bien documentado, con el fin de lograr un mejor uso de los servicios que este ofrecerá, para ello se debe realizar una manual de usuario que explique paso a paso cada una de las funcionalidades del software.

2.5. Paquetes y sus relaciones

Los Paquetes constituyen el mecanismo que permite describir los grupos de cualquier tipo de elementos o subsistemas de un modelo, ya sea clases, casos de uso u otros paquetes anidados. Un sistema puede examinarse íntegramente dentro del ámbito de un solo paquete de alto nivel. Además se define al diagrama de paquetes como la representación de agrupamientos lógicos mediante los diagramas de paquetes de UML (Larman, 1999). (Ver Figura 1)

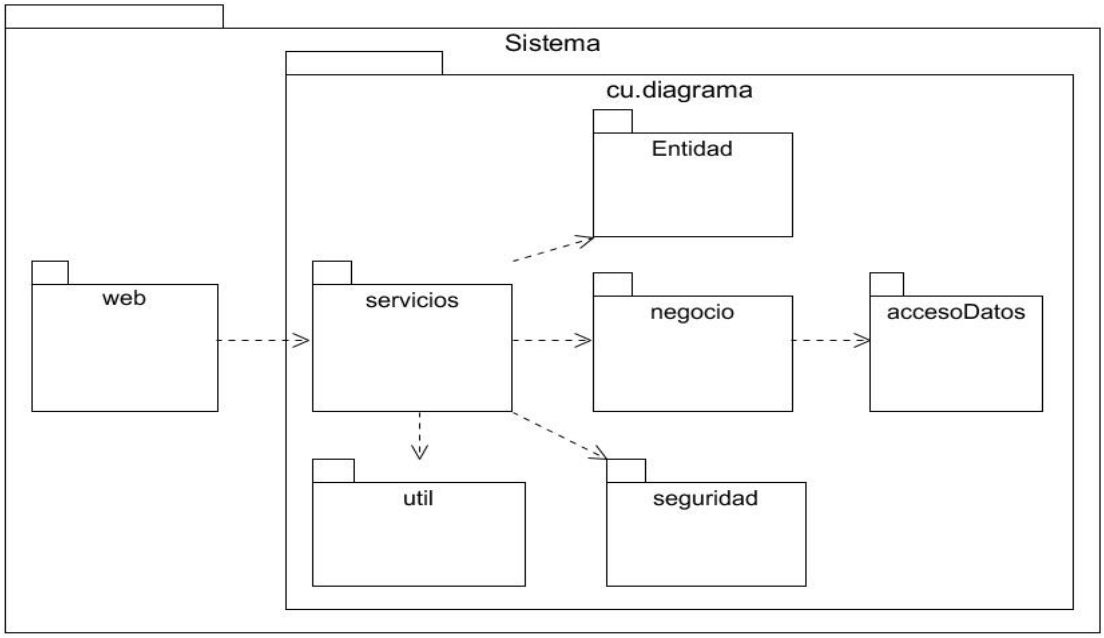


FIGURA 1 DIAGRAMA DE PAQUETES

A continuación se describen los paquetes que se encuentran anidados dentro del paquete principal Sistema:

- ***Paquete cu.diagrama*:** constituye la capa arquitectónica de la lógica y los servicios de la aplicación.
- ***Paquete web*:** constituye la capa de presentación de la aplicación.
- ***Paquete accesoDatos***: permitirá ejecutar las consultas mediante el driver JDBC (acrónimo del inglés de *Java Database Connectivity*) para PostgreSQL.
- ***Paquete negocio***: este paquete maneja la lógica principal de la aplicación, además de que actúa de intermediario entre los paquetes *servicios* y *accesoDatos*.
- ***Paquete servicios***: está constituido por clases en Java clases con los servicios asociados a las entidades.
- ***Paquete seguridad*:** está constituido por clases en Java con las restricciones de seguridad de la aplicación.
- ***Paquete util*:** está constituido por clases en Java donde se encuentran las utilidades de la aplicación.
- ***Paquete entidad*:** está constituido por clases en Java de todas las entidades o conceptos del dominio persistentes en la aplicación.

2.6. Modelo del dominio o conceptual

El modelo del dominio permite mostrar de manera visual abstracciones que se manejan en el contexto en que se desarrolla un sistema a través de la identificación de personas, eventos y objetos involucrados en el mismo. El modelo constituye la representación de conceptos en un dominio del problema, muestra asociaciones entre los conceptos, logra el análisis y la descomposición del problema en unidades comprensibles. Este modelo contribuye a comunicar a los desarrolladores los términos más importantes de un problema y cómo se relacionan entre sí (Larman, 1999).

En la Figura 2 se muestra el diagrama de modelo de dominio realizado, donde se identificaron los principales objetos involucrados en el dominio del problema. Se identificaron al *Usuario*, *Proyecto*, *Versión* y al *Modelo* como objetos del dominio.

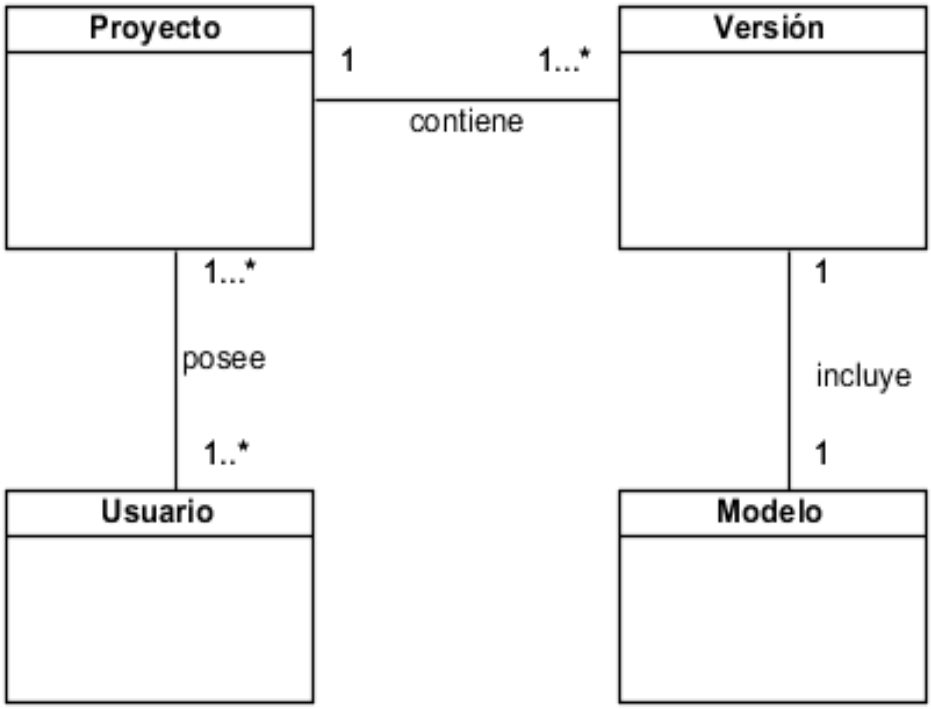


FIGURA 2 MODELO DEL DOMINIO O CONCEPTUAL

2.7. Diagrama de Casos de Uso del Sistema

Larman (1999) caracteriza los diagrama de Casos de Uso cuando expresa:

> *Un diagrama de casos de uso explica gráficamente un conjunto de casos de uso de un sistema, los actores y la relación entre éstos y los casos de uso. (…) El diagrama tiene por objeto ofrecer una clase de diagrama contextual que nos permite conocer rápidamente los actores externos de un sistema y las formas básicas en que lo utilizan.*

En la Figura 3 se muestran el diagrama de caso de uso del sistema, donde se muestra al actor del sistema y su relación con los casos de uso *Gestionar usuario*, *Gestionar proyecto*, *Gestionar versión* y *Exportar modelo*. Además cada una de las secciones que posee cada caso de uso. Se identificaron como los más significativos los caso de uso *Gestionar versión* y *Gestionar proyecto*.

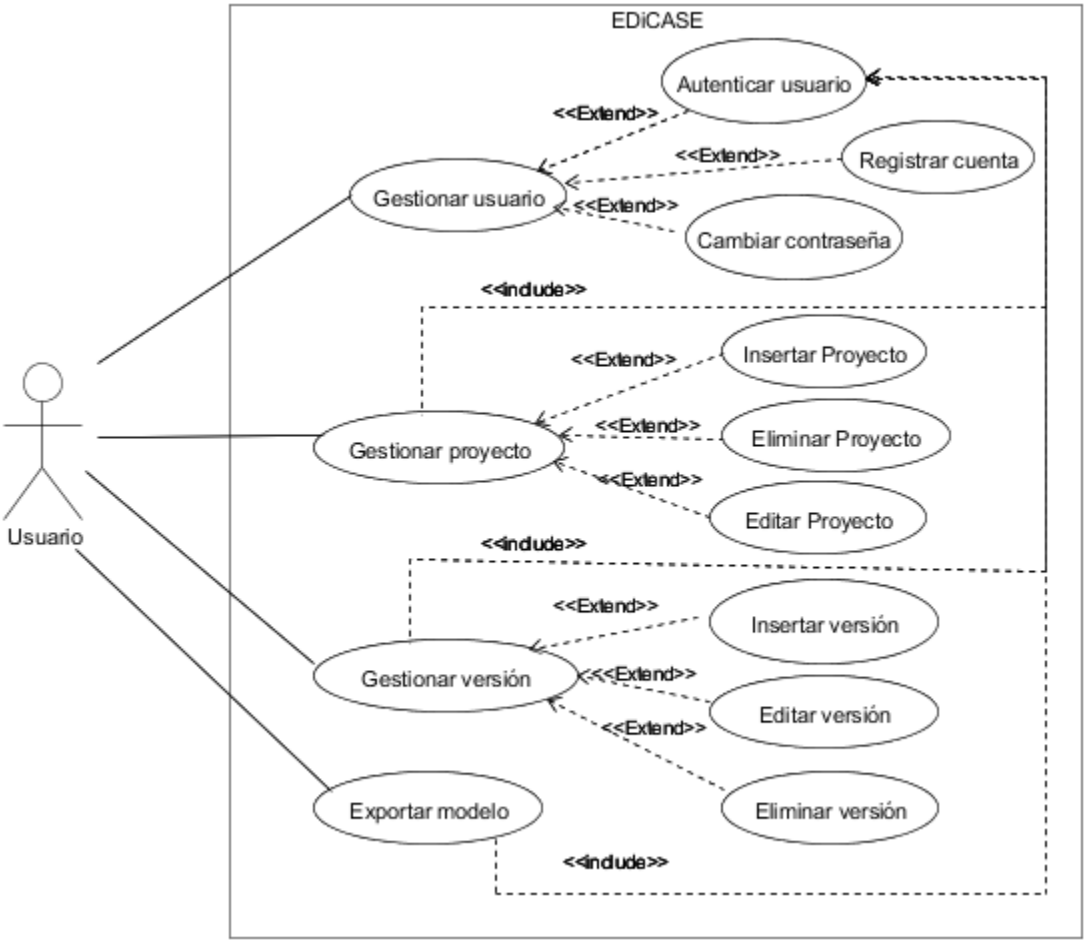


FIGURA 3 DIAGRAMA DE CASOS DE USO DEL SISTEMA

2.8. Descripción de los casos de uso del Sistema.

La descripción de los casos de uso del sistema constituye una parte muy útil, ya que se explica de forma detallada la interacción entre el sistema y el usuario. Para la realización de los prototipos de interfaz se utilizó la herramienta WireframeSketcher en su versión 4.6.0. A continuación se presentan los casos de uso más significativos (Ver Tabla 5 y 6, las demás descripciones se localizan en los Anexos Tablas 19 y 18):

TABLA 5 CASO DE USOS DEL SISTEMA GESTIONAR PROYECTO

Caso de Uso:	Gestionar Proyecto.
Actores:	Usuario.

Resumen:	El caso de uso inicia cuando un usuario se autentica en el sistema. El usuario inserta los datos de un proyecto en el sistema. También se brinda la posibilidad de eliminar y editar los datos de un proyecto, así como mostrar sus principales datos.
Precondiciones:	El usuario debe encontrarse registrado y autenticado en el sistema.
Referencias:	RF2.1, RF2.2, RF2.3, RF3.1.
Prioridad:	Alta
Flujo Normal de Eventos	
Sección "Insertar Proyecto**"**	
Acción de Actor	**Respuesta del Sistema**
1. Selecciona la opción de Insertar Proyecto en la página principal.	2. Muestra una ventana con los siguientes datos a insertar: • Nombre del proyecto. • Descripción del proyecto. • Permiso del proyecto. • Usuarios Permitidos.
3. Inserta los datos especificados por el sistema. 4. Presiona el botón Aceptar.	5. Muestra los resultados de la inserción en una tabla con los siguientes datos. • Nombre del proyecto. • Descripción del proyecto. • Permiso del proyecto. • Versión. • Creador del proyecto. • Fecha de Creación del proyecto.

	6. Termina el caso de uso.
Flujos Alternos	
3. El actor selecciono la opción Cancelar.	
Acción de Actor	**Respuesta del Sistema**
	3.1. Se cierra la ventana de inserción de un proyecto. 3.2. Termina el caso de uso.
3. El actor no inserta los datos especificados por el sistema y presiona el botón de Aceptar.	
Acción de Actor	**Respuesta del Sistema**
	3.1. Muestra un aviso que debe llenar el campo Nombre del proyecto obligatoriamente. 3.2. Regresa al paso 3 del flujo de trabajo.
Prototipo de Interfaz	

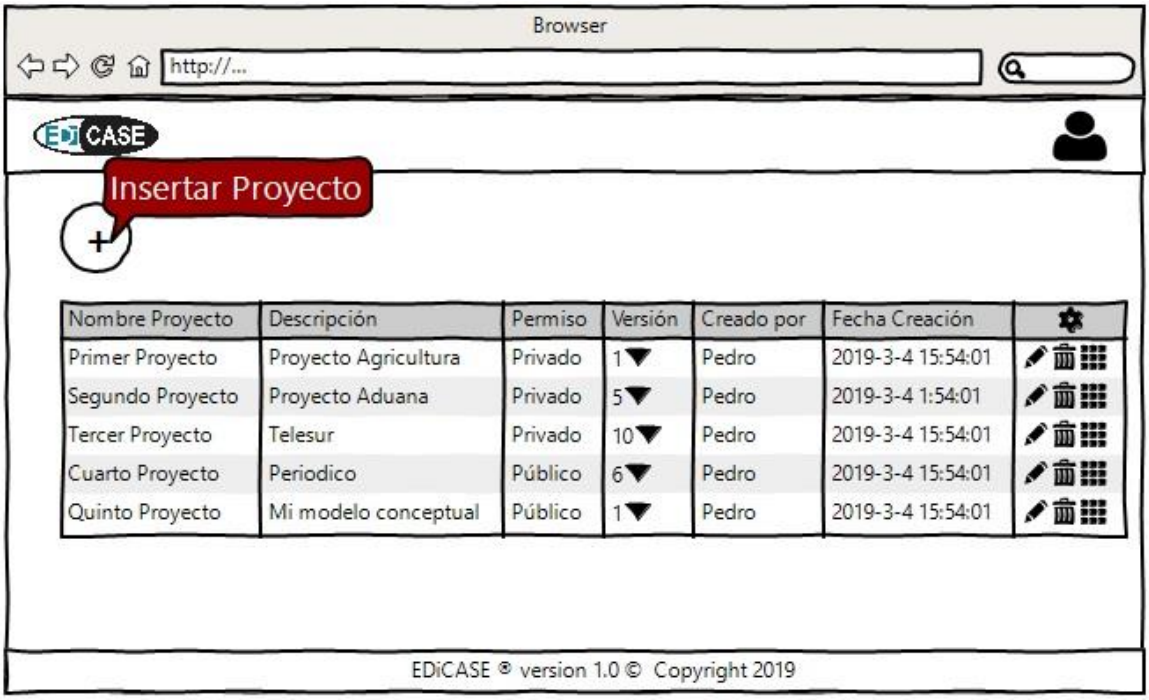


FIGURA 4 PROTOTIPO LISTAR PROYECTO

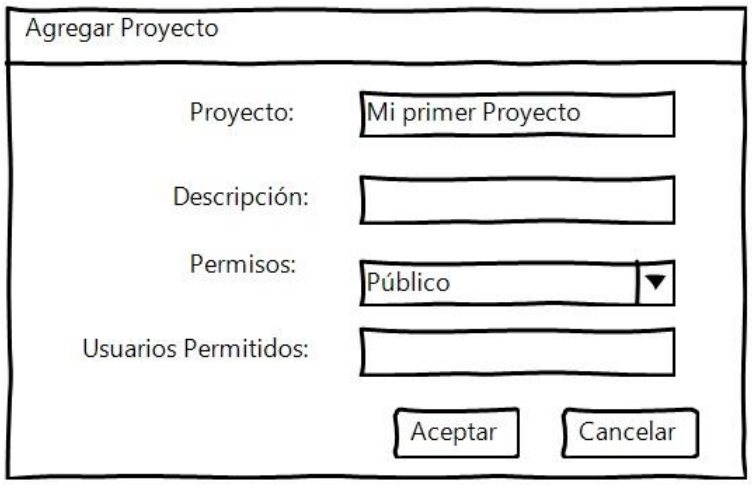


FIGURA 5 PROTOTIPO AGREGAR PROYECTO

Flujo Normal de Eventos	
Sección "Eliminar Proyecto**"**	
Acción de Actor	**Respuesta del Sistema**

1. Selecciona la opción eliminar, en un proyecto determinado.	2. Muestra una ventana con la botones siguientes: • Borrar Todo. • Borrar Versión.
3. Presiona el botón Eliminar Todo.	4. Elimina todos los datos del proyecto. 5. Termina el caso de uso.
Flujos Alternos	
1. El actor selecciono la opción Cancelar.	
Acción de Actor	**Respuesta del Sistema**
	1.1. Se cierra la ventana de eliminación de un proyecto. 1.2. Termina el caso de uso.
Prototipo de Interfaz	

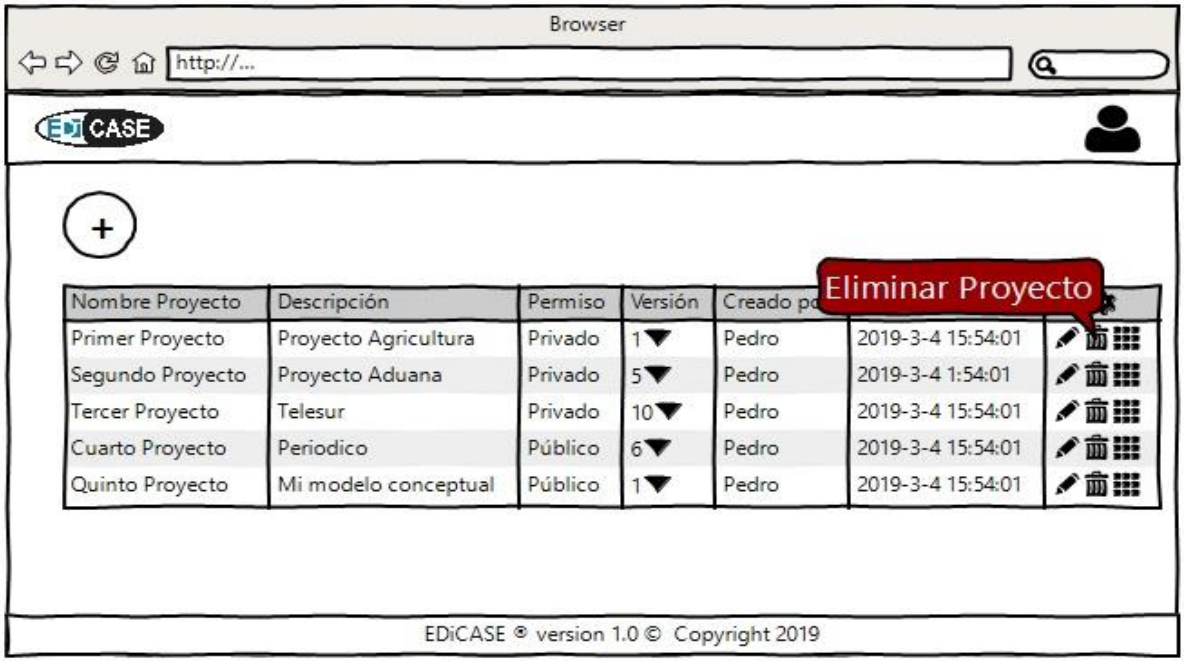


FIGURA 6 PROTOTIPO ELIMINAR PROYECTO

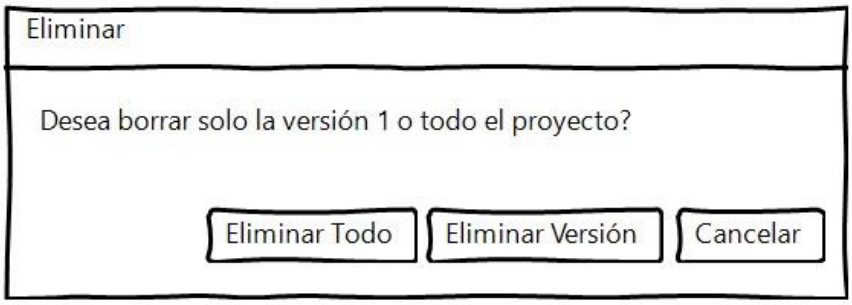


FIGURA 7 PROTOTIPO DIÁLOGO ELIMINAR PROYECTO

Flujo Normal de Eventos	
Sección "Editar Proyecto**"**	
Acción de Actor	**Respuesta del Sistema**
1. Selecciona la opción editar, en un proyecto determinado.	2. Muestra una ventana con los datos del proyecto a editar: • Nombre del proyecto. • Descripción del proyecto. • Permiso del proyecto. • Usuarios Permitidos.
3. Edita los campos de un proyecto. 4. Selecciona la opción Aceptar.	5. Muestra los resultados actualizados de un proyecto en la tabla de proyectos. 6. Termina el caso de uso.
Flujos Alternos	
4. El actor selecciono la opción cancelar.	
Acción de Actor	**Respuesta del Sistema**
	4.1. Se cierra la ventana de edición de un proyecto. 4.2. Termina el caso de uso.
3. El actor al editar el nombre del proyecto deja el campo vacío y presiona el botón de Aceptar.	

Acción de Actor	Respuesta del Sistema
	3.1. Muestra un aviso que debe llenar el campo Nombre del proyecto obligatoriamente. 3.2. Regresa al paso 3 del flujo de trabajo.
Prototipo de Interfaz	

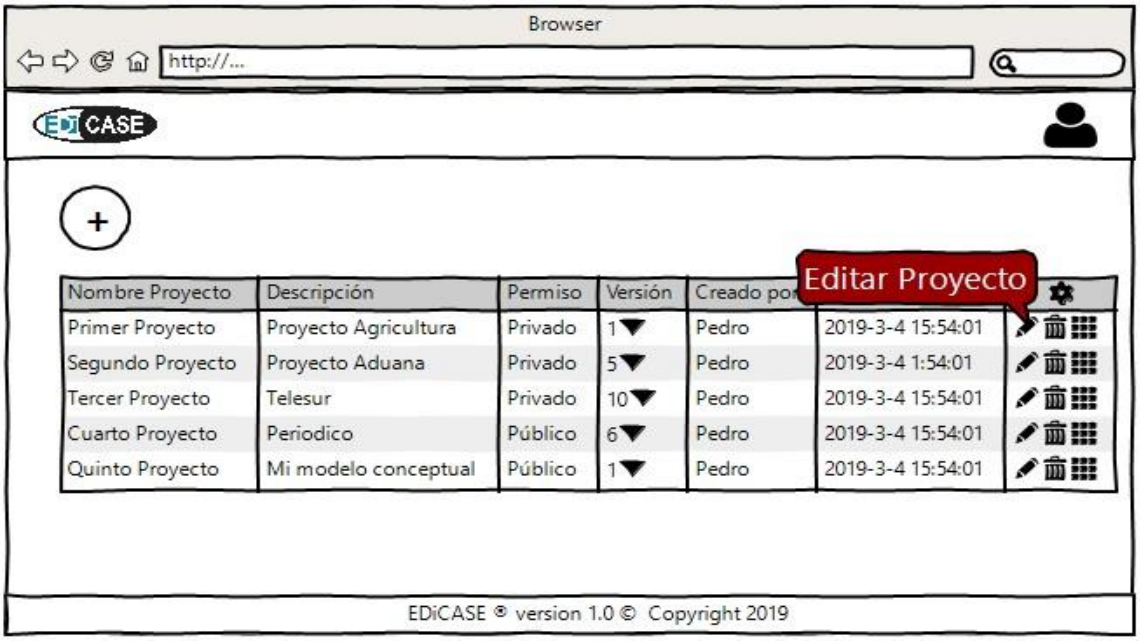


FIGURA 8 PROTOTIPO EDITAR PROYECTO

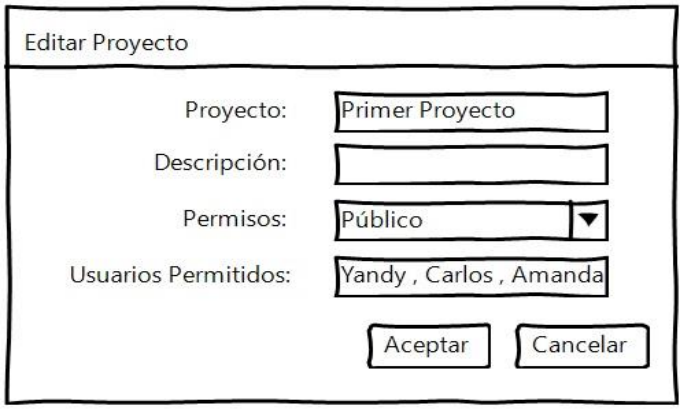


FIGURA 9 PROTOTIPO DIÁLOGO EDITAR PROYECTO

TABLA 6 CASO DE USOS DEL SISTEMA GESTIONAR VERSIÓN

Caso de Uso:	Gestionar Versión.
Actores:	Usuario.
Resumen:	El caso de uso inicia cuando un usuario accede a la página de modelación de diagramas. El usuario tiene la posibilidad de crear una nueva versión o actualizarla. Para eliminar una versión deberá retornar a la página de proyectos.
Precondiciones:	El usuario debe encontrarse registrado y autenticado en el sistema.
Referencias:	RF3.1, RF3.2, RF3.3.
Prioridad:	Crítico
Flujo Normal de Eventos	
Sección "Insertar Versión**"**	
Acción de Actor	**Respuesta del Sistema**

1. Selecciona la opción de Salvar versión en la página de modelación.	2. Muestra la ventana Salvar Versión con el siguiente campo los siguientes botones: • Actualizar • Nueva • Cancelar
3. Presiona el botón Nueva.	4. Cierra la ventana Salvar Versión y muestra la ventana Nueva Versión con el siguiente campo a llenar: • Descripción
5. Completa la descripción de la versión a insertar. 6. Presiona el botón Aceptar.	7. Cierra la ventana. 8. Termina el caso de uso.
Flujos Alternos	
3. El actor presiona Cancelar.	
Acción de Actor	**Respuesta del Sistema**
	3.1. Cierra la ventana. 3.2. Termina el caso de uso.
5. El actor presiona Cancelar.	
	5.1. Cierra la ventana. 5.2. Termina el caso de uso.
Prototipo de Interfaz	

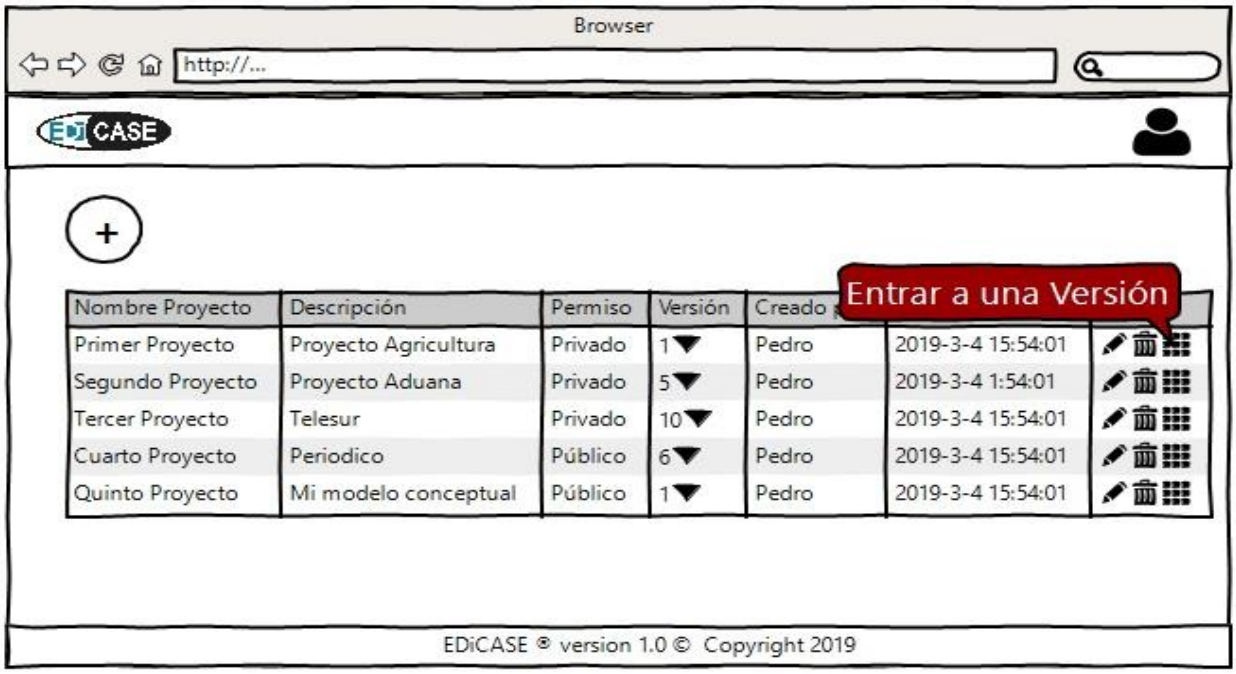


FIGURA 10 PROTOTIPO CARGAR VERSIÓN

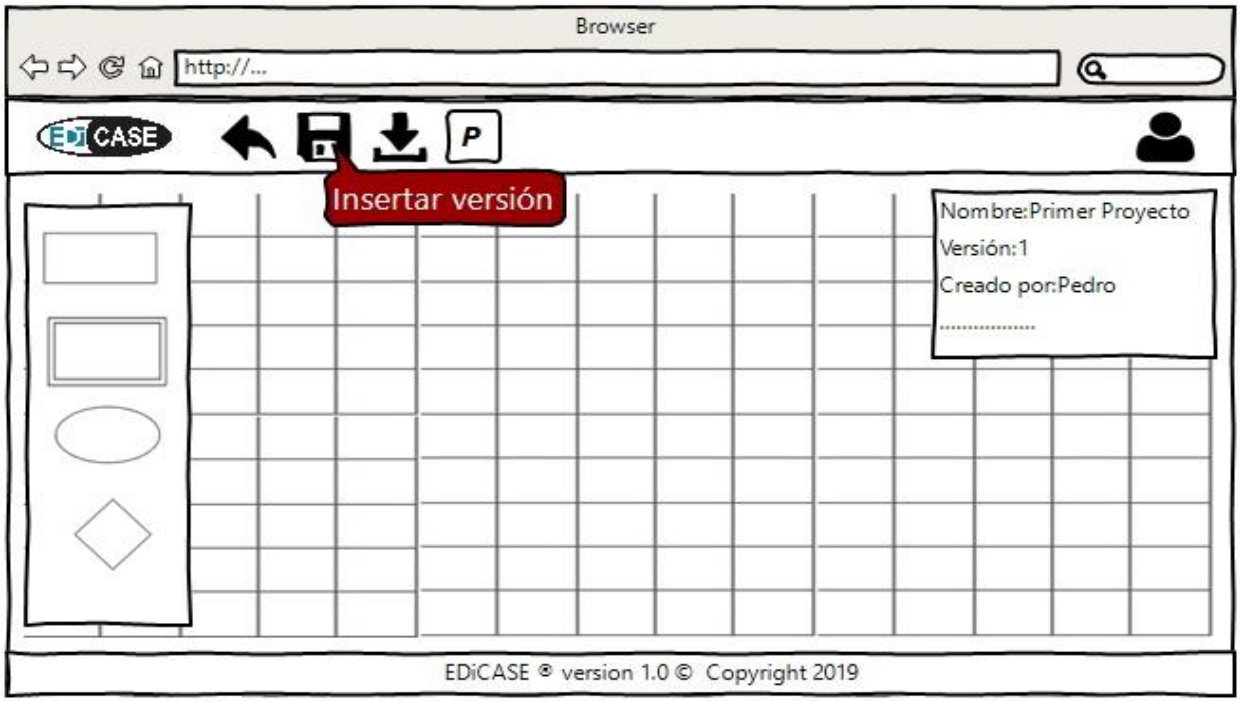


FIGURA 11 PROTOTIPO INSERTAR VERSIÓN

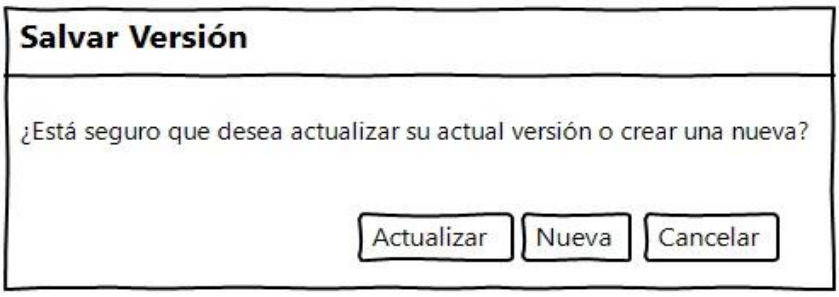


FIGURA 12 PROTOTIPO SALVAR VERSIÓN

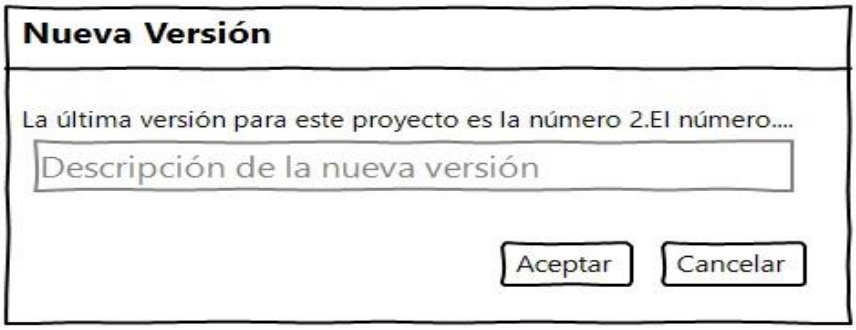


FIGURA 13 PROTOTIPO NUEVA VERSIÓN

Flujo Normal de Eventos	
Sección "Actualizar Versión**"**	
Acción de Actor	**Respuesta del Sistema**
1. Selecciona la opción de Salvar versión en la página de modelación.	2. Muestra la ventana Salvar Versión con los siguientes botones: • Actualizar • Nueva • Cancelar
3. Presiona el botón Actualizar.	4. Actualiza la versión. 5. Se cierra la ventana Salvar Versión. 6. Termina el caso de uso.
Flujos Alternos	

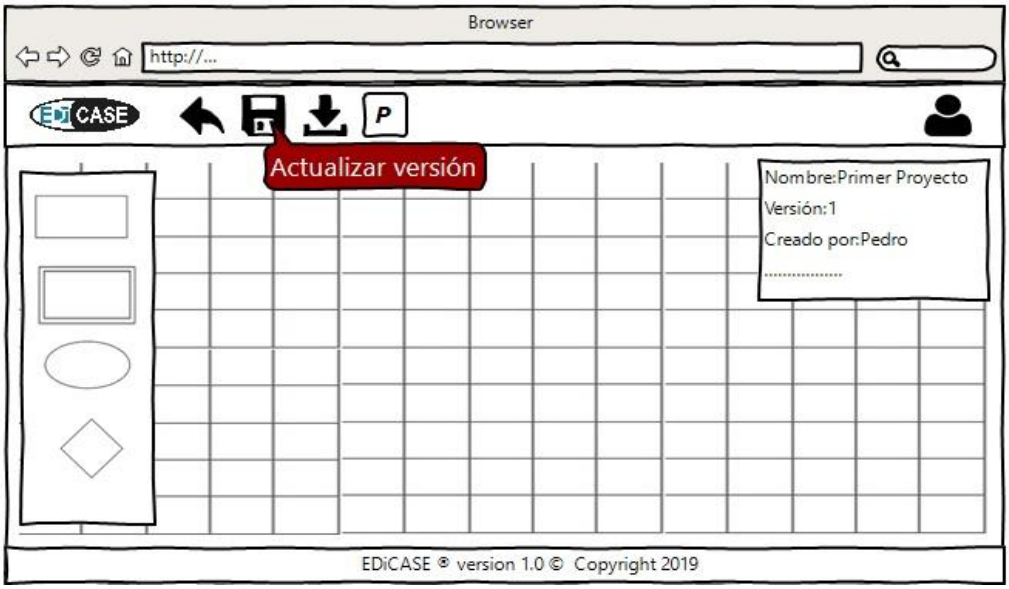
3. El actor presiona Cancelar.	
Acción de Actor	**Respuesta del Sistema**
	3.1. Cierra la ventana Salvar Versión. 3.2. Termina el caso de uso.
Prototipo de Interfaz	
	
Sección "Eliminar Versión**"**	
Flujo Normal de Eventos	
Acción de Actor	**Respuesta del Sistema**
1. Presiona el botón Volver a Proyectos.	2. Redirige la Interfaz a la página principal mostrando todos los proyectos :
3. Presiona el botón Eliminar Proyecto en la tabla de proyectos.	4. Muestra una ventana Eliminar con la opciones: • Eliminar Todo. • Eliminar Versión.

FIGURA 14 PROTOTIPO ACTUALIZAR VERSIÓN.

	• Cancelar
5. Presiona el botón Eliminar Versión	6. Muestra una notificación “La versión fue eliminada correctamente”. 7. Termina el caso de uso.
Flujos Alternos	
3. El actor presiona el botón Cancelar.	
Acción de Actor	**Respuesta del Sistema**
	3.1. Se cierra la ventana Eliminar. 3.2. Termina el caso de uso.
Prototipo de Interfaz	

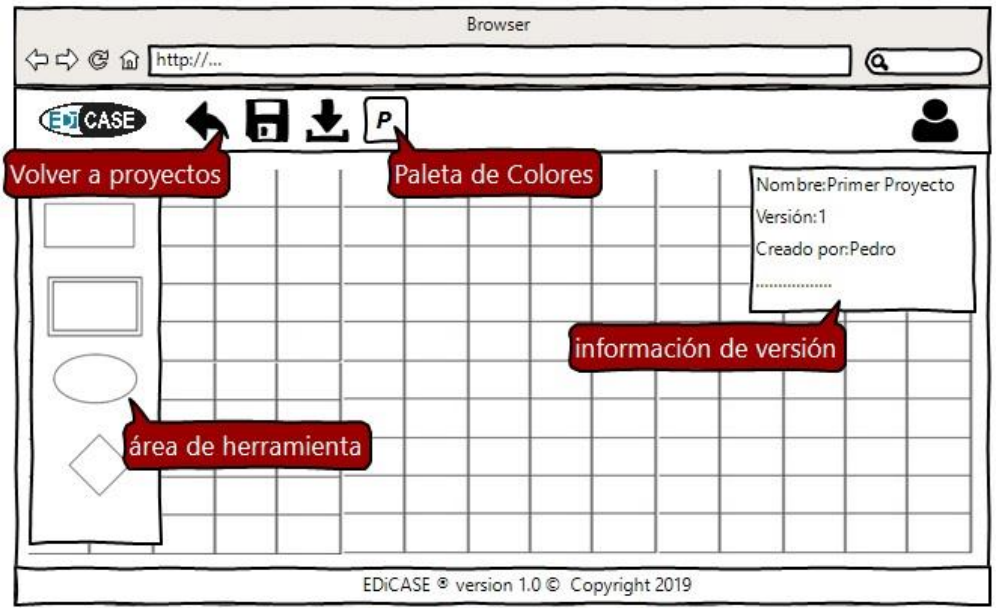


FIGURA 15 PROTOTIPO VOLVER A PROYECTO.

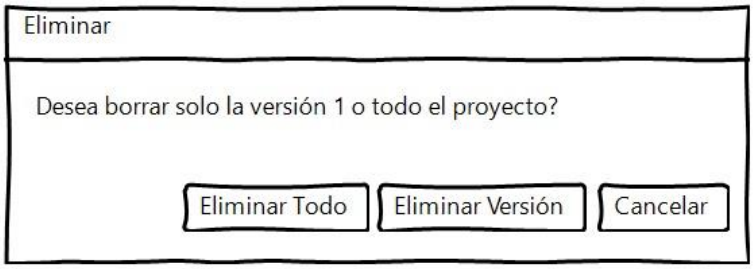


FIGURA 16 PROTOTIPO ELIMINAR VERSIÓN.

2.9. Conclusiones

En este capítulo se realizó un estudio detallado de la descripción del problema que sirvió como guía para la identificación del actor que va a intervenir en el sistema, además de las funcionalidades y requerimientos que debe cumplir la aplicación. Se describió la estructura lógica del sistema para una mejor comprensión de su funcionamiento y se determinaron los casos de uso del sistema, especificando los más significativos. Se ofreció una visión general del problema y de lo que será el producto, constituyendo la base fundamental para la realización del análisis y el diseño.

3. Descripción de la propuesta de solución

3.1. Introducción

A partir de la descripción de las principales características de la aplicación expuestas en el capítulo anterior, se pretende transformar los requisitos identificados en un diseño de lo que será el sistema. Por tanto, el presente capítulo tiene como objetivo la realización del modelo de análisis con las clases que lo componen, el diagrama de clases del diseño de los casos de uso más significativos, los diagramas de secuencia, así como el modelo conceptual, lógico y físico de los datos.

3.2. Arquitectura del Sistema

Según Pressman (2010) “Un estilo arquitectónico es una transformación que se impone al diseño de todo el sistema. El objetivo es establecer una estructura para todos los componentes del sistema”.

En el caso del que se desea implementar, cuenta con la arquitectura cliente/servidor. Esta arquitectura es propia del desarrollo de aplicaciones Web, y tiene como objetivo optimizar el uso tanto de hardware como del software a través de la separación de funciones: el cliente, que maneja la porción de la aplicación y el servidor que administra los procesos de almacenamiento y recuperación de los datos. El *Back-end* de la aplicación a implementar estará estructurado en una arquitectura en capas tradicional (Ver Figura 17).

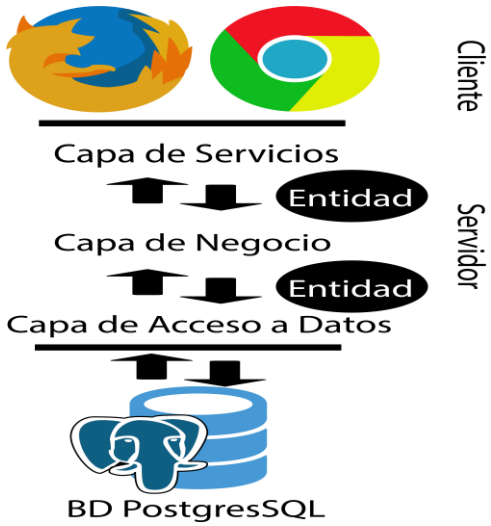


FIGURA 17 ARQUITECTURA DEL SISTEMA (FUENTE: ELABORACIÓN PROPIA)

Estructura de la arquitectura en capas:

- La *Capa de Servicios* está basada en el patrón arquitectónico REST con especificación JAX-RS utilizando la librería Jersey para definir los puntos de entrada a la aplicación.
- La *Capa de Negocio* estará completamente construida en Java, sin el uso de ninguna librería especial, su función será manejar la lógica principal de la aplicación, además que actuará de intermediaria entre la capa de Servicios y de Acceso a Datos.
- La *Capa de Acceso a Datos* permitirá ejecutar las consultas mediante el driver JDBC para PostgreSQL.
- Las *Entidades* son clases Java comúnmente conocidas como POJO (acrónimo del inglés *Plain Old Java Object*).

3.3. Plugin JQuery: Diagrama

Es deseable optar por desarrollar plugins que sean de gran utilidad para aplicaciones Web. Sobre todo si se escribe código muy similar varias veces, es una señal de que se debería dedicar tiempo a producir un complemento que pueda reutilizarse fácilmente. Por tal razón se usa JQuery, el cual es un framework muy versátil que permite ser extendido para así poder crear nuevas funciones.

En la aplicación, Plugin Diagrama se encarga de dibujar los diferentes elementos del modelo. Fue desarrollado con JQuery por sus amplias posibilidades para el manejo de elementos HTML y eventos que pueden ser aplicados. Se trata de un interesante desarrollo que permite:

- Dibujar, editar entidades especificando su nombre y validando que no existan nombres repetidos, además también posibilita eliminar las entidades agregadas así como sus atributos.
- Dibujar relaciones especificando una o más entidades que estarán vinculadas a la relación, se verifican si se trata de una relación débil o fuerte por las entidades involucradas, además es posible editar y eliminar las relaciones agregadas así como sus atributos.
- Dibujar relaciones de generalización con la posibilidad de especificar si es Total o Parcial y Solapada o Disjunta, así como especificar la entidad padre y entidades hijas involucradas, también es posible borrar este tipo de relaciones.

- Dibujar atributos que solo pueden ser asociados a entidades y relaciones normales, que puedan ser editados y eliminados, es posible solo a través de la entidad o la relación que se encuentra vinculado.
- Leer o guardar un JSON que proviene de la base de datos como parte de un atributo más, que se presenta para el manejo de las versiones. Este JSON trae las especificaciones necesarias para que el plugin pueda pintar los diferentes elementos del modelo y de esa misma forma pueda ser almacenado para futuras visualizaciones del usuario.
- Personalizar los colores de los atributos, entidades y relaciones.

El desarrollo de todas las funcionalidades que presenta esta herramienta de modelado solo fue posible con el uso conjunto de CSS3, JavaScript, HTML5 y JQuery en el manejo de los elementos del DOM. Con ello se logró convertir los elementos HTML dibujados en el área gráfica a un JSON para luego ser almacenado en la base de datos y, de forma inversa, también traducir un JSON a elementos gráficos HTML. Cualquier modificación deseada en el modelo solo sería a nivel del plugin desarrollado.

3.4. Diagrama de clases de diseño

El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Normalmente contiene clases, asociaciones, atributos, interfaces, métodos, navegabilidad y dependencias (Larman, 1999). A continuación se presentan los diagramas de clases de diseño más significativos (Ver Figuras 18, 19, 20, 21, 22 y 23 los demás en los Anexos Figuras 49, 50, 51 y 52).

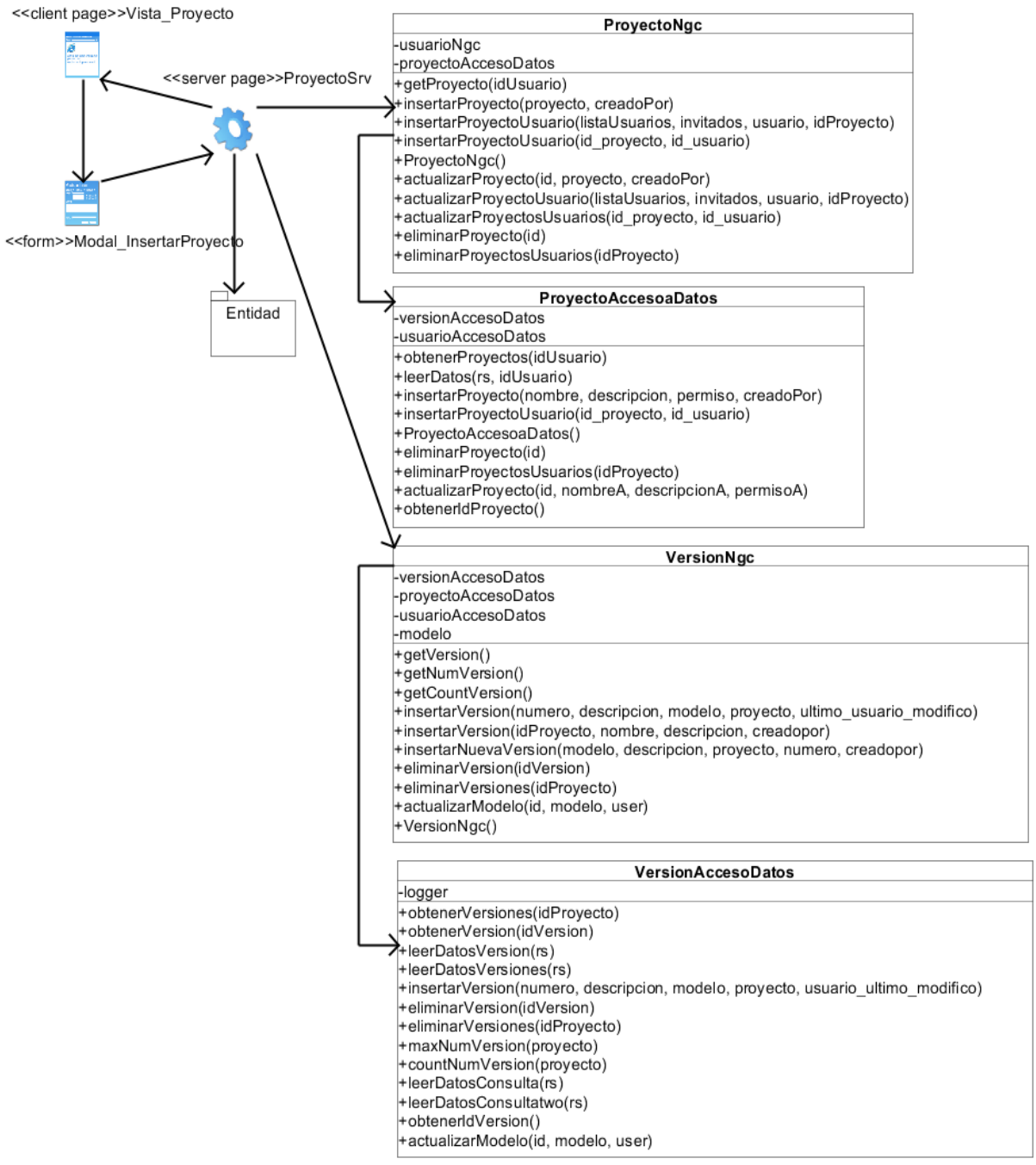


FIGURA 18 DIAGRAMA DE CLASE DE DISEÑO: INSERTAR PROYECTO

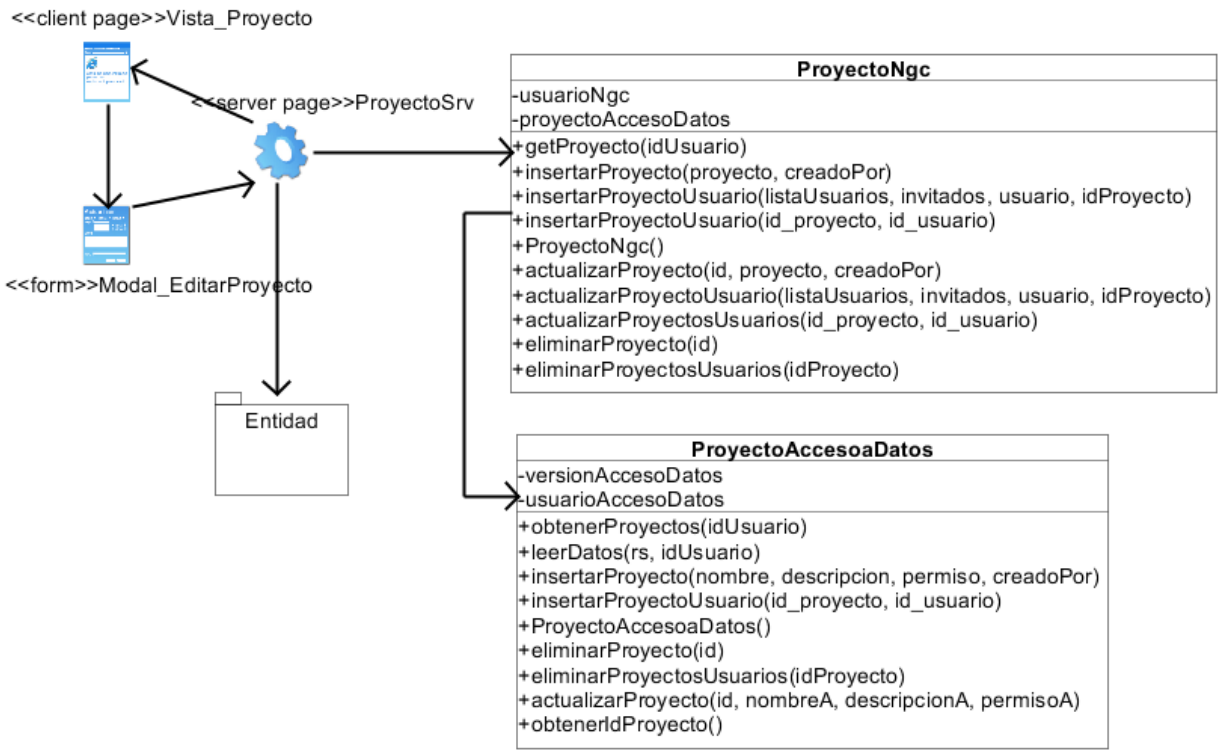


FIGURA 19 DIAGRAMA DE CLASE DE DISEÑO: EDITAR PROYECTO

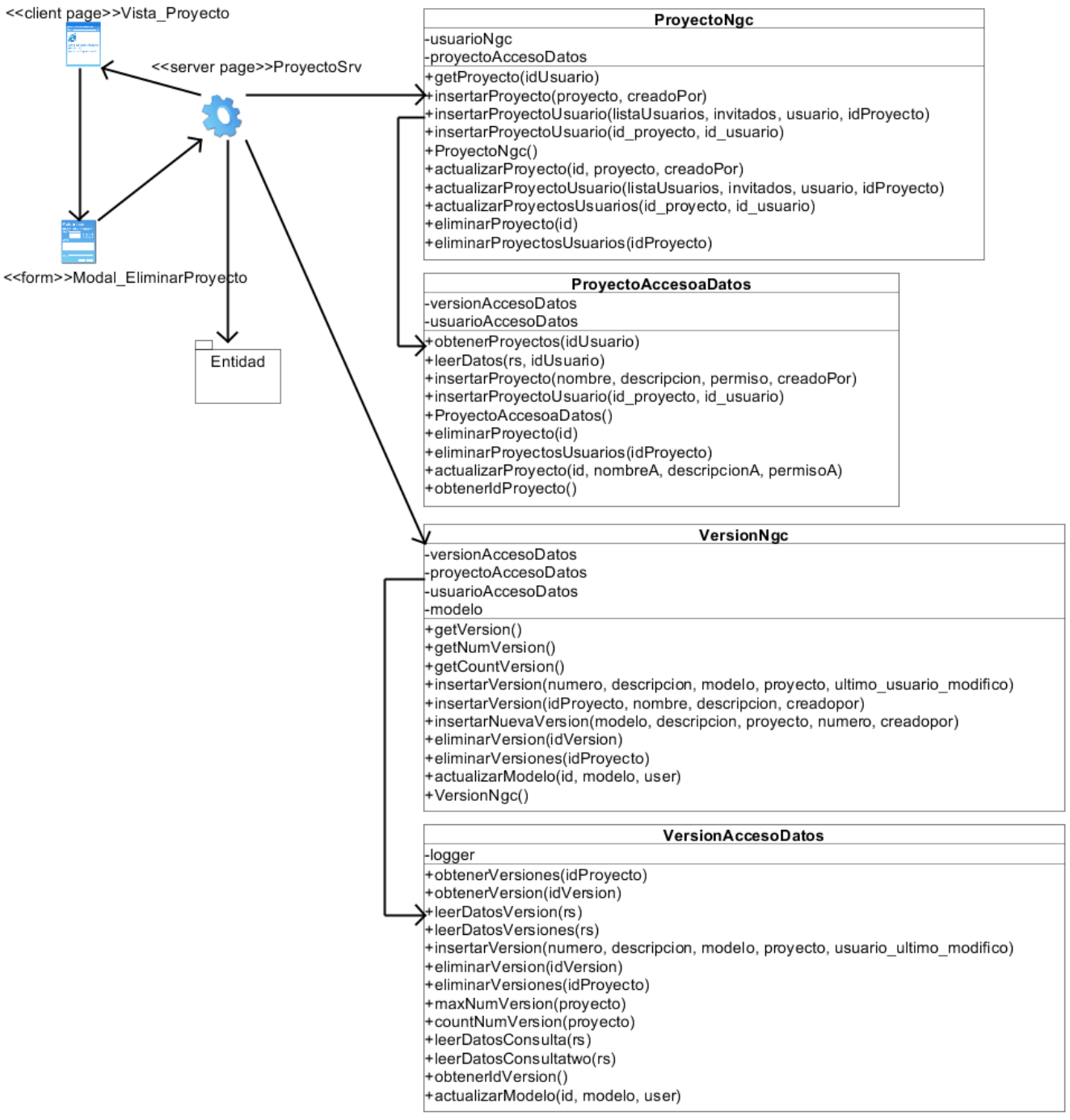


FIGURA 20 DIAGRAMA DE CLASE DE DISEÑO: ELIMINAR PROYECTO

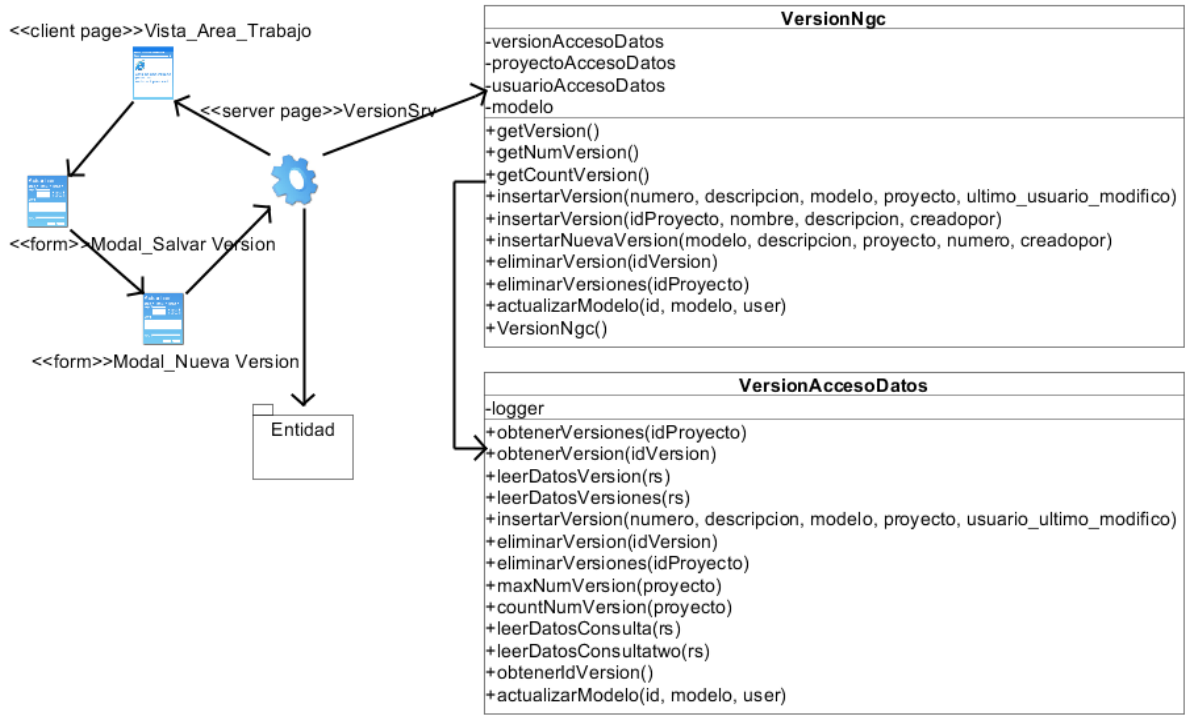


FIGURA 21 DIAGRAMA DE CLASE DE DISEÑO: INSERTAR VERSIÓN

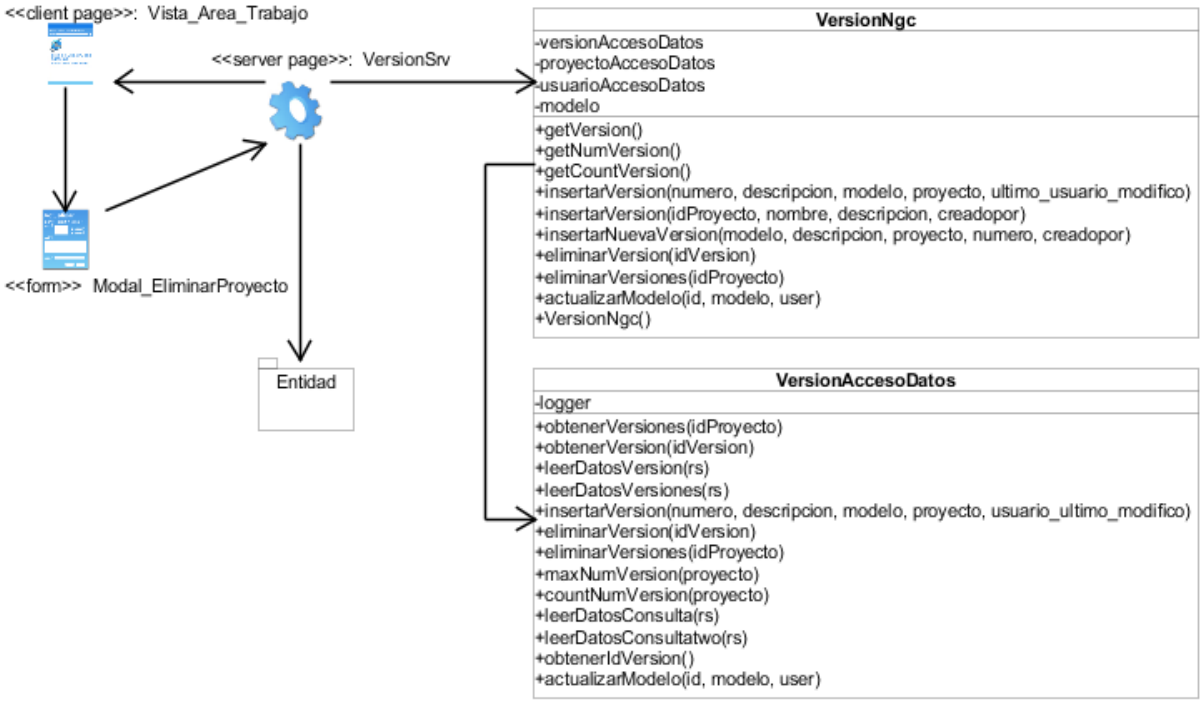


FIGURA 22 DIAGRAMA DE CLASE DE DISEÑO: EDITAR VERSIÓN

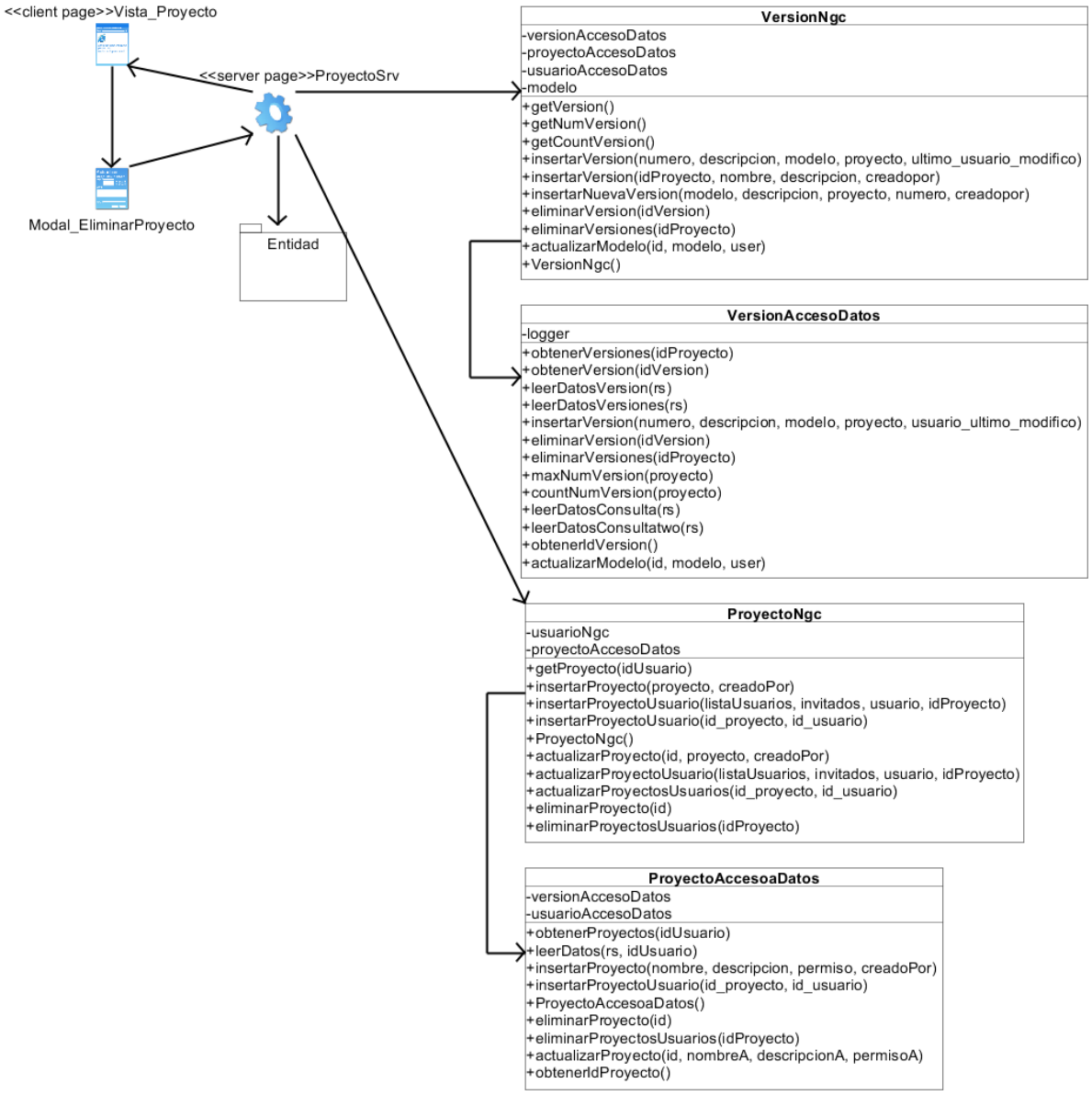


FIGURA 23 DIAGRAMA DE CLASE DE DISEÑO: ELIMINAR VERSIÓN

3.5. Diagrama de secuencia

El diagrama de secuencia constituye la representación del modo en que eventos provocan transiciones de un objeto a otro a través del tiempo. Permite observar la perspectiva cronológica de las transiciones. Esta información es útil en la generación de un diseño eficaz para el sistema que se va a construir (Pressman, 2010).

En las Figuras 24, 25 y 26 para el caso de uso Gestionar Proyecto sección: insertar, eliminar y editar, se evidencia la relación entre los objetos que intervienen en esta acción donde el

usuario es quien inicia y concluye esta secuencia de acciones. Al igual que en las Figuras 27, 28 y 29 para el caso de uso Gestionar Versión sección: insertar, eliminar y editar. Estos casos de uso constituyen los más significativos (Ver Anexos los demás diagramas de secuencias Figuras 53, 54, 55 y 56).

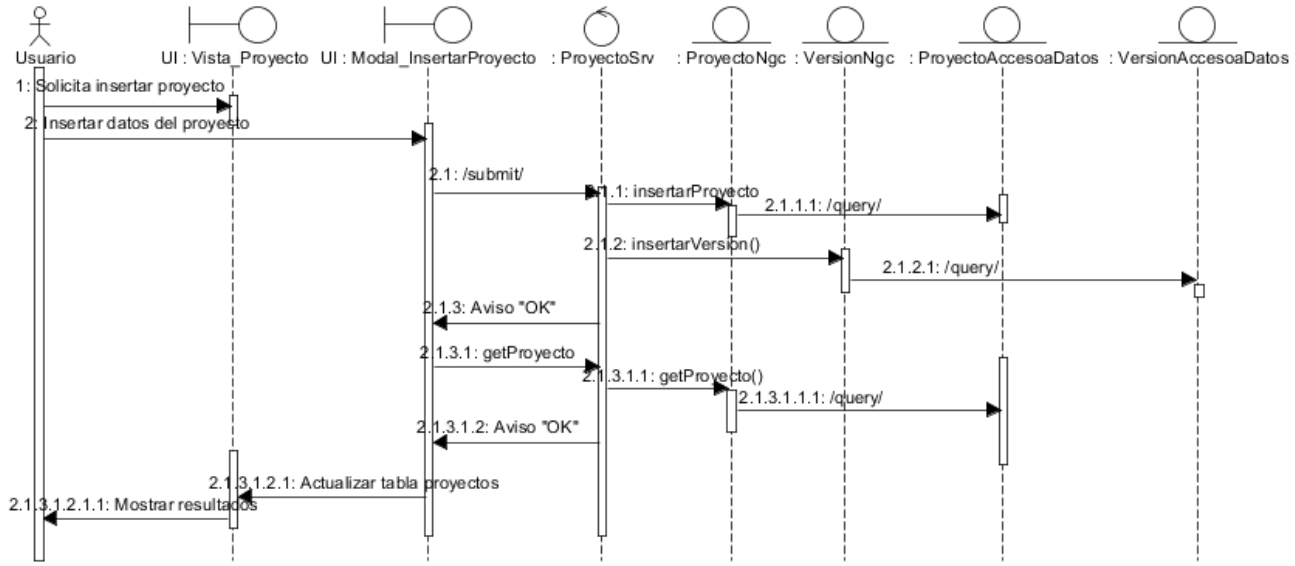


FIGURA 24 DIAGRAMA DE SECUENCIA SECCIÓN: INSERTAR PROYECTO

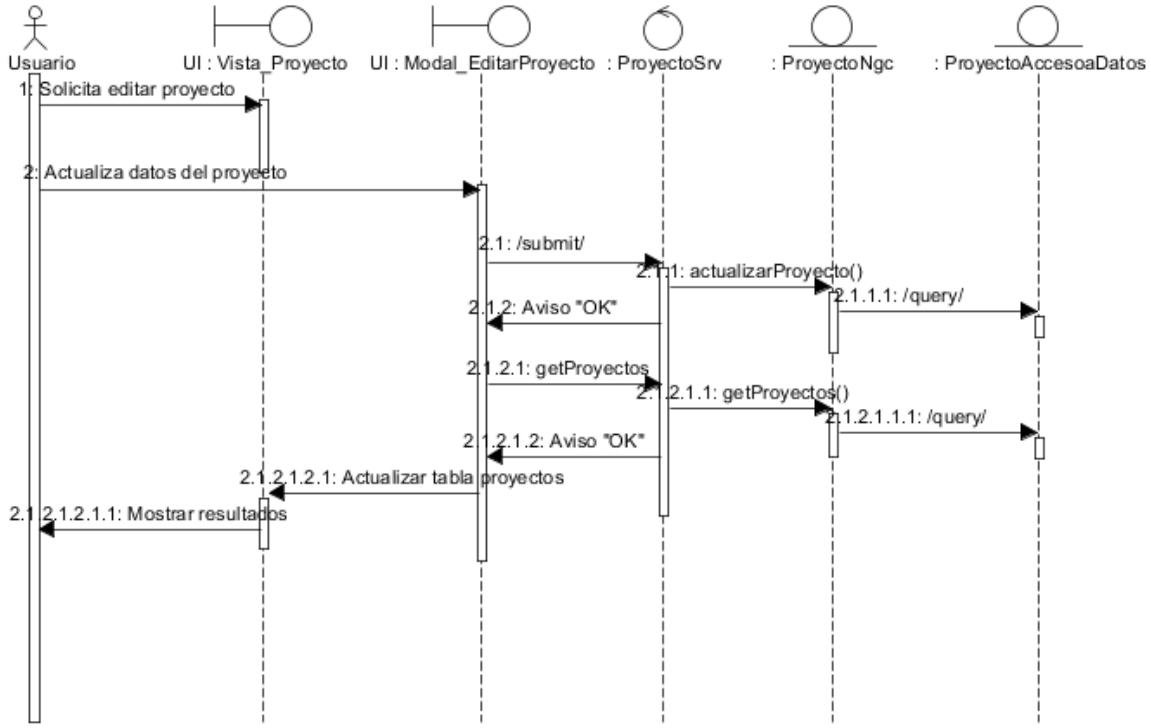


FIGURA 25 DIAGRAMA DE SECUENCIA SECCIÓN: EDITAR PROYECTO

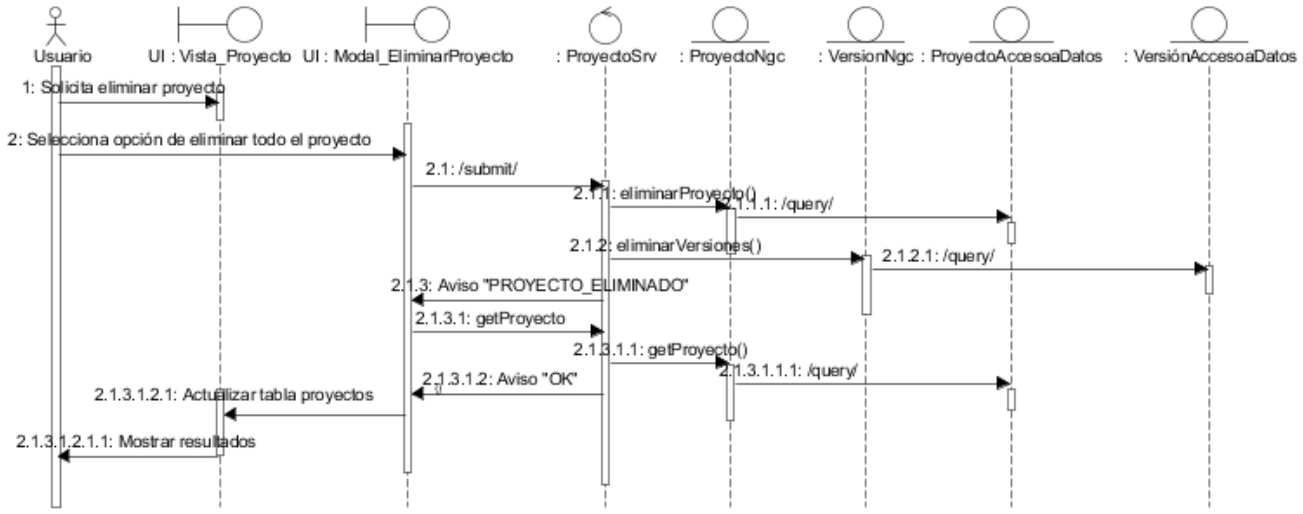


FIGURA 26 DIAGRAMA DE SECUENCIA SECCIÓN: ELIMINAR PROYECTO

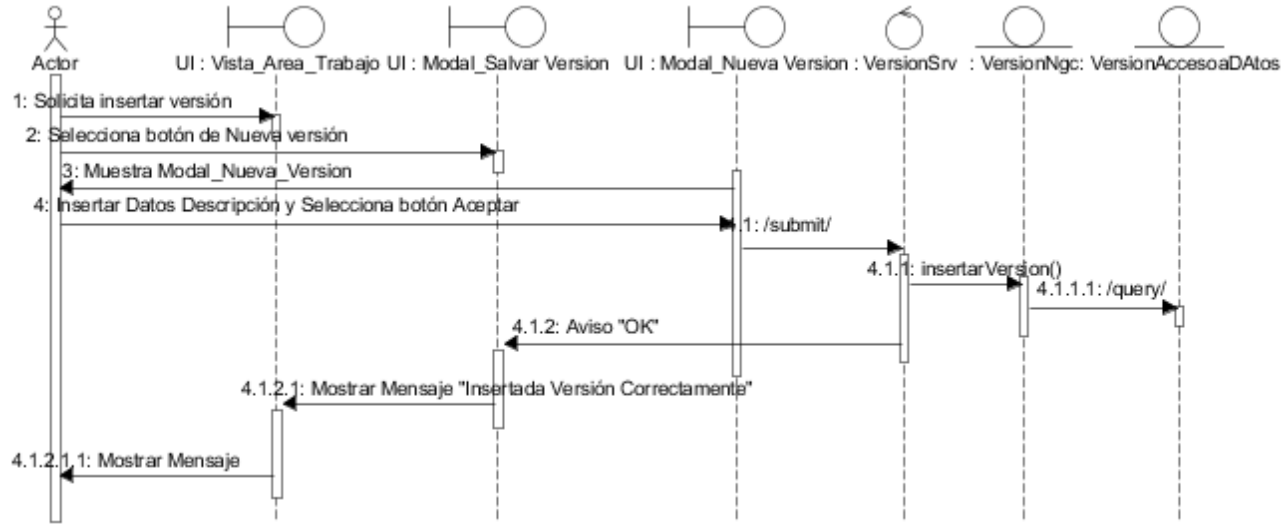


FIGURA 27 DIAGRAMA DE SECUENCIA SECCIÓN: INSERTAR VERSIÓN

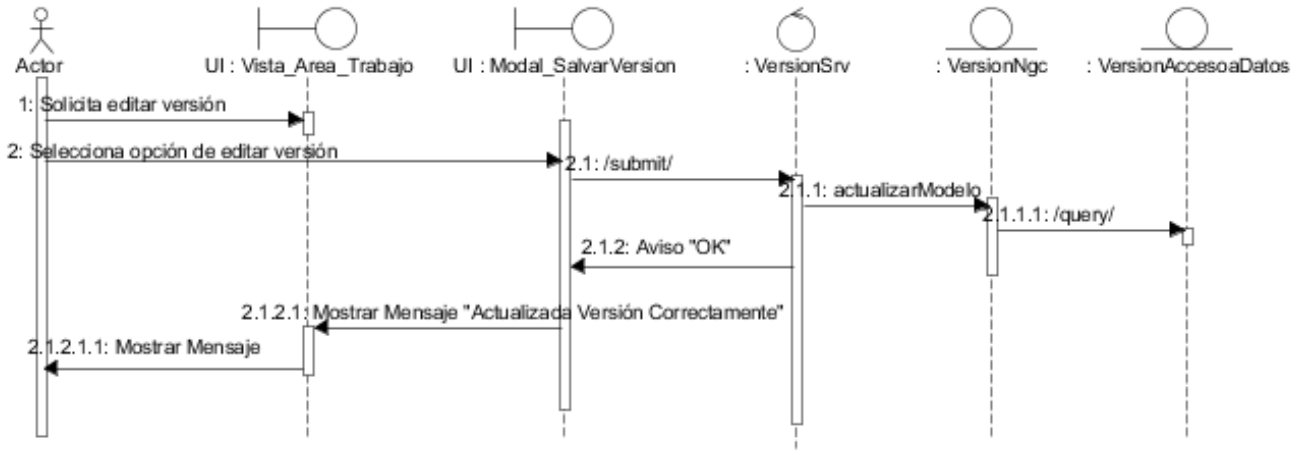


FIGURA 28 DIAGRAMA DE SECUENCIA SECCIÓN: EDITAR VERSIÓN

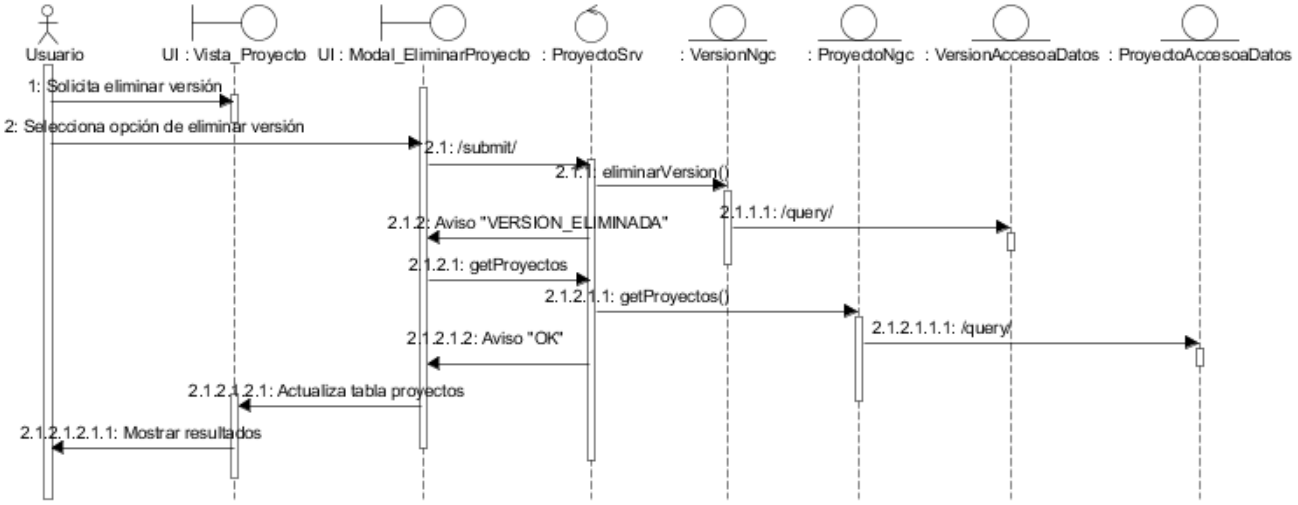


FIGURA 29 DIAGRAMA DE SECUENCIA SECCIÓN: ELIMINAR VERSIÓN

3.6. Diseño de la base de datos

3.6.1. Modelo conceptual de datos

El modelo conceptual está orientado a la descripción de estructuras de datos y restricciones de integridad. Se usa fundamentalmente durante la etapa de análisis de un problema dado, representando los elementos que intervienen y sus relaciones. El ejemplo más típico es el Modelo Entidad Relación Extendido (Elmasri and Navathe, 2007). En la Figura 30 se muestra el modelo conceptual de la base de datos realizado con la propia herramienta EDiCASE.

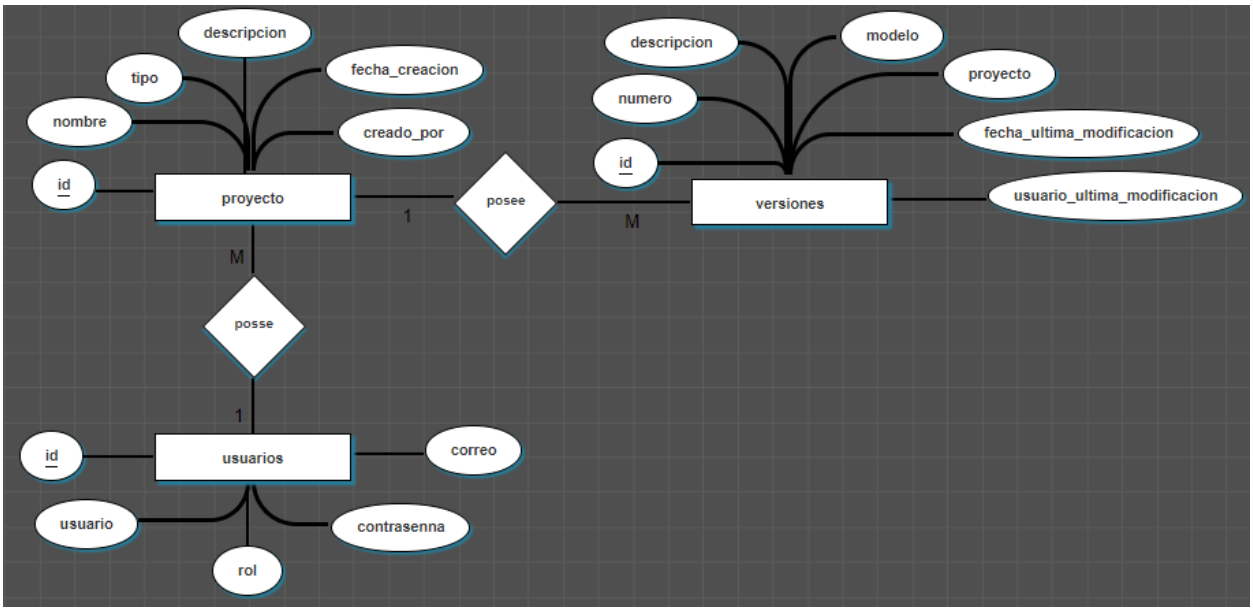


FIGURA 30 MODELO CONCEPTUAL DE DATOS

3.6.2. Modelo lógico de datos

El modelo lógico está orientado a representar la base de datos como una colección de relaciones. El ejemplo más típico es el Modelo Relacional. Este busca obtener una representación del modelo conceptual que use de forma eficiente las facilidades de estructuración de datos y modelado de restricciones, disponibles en el modelo (Elmasri and Navathe, 2007)(Ver Figura 31).

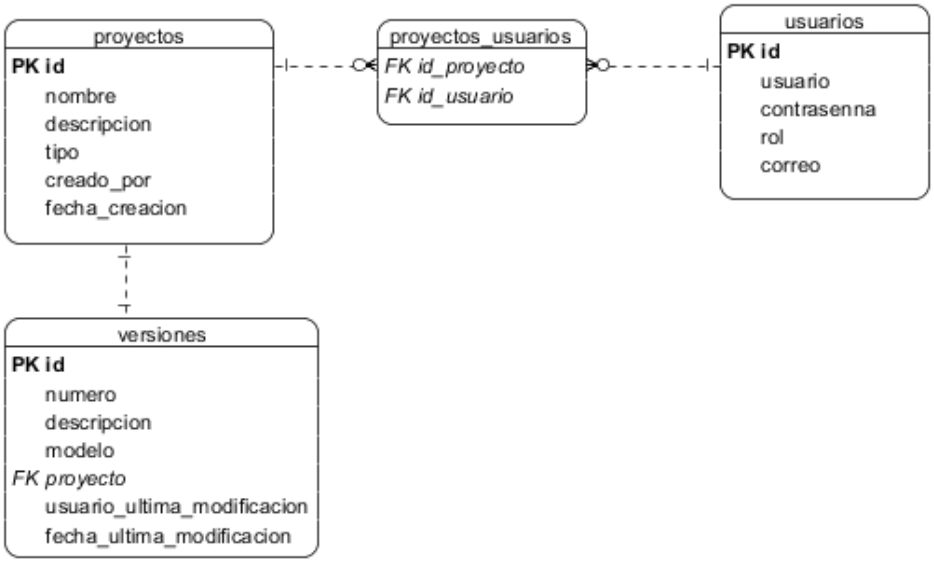


FIGURA 31 MODELO LÓGICO DE DATOS

3.6.3. Modelo físico de datos

El modelo físico constituye la representación de las relaciones bases y las estructuras adecuadas de almacenamiento. Además de los métodos de acceso que se utilizaran para acceder a los datos de modo eficiente. Para realizar el modelo físico se transforma la estructura obtenida en la etapa del diseño lógico (Elmasri and Navathe, 2007).

3.7. Modelo de componentes

Los diagramas de componentes muestran la separación de un sistema de software en componentes físicos y las dependencias entre ellos. Los componentes forman la arquitectura del software, juegan un papel en el logro de los objetivos y los requerimientos del sistema (Pressman, 2010). En la Figura 57 en el Anexo se encuentra el diagrama de componentes general donde se encuentra los siguientes paquetes y archivos:

- El paquete *plugins* contiene los archivos *table.js* y *graphic.js* los cuales constituyen los ficheros para la creación de los componentes visuales más importantes de la aplicación; el primero se encarga del control de la tabla para la gestión de los proyectos, mientras el segundo construye la interfaz gráfica para la modelación de los diagramas.
- El paquete *jsNegocio* contiene los ficheros *Login.js*, *Proyecto.js* y *Diagrama.js* los cuales contienen todas las peticiones del cliente.
- El paquete *servicios* constituye la capa de Servicios de la aplicación, por tanto gestiona la información que es enviada desde el paquete *jsNegocio*, contiene los ficheros *ProyectoSrv.java*, *UsuarioSrv.java* y *VersionSrv.java*.
- El paquete *negocio* constituye la capa de Negocio de la aplicación, contiene los ficheros *ProyectoNgc.java*, *UsuarioNgc.java*, *VersionNgc.java*.
- El paquete *accesoDatos* constituye la capa de Acceso a Datos, contiene los ficheros *ProyectoAccesoDatos.java*, *UsuarioAccesoDatos.java* y *VersionAccesoDatos.java*.
- El paquete *dominio* contiene el paquete entidad es donde se localizan los ficheros *Proyecto.java*, *Usuario.java* y *Version.java* los cuales constituyen las Entidades de la aplicación.

3.8. Diagrama de despliegue

Los diagramas de despliegue describen la arquitectura física del sistema durante la ejecución, en términos de: procesadores, dispositivos y componentes de software. Básicamente este tipo

de diagrama se utiliza para modelar el hardware utilizado en la implementación del sistema y las relaciones entre sus componentes (Rumbaugh et al., 2007).

En la Figura 32 se representan los aspectos más importantes a destacar, que son:

- La *PC Cliente* constituye el puesto donde el usuario estará interactuando con la aplicación, este nodo tendrá una conexión mediante <<HTTPS>>con el nodo *Servidor Web*.
- El *Servidor Web* será el encargado de responder todas las peticiones hechas por el cliente, mediante el servicio REST y la comunicación con la base de datos, para obtener y brindar los datos solicitados por el usuario.

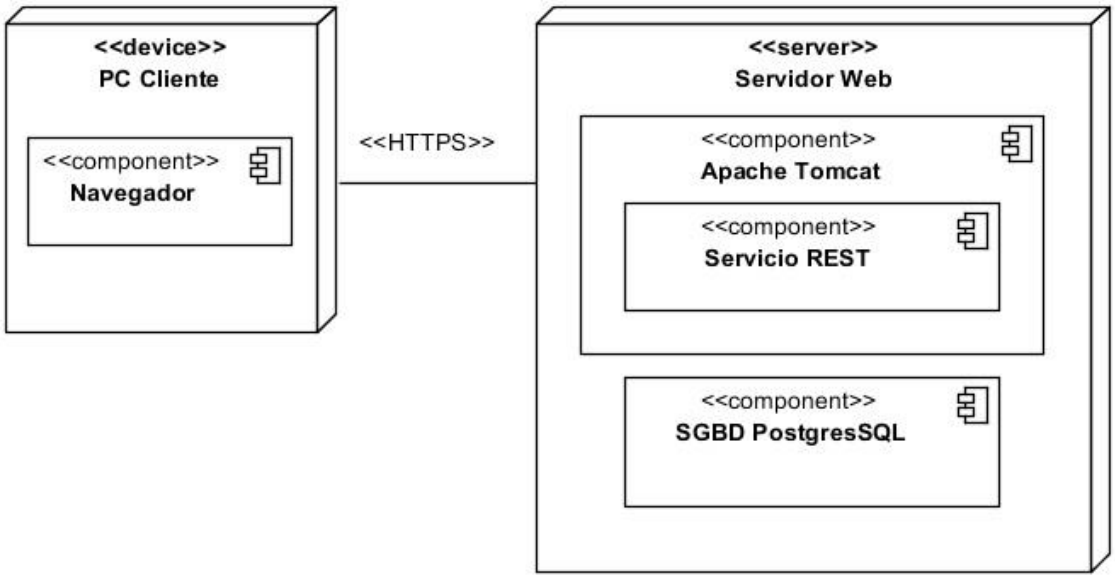


FIGURA 32 DIAGRAMA DE DESPLIEGUE

3.9. Conclusiones

En este capítulo se analizó la arquitectura de sistema, el *plugin* desarrollado para la modelación de los diagramas Entidad Relación Extendido. Se desarrollaron los elementos correspondientes al modelo de implementación: diagramas de clases de diseño y de secuencias de los casos de uso más significativos, diagrama de componentes general y de despliegue. Además se analizó los diagramas conceptual y lógico para el diseño de la base de datos. Al concluir se completaron los artefactos fundamentales que constituyen la base de la aplicación EDiCASE.

4. Pruebas y análisis de factibilidad

4.1. Introducción

Los estudios de factibilidad son investigaciones altamente enfocadas en un proyecto, diseñados para producir información crítica sobre la viabilidad de implementar el sistema de software, y con base en ello decidir si se va a llevar a cabo o no. En este capítulo se determina la estimación y cálculo de tamaño del software, así como el esfuerzo y los costos que reportará el sistema realizado. Además se describe la estrategia de pruebas a realizar y se muestran los resultados de las mismas aplicadas a la herramienta informática desarrollada, con el objetivo de validar sus funcionalidades.

4.2. Planificación

4.2.1. Puntos de casos de uso

Es un método de estimación y cálculo de tamaño del software basado en cuentas hechas sobre los casos de uso para un sistema de software. El método requiere de casos de uso en modo textual y gráfico, por lo que es necesario tener dominio del problema (Remón and Thomas, 2010).

4.2.1.1. Cálculo de puntos de casos de uso sin ajustar

El cálculo de los puntos de caso de uso sin ajustar constituye el primer paso y se calcula a partir de la siguiente ecuación (Ver Tabla 7):

TABLA 7 CÁLCULO DE PUNTOS DE CASOS DE USO SIN AJUSTAR

UUCP=UAW+UUCW	Donde, UUCP: Puntos de Casos de Uso sin ajustar. UAW: Factor de Peso de los Actores sin ajustar. UUCW: Factor de Peso de los Casos de Uso sin ajustar

Para obtener el Factor de Peso de los Actores sin Ajustar se debe tener en cuenta la cantidad de actores presentes en el sistema y la complejidad de los mismos. Los criterios se muestran en la siguiente tabla (Ver Tabla 8):

TABLA 8 FACTOR DE PESO DE LOS ACTORES SIN AJUSTAR

Tipo de Actor	**Descripción**	**Peso**	**Cantidad*Peso**
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación(API)	1	0 * 1
Medio	Otro sistema que interactúa con el sistema a desarrollar mediante un protocolo o una interfaz basada en texto.	2	0 * 2
Complejo	Una persona que interactúa con el sistema mediante interfaz grafica	3	1 * 3
Total			3

Para obtener el valor del factor de Peso de los Casos de Uso sin ajustar se debe tener en cuenta la cantidad de casos de uso y la complejidad de cada uno de ellos. Los criterios se muestran en la siguiente tabla (Ver Tabla 9):

TABLA 9 FACTOR DE PESO DE CASOS DE USO SIN AJUSTAR

Tipo de Caso de Uso	**Descripción**	**Peso**	**Cantidad*Peso**
Simple	El Caso de uso contiene 1 a 3 transacciones	5	1 * 5
Medio	El Caso de uso contiene 4 a 7 transacciones	10	2 * 10
Complejo	El Caso de uso contiene más de 8 transacciones	15	1 * 15

Total		40

Finalmente los casos de uso sin ajustar resultan:

UUCP = UAW + UUCW = 40 + 3 = **43**

4.2.1.2. Cálculo de puntos de casos de uso ajustados

Una vez obtenidos los puntos de caso de uso sin ajustar se precede al cálculo de los puntos de casos de uso ajustados mediante la siguiente ecuación (Ver Tabla 10).

TABLA 10 CÁLCULO DE PUNTOS DE CASOS DE USO AJUSTADOS

UCP=UUCP*TCF*EF	Donde, UCP: Puntos de casos de uso ajustado. UUCP: Puntos de casos de uso sin ajustar. TCF: Factor de complejidad técnica. EF: Factor de ambiente.

Factor de complejidad técnica: Este coeficiente se calcula mediante un conjunto de Factores que determinan la complejidad del sistema, cada factor se cuantifica con valor de 0 a 5, donde 0 significa un aporte irrelevante y 5 un aporte muy importante (Ver Tabla 11). La ecuación para su cálculo es:

TCF=0.6+0.01*∑ (Peso i *Valor Asignado i)

TABLA 11 VALORES DEL FACTOR DE COMPLEJIDAD TÉCNICA

Factor	Descripción	Peso	Peso*Valor
T1	Sistema distribuido.	2	2 * 0
T2	Objetivos de performance o tiempo de respuesta.	1	1 * 5
T3	Eficiencia de usuario final.	1	1 * 5
T4	Procesamiento interno complejo.	1	1 * 1

T5	El código debe ser reutilizable.	1	1 * 4
T6	Facilidad de instalación.	0.5	0.5 * 4
T7	Facilidad de uso.	0.5	0.5 * 4
T8	Portabilidad.	2	2 * 4
T9	Facilidad de cambio.	1	1 * 4
T10	Concurrencia	1	1 * 4
T11	Incluye objetivos especiales de seguridad.	1	1 * 3
T12	Provee acceso directo a terceras partes.	1	1 * 2
T13	Se requieren facilidades especiales de entrenamiento a los usuarios.	1	1 * 3
Total			43

Entonces, TCF = 0.6 + 0.01 * 43 = **1.03**

Factor de ambiente: contempla las habilidades y el entrenamiento del grupo de desarrollo por su importancia en las estimaciones de tiempo. Al igual que el factor de complejidad técnica se cuantifican con valores de 0 a 5 (Ver Tabla 12).La ecuación para su cálculo es:

EF=1.4-0.03*∑ (Peso i* Valor Asignado i)

TABLA 12 VALORES DEL FACTOR AMBIENTE

Factor	Descripción	Peso	Peso*Valor
E1	Familiaridad con el modelo de proyecto utilizado	1.5	1.5 * 1
E2	Experiencia en la aplicación.	0.5	0.5 * 1
E3	Experiencia en orientación a objetos.	1	1 * 4
E4	Capacidad del analista líder	0.5	0.5 * 0

E5	Motivación	1	1 * 5
E6	Estabilidad de los requerimientos.	2	2 * 3
E7	Personal a tiempo parcial.	-1	-1 * 1
E8	Dificultad del lenguaje de programación.	-1	-1 * 4
Total			12

Entonces, EF= 1.4 - 0.03 * 12 = **1.04**

Finalmente, UCP (Puntos de Caso de Uso ajustados)= UUCP * TCP * EF= 43 * 1.03 * 1.04 = **46.0616**

4.2.2. Estimación del esfuerzo

El esfuerzo en horas –hombre viene dado por (Ver Tabla 13):

TABLA 13 CÁLCULO DEL ESFUERZO

E=UCP*CF	Donde, UCP: Puntos de Casos de Uso ajustado. CF: Factor de conversión.

Para calcular el factor de conversión se contabilizan cuantos factores de los que afectan el factor ambiente están por debajo del valor medio, para los factores E1 a E6 (Ver Tabla 12).

También se contabilizan cuantos factores de los que afectan el factor ambiente están por encima del valor medio, para los factores E7 y E8 (Ver Tabla 12).

Entonces, si:

(Total EF ≤2): CF=20 horas –hombre

(Total EF =2 o Total EF=4): CF=28 horas -hombre

(Total EF≥5) CF=Hacer cambios en proyecto ya que el riesgo de fracaso es alto.

Para el caso del módulo hay un total de **2** que cumplen con las condiciones planteadas por lo que el Factor de conversión es **20 horas-hombre**.

Entonces, E=UCP*CF = 46.0616 * 20 = 921,232 **≈ 921 horas – hombre**.

Por lo que el esfuerzo en el desarrollo de las funcionalidades de los casos de uso es **921 horas – hombre**.

Finalmente, para llevar a cabo una estimación más completa de la duración total del desarrollo de la aplicación, se agrega el esfuerzo de las demás actividades. Para ello se plantea la distribución del esfuerzo entre las diferentes actividades del proyecto, según la siguiente aproximación (Ver Tabla 14):

TABLA 14 ESFUERZO DEL PROYECTO

Actividad	**Porcentaje**	**Horas-Hombre**
Análisis	10%	92,1
Diseño	20%	184,2
Programación	40%	921
Pruebas	15%	138,15
Sobrecargas(Otras actividades)	15%	138,15
Total		1473,6 **≈ 1474**

Conociendo además, que un mes tiene como promedio 30 días, que se trabaja en el proyecto 8 horas diarias aproximadamente y que la cantidad de horas que trabaja una persona en 1 mes es de 240 horas entonces:

Si el Esfuerzo total (ET) es de 1474 horas hombre. Se estima a partir de los datos conocidos que una persona puede realizar el trabajo en aproximadamente en 6 meses.

CH (Cantidad de hombres):1-------Salario promedio mensual: $100.

CHM (Costo por hombre/mes) = CH * Salario Promedio = 1 * 100 = $100/ mes.

Costo = CHM * ET (meses) /CH = 100 * 6 = $600

Tiempo = ET (meses)/ CH = 6 / 1 = 6 meses.

A partir de los resultados obtenidos se puede demostrar que con un hombre trabajando en el desarrollo del editor, este se puede desarrollar en aproximadamente 6 meses para un costo total asociado de $600.

4.3. Pruebas de caja negra

La prueba de caja negra es un método de prueba de software en el que no se requiere que el *tester (*desarrolladores que crean la implementación de las pruebas*)* conozca y comprenda el código o la estructura interna del software en prueba. Ignorara la estructura de control, concentrándose en los requisitos funcionales del sistema y comprobándolos. Se considera que estas pruebas permiten encontrar funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos y errores de rendimiento (Martin, 2016).

Las ventajas de este tipo de pruebas incluyen:

- El *tester* no necesita conocimiento de ningún lenguaje de programación específico.
- La prueba se realiza desde el punto de vista del usuario, no del diseñador.
- Los casos de prueba se pueden diseñar tan pronto como las especificaciones estén completas.

A continuación se muestran la interfaz gráfica para la realización de la operación de inserción de un proyecto (Ver Figura 33), las condiciones de entrada el caso de prueba para al caso de uso gestionar proyecto: sección insertar proyecto (Ver Tabla 15) y el caso de prueba (Ver Tabla 16).

FIGURA 33 INTERFAZ GRÁFICA PARA INTRODUCIR LOS DATOS

TABLA 15 CONDICIONES DE ENTRADA

Condición de entrada	**Tipo**	**Clases válidas**	**Clases inválidas**
Proyecto	Valor Específico	1. Cadena de Texto.	2. En blanco.
Descripción	Valor Específico	3. Cadena de Texto.	-
Permiso	Valor Específico	-	-
Usuarios Permitidos	Valor Específico	4. Cadena de Texto.	5. Cadena de Texto que no coincide con los nombres del autocompletar.

TABLA 16 CASOS DE PRUEBA PARA GESTIONAR PROYECTO: SECCIÓN INSERTAR PROYECTO

Condición de Entrada	**Casos de Prueba**	**Clases**	**Resultado**
Proyecto	UEB Prefabricado V.C	1, 3, 4	Se abre una ventana de notificación informando: Proyecto insertado correctamente.
Descripción	Proyecto relacionado con la ventas de lozas prefabricadas		
Permiso	Público		
Usuarios	Juan@2019,pepeRK		

Permitidos			
Proyecto	UEB Prefabricado V.C	2, 3, 4	En la propia ventana de inserción un *label* informa que se debe completar el nombre del proyecto.
Descripción	Proyecto relacionado con la ventas de lozas prefabricadas		
Permiso	Privado		
Usuarios Permitidos	-		
Proyecto	UEB Prefabricado V.C	1, 3, 5	Se abre una ventana de notificación informando que el proyecto no se insertó debido a que se encontraban usuarios no válidos.
Descripción	Proyecto relacionado con la ventas de lozas prefabricadas		
Permiso	Público		
Usuarios Permitidos	Lppkoplkijmm+-(		

4.4. Pruebas unitarias y dobles.

Las pruebas unitarias son un tipo de prueba de caja blanca que realizan los desarrolladores de software. Usando estas técnicas, los *testers* (generalmente desarrolladores que crean la implementación de las pruebas) verifican que el código hace lo que se pretende que haga a un nivel estructural muy bajo (Martin, 2016). Las pruebas unitarias son a menudo candidatas ideales para la automatización. Hay muchos marcos de prueba unitarias útiles disponibles que pueden hacer que sea más efectiva, como JUnit (Appel, 2015). Adicionalmente a la realización de pruebas unitarias, se emplean las pruebas dobles, ya que permite la ejecución de estas sin

tener que crear nuevas clases que aumenten la complejidad del código de prueba. Ejemplo de ello es cuando un fragmento de código depende del acceso a la base de datos, no es posible realizar una prueba unitaria del código a menos que la base de datos sea accesible, entonces se hace necesario simular dicha dependencia (Acharya, 2014).

Librerías para pruebas unitarias y dobles (Acharya, 2014):

- JUnit: librería *OpenSource* para java. Se le considera una de las herramientas más usadas para la realización de pruebas unitarias. JUnit es un marco flexible basado en anotaciones. Se integra con otros marcos de trabajo como Mockito.
- Mockito: librería *OpenSource* para java que ofrece herramientas para simular pruebas dobles. Permite indicar directamente lo que deben hacer los objetos. Su objetivo es simular una dependencia en una prueba unitaria sin tener que crear clases nuevas.

Para la realización de las pruebas unitarias y dobles en la aplicación desarrollada se utilizaron las librerías JUnit versión 4.12 y Mockito versión 1.9.5. Se crearon las clases y el código pertinente para la automatización y realización de las pruebas.

En las Figuras 34, 36 y 38 se muestran las clases creadas para la ejecución de las pruebas anteriormente descritas. Se ejecutaron las pruebas sobre los métodos actualizarModelo() de la clase VersionNgc, insertarUsuario() de la clase UsuarioNgc y actualizarProyecto() de la clase ProyectoNgc, los cuales intervienen en la capa Negocio.

Para la realización de las pruebas unitarias se utilizó métodos de la herramienta Mockito para simplificar el código y simular dependencias tales como el acceso a la capa de datos. Todas las pruebas realizadas a los componentes fueron satisfactorias (Ver Figuras 35, 37 y 39).

```
@BeforeClass
public static void inicializar() {
    Usuario usuario = new Usuario( id: 1L,  nombreUsuario: "usuarioTest",  contrasenna: "contrasenna",  correo: "usuarioTest@test.cu",  rol: "USER");
    Version version = new Version();
    version.setId(2L);
    versionResp = new VersionResp();
    versionResp.setVersion(version);
    versionNgc = Mockito.mock(VersionNgc.class);
    usuarioNgc = Mockito.mock(UsuarioNgc.class);
    usuarioAccesoDatos = Mockito.mock(UsuarioAccesoDatos.class);
    versionAccesoDatos = Mockito.mock(VersionAccesoDatos.class);

    Mockito.when(usuarioAccesoDatos.obtenerUsuario(Mockito.anyLong())).thenReturn(usuario);
    Mockito.when(versionNgc.getVersion(Mockito.anyLong(), Mockito.anyLong(), Mockito.anyObject())).thenReturn(versionResp);
    Mockito.when(versionAccesoDatos.actualizarModelo(Mockito.anyLong(), Mockito.anyString(), Mockito.anyString())).thenReturn(2L);
}
@Test
public void actualizarModeloTest() {
    Usuario userModifico = usuarioAccesoDatos.obtenerUsuario( id: 1L);
    // Datos de la versión
    versionAccesoDatos.actualizarModelo( id: 2L,  modelo: "{}", userModifico.getNombreUsuario());
    Assert.assertEquals(versionNgc.getVersion( idVersion: 2L,  idUsuario: 1L, versionResp).getVersion().getId(), new Long( value: 2));
}
```

FIGURA 34 FRAGMENTO DEL CÓDIGO DE LA CLASE DISEÑADA PARA LA REALIZACIÓN DE LAS PRUEBAS UNITARIAS Y DOBLES PARA LA OPERACIÓN ACTUALIZAR MODELO

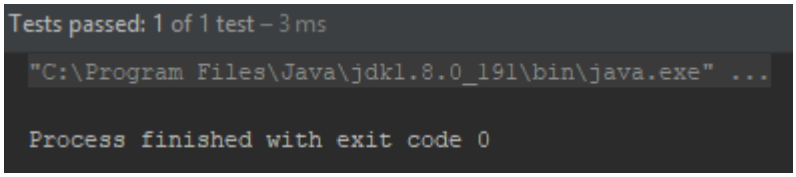


FIGURA 35 RESULTADO SATISFACTORIO PARA LA OPERACIÓN ACTUALIZAR MODELO

```
@BeforeClass
public static void inicializar() {
    Usuario usuario = new Usuario( id: 1L, nombreUsuario: "usuarioTest", contrasenna: "contrasenna", correo: "usuarioTest@test.cu", rol: "USER");
    usuarioNgc = Mockito.mock(UsuarioNgc.class);
    usuarioAccesoDatos=Mockito.mock(UsuarioAccesoDatos.class);

    Mockito.when(usuarioNgc.getUsuario(Mockito.anyString())).thenReturn(usuario);
    Mockito.doNothing().when(usuarioAccesoDatos).insertarUsuario(Mockito.anyObject(),Mockito.anyString(),Mockito.anyString(),Mockito.anyString());
}

@Test
public void insertarUsuarioTest() {
    Usuario u = new Usuario() ;
    usuarioAccesoDatos.insertarUsuario(u.getNombreUsuario(), u.getContrasenna(), u.getCorreo(), u.getRol());
    Assert.assertEquals(usuarioNgc.getUsuario(u.getNombreUsuario()).getNombreUsuario(), actual: "usuarioTest");
}
```

FIGURA 36 FRAGMENTO DEL CÓDIGO DE LA CLASE DISEÑADA PARA LA REALIZACIÓN DE LAS PRUEBAS UNITARIAS Y DOBLES PARA LA OPERACIÓN INSERTAR USUARIO

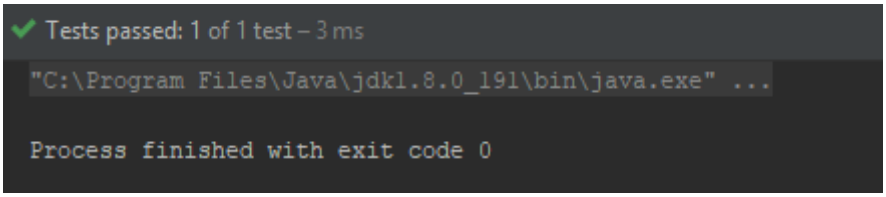


FIGURA 37 RESULTADO SATISFACTORIO PARA LA OPERACIÓN INSERTAR USUARIO

```java
@BeforeClass
public static void inicializar() {

    Usuario usuario = new Usuario( id: 1L, nombreUsuario: "usuarioTest", contrasenna: "contrasenna", correo: "usuarioTest@test.cu", rol: "USER");
    Proyecto proyecto = new Proyecto( id: 2L, nombre: "Proyecto Test", descripcion: "Proyecto de prueba", usuario, permiso: 1);

    usuarioNgc = Mockito.mock(UsuarioNgc.class);
    proyectoNgc = Mockito.mock(ProyectoNgc.class);
    proyectoAccesoDatos = Mockito.mock(ProyectoAccesoDatos.class);

    Mockito.when(usuarioNgc.getUsuario(Mockito.anyString())).thenReturn(usuario);
    Mockito.when(proyectoNgc.getProyectos(Mockito.anyLong())).thenReturn(Arrays.asList(proyecto));
    Mockito.doNothing().when(proyectoAccesoDatos).eliminarProyectosUsuarios(Mockito.anyLong());
    Mockito.doNothing().when(proyectoNgc).actualizarProyectoUsuario(Mockito.anyList(), Mockito.anyList(), Mockito.anyObject(), Mockito.anyLong());
    Mockito.doNothing().when(proyectoAccesoDatos).actualizarProyecto(Mockito.anyLong(), Mockito.anyString(), Mockito.anyString(), Mockito.anyInt());
}

@Test
public void actualizarProyectoTest() {
    List<Proyecto> proyectos = proyectoNgc.getProyectos( idUsuario: 1L);
    Proyecto p = proyectos.get(0);
    Usuario u = usuarioNgc.getUsuario( nombreUsuario: "usuarioTest");
    proyectoNgc.actualizarProyecto(p.getId(), p, u.getId());
    Assert.assertEquals(proyectoNgc.getProyectos( idUsuario: 1L).size(), actual: 1);
}
```

FIGURA 38 FRAGMENTO DEL CÓDIGO DE LA CLASE DISEÑADA PARA LA REALIZACIÓN DE LAS PRUEBAS UNITARIAS Y DOBLES PARA LA OPERACIÓN ACTUALIZAR PROYECTO

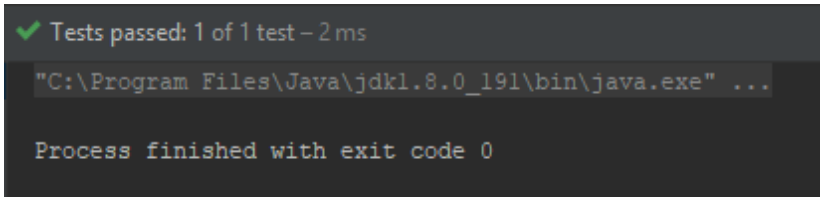


FIGURA 39 RESULTADO SATISFACTORIO PARA LA OPERACIÓN ACTUALIZAR PROYECTO

4.5. Pruebas de rendimiento.

Las Pruebas de Rendimiento se ejecutan tanto para determinar cómo responde un sistema ante una cierta carga, como para validar otros atributos relacionados con la calidad, como pueden ser la escalabilidad, la fiabilidad o el uso de recursos (Martin, 2016).

Existen distintos tipos de pruebas de rendimiento como son las de carga, stress y resistencia, la herramienta POSTMAN facilita la ejecución de dichas pruebas. Se creó un escenario con dicha herramienta para la ejecución de las mismas. Se puede ver un ejemplo de ejecución y configuración de un escenario de prueba en la Figura 58 en los Anexos para el caso de uso gestionar usuarios: sección autenticar usuario, donde se configura el tipo de petición, la petición HTTP y los parámetros de la petición. En la Figura 59 en los Anexos se muestra la respuesta dada por el servidor, parámetros enviados por este y tiempo de repuesta. Para la realización de las pruebas se creó un *script* con 2 verificaciones para comprobar que la respuesta del servidor

fuese la correcta y el tiempo de respuesta inferior a 200 ms, pues se consideró el mínimo tiempo de repuesta óptimo para el sistema (Ver Figura 40).

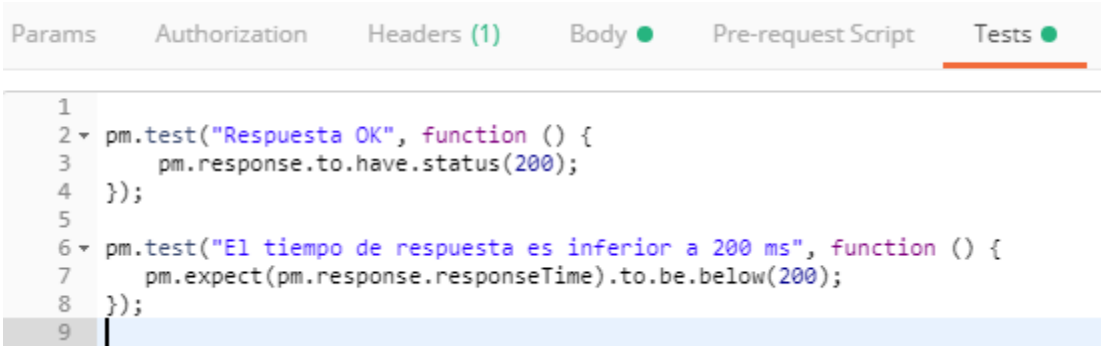


FIGURA 40 *SCRIPT* PARA LA EJECUCIÓN DE LAS PRUEBAS DE RENDIMIENTO

El escenario de pruebas se ejecutó en una PC de escritorio con procesador Intel(R) Core(TM) i3-4170 CPU @3.70GHz teniendo una memoria RAM de 8Gb y sistema operativo Windows 10 de 64bits. A continuación se muestran los resultados de tres pruebas realizadas al sistema con la herramienta POSTMAN la cual facilita este tipo de pruebas en su versión 6.7.4.

4.5.1. Pruebas de Carga

El objetivo de una prueba de carga es evaluar el comportamiento del sistema bajo una cantidad de peticiones determinadas por segundo. Este número de peticiones, a nivel básico, puede ser igual que el número de usuarios concurrentes esperados en la aplicación, más unos márgenes (Erinle, 2015). Los datos a recopilar son principalmente dos: tiempo de respuesta de la aplicación y respuestas erróneas. La prueba de Carga se configuró con 20 iteraciones con 6 peticiones por usuario, con un tiempo de 200 milisegundos de diferencia entre las peticiones. Además el *script* cuenta con 2 verificaciones:

20/iteraciones*6/peticiones*2/verificaciones = 240/verificaciones

De las 240 verificaciones todas las peticiones se ejecutaron satisfactoriamente solo 6 no superaron el tiempo propuesto de respuesta de 200ms (Ver Figura 41).

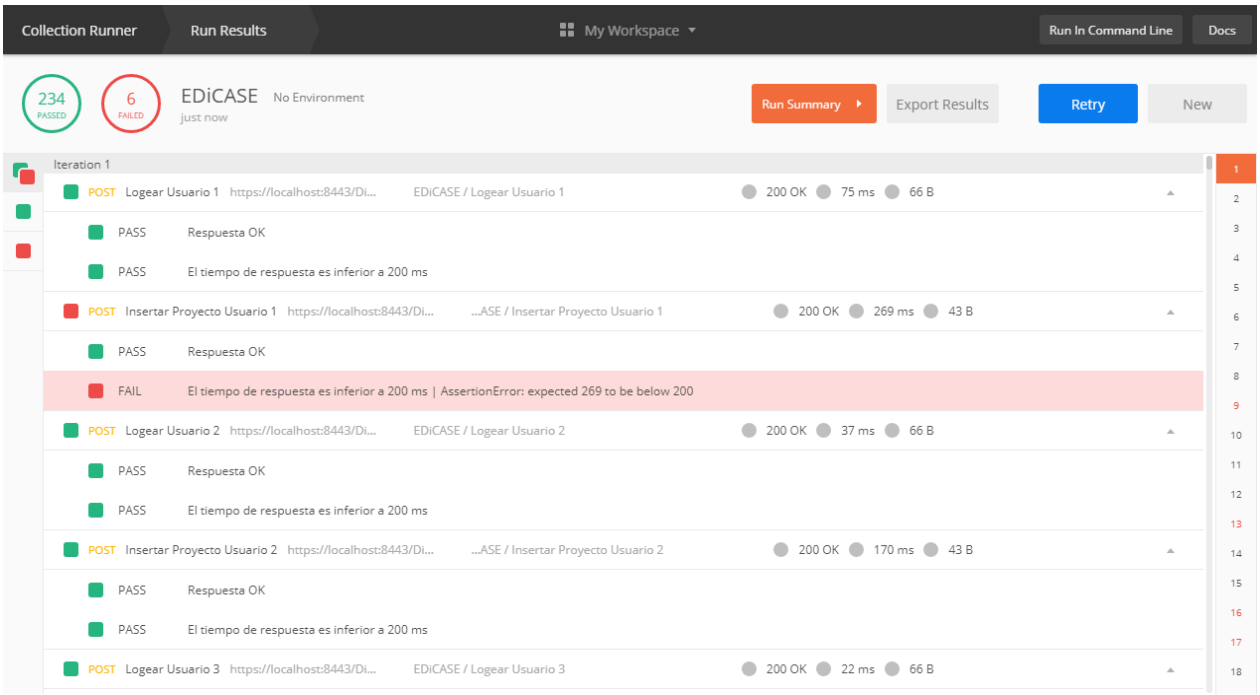


FIGURA 41 RESULTADO DE LAS PRUEBAS DE CARGA

4.5.2. Pruebas de Stress

Las pruebas de stress, a diferencia de las pruebas de carga, tienen por objetivo determinar el punto de ruptura del servicio y analizar sus causas que suelen estar derivadas de problemas que suceden ante condiciones muy elevadas de carga: mala escalabilidad, agotamiento de capacidad y fugas de memoria (Erinle, 2015). La prueba de Stress se configuró con 60 iteraciones con 3 peticiones por usuario, con un tiempo de 400 milisegundos de diferencia entre las peticiones. Además el *script* cuenta con 2 verificaciones:

60/iteraciones*3/peticiones*2/verificaciones = 360/verificaciones

De las 480 verificaciones todas las peticiones se ejecutaron satisfactoriamente solo 5 no superaron el tiempo propuesto de respuesta de 200ms (ver Figura 42).

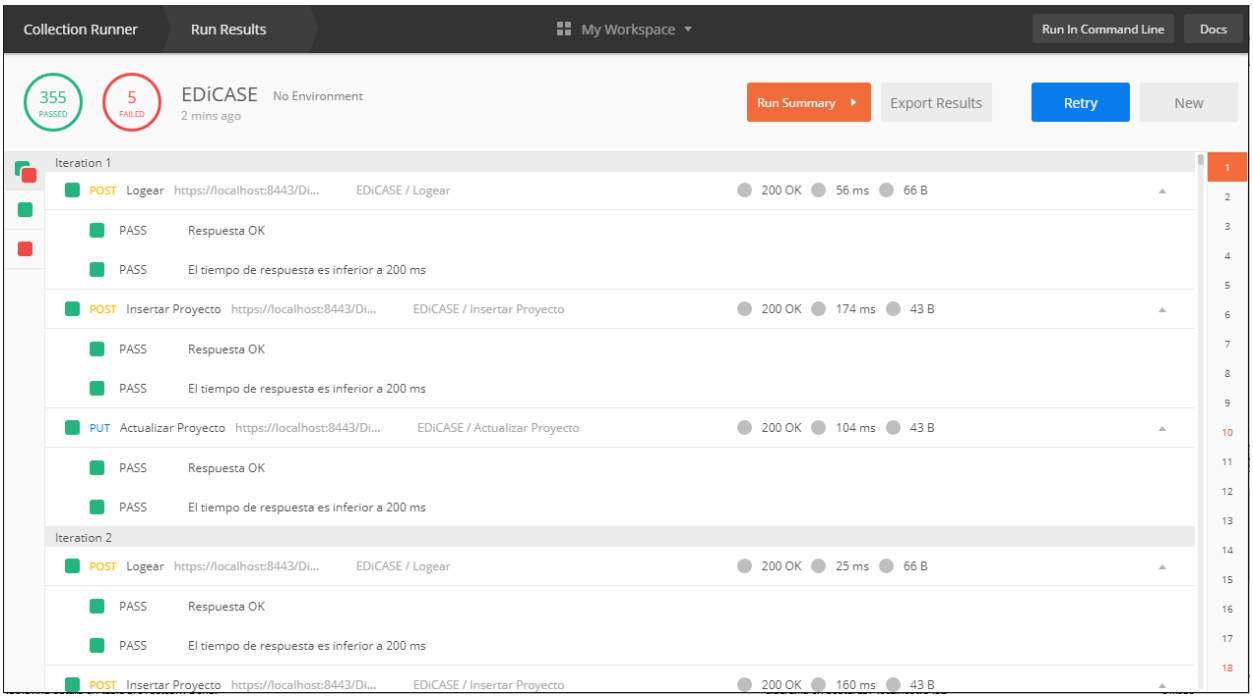


FIGURA 42 RESULTADO DE LAS PRUEBAS DE STRESS

4.5.3. Pruebas de Resistencia

Las pruebas de resistencia o SOAK, también llamadas de estabilidad, son una variante de las pruebas de carga. Su objetivo es determinar el comportamiento de la aplicación a lo largo del tiempo y evaluar la degradación del rendimiento derivada del uso sostenido. En las aplicaciones Web han adquirido una importancia significativa, ya que la disponibilidad de estas es fundamental (Martin, 2016).

La prueba de Resistencia se configuró con 80 iteraciones con 3 peticiones por usuario, con un tiempo de 200 milisegundos de diferencia entre las peticiones. Además el *script* cuenta con 2 verificaciones:

80/iteraciones*3/peticiones*2/verificaciones = 480/verificaciones

De las 480 verificaciones todas las peticiones se ejecutaron satisfactoriamente solo 2 no superaron el tiempo propuesto de respuesta de 200ms (ver Figura 43).

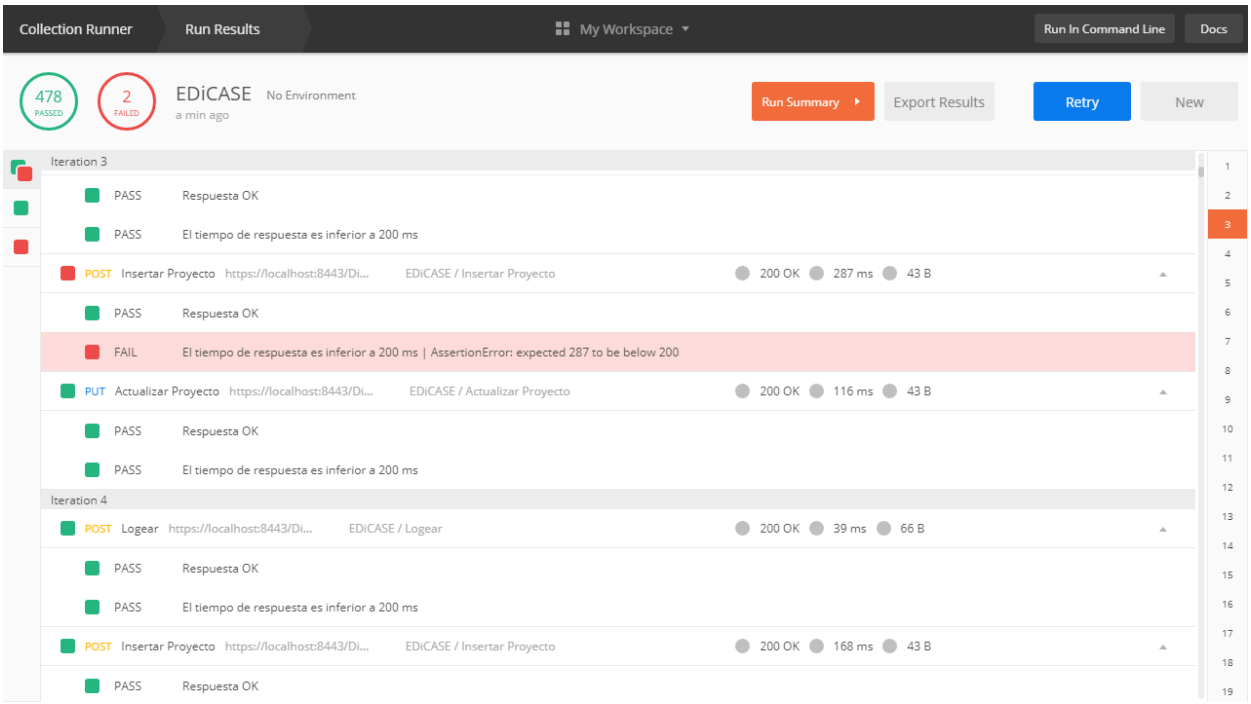


FIGURA 43 RESULTADO DE LAS PRUEBAS DE RESISTENCIA

4.6. Conclusiones

La estimación, el cálculo de tamaño del software, el esfuerzo y los costos fueron algunos de los elementos presentados en este capítulo correspondiente al estudio de la factibilidad del sistema. Finalmente después de analizar el costo del proyecto, es posible concluir que su elaboración es totalmente factible.

También se efectuaron las pruebas pertinentes al software como parte de metodología empleada para el desarrollo del mismo, la cual hace énfasis en este aspecto, realizándose las pruebas de caja negra, automatizándose las pruebas unitarias y dobles y la realización de pruebas de carga, stress y resistencia lográndose resultados satisfactorios en la ejecución de las mismas.

Conclusiones

Con el desarrollo de EDiCASE se creó una herramienta Web que permite modelar diagramas Entidad Relación Extendido. La puesta en marcha de este editor no solo permitirá llevar a cabo el modelado de diagramas, sino que posibilitará la colaboración entre usuarios y manejará el versionado de los proyectos creados por los usuarios.

Durante el progreso de la investigación se obtuvieron los siguientes resultados:

1. Se identificó la plataforma Web apropiada, tecnologías y herramientas que fueron aplicadas para la creación de esquemas conceptuales y el desarrollo de manera general de la aplicación.
2. Se determinó que la notación Elmari&Navathe es la más apropiada para la construcción de esquemas conceptuales por poseer gran expresividad para representar los diferentes tipos de atributos, vínculos y restricciones del modelo Entidad Relación Extendido.
3. Se implementó un editor de esquemas conceptuales que permite modelar varias construcciones y restricciones existentes en la notación seleccionada.

Recomendaciones

Con el objetivo de brindar una aplicación que incluya las funcionalidades necesarias para modelar los diagramas entidad relación extendido, se recomienda:

- Ampliar el plugin desarrollado con jQuery para que permita representar las relaciones de categorización, de tipo ISA y la agregación.
- Analizar el XML que es exportado para establecer una estructura que permita que pueda ser cargado por otras herramientas de la misma índole, permitiendo la colaboración entre herramientas.
- Incorporar la funcionalidad de importar un XML y que a partir de este documento se dibujen los elementos del diagrama.
- Incorporar la funcionalidad de importar a un formato de imagen los elementos del diagrama.
- Investigar de nuevas tecnologías para garantizar mejoras en futuras versiones del sistema.

Referencias bibliográficas

Acharya, S., 2014. Mastering Unit Testing Using Mockito and JUnit. Packt Publishing.

Álvarez Martínez de la Cotera, W.A., 2006. ERECASE v.2.0 Una herramienta para el diseño conceptual de bases de datos con validación estructural. UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS, Cuba.

Ambler, S.W., 2002. Agile Modeling - Effective Practices for eXtreme Programming & the Unified Process. John Wiley & Sons.

André, M., Baldoquín, M.G., 2008. La formación de equipos de proyectos de software. V Simp. Ing. Ind. Afines.

Appel, F., 2015. Testing with Junit. Packt Publishing - ebooks Account.

Bae, S., 2019. JavaScript Data Structures and Algorithms: An Introduction to Understanding and Implementing Core Data Structure and Algorithm Fundamentals, 1st ed. edition. ed. Apress, New York, NY.

Batini, C., Ceri, S., Navathe, S.B., 1992. Conceptual Database Design: An Entity-Relationship Approach. Addison-Wesley, San Francisco, Ca, U.s.a.

Burd, B., 2017. Java For Dummies, 7th Edition. ed. John Wiley & Sons Inc, Hoboken, New Jersey.

Chavarría Báez, L., Ocotilla Rojas, N., 2016. Sobre el uso de herramientas CASE para la enseñanza de bases de datos. Mem. Sexta Conf. Iberoam. Complejidad Informática Cibernética.

Chen, P.P.-S., 1976. The Entity Relationship Model – Toward a Unified View of Data ACM Transactions on Database Systems Vol. 1 (1), 1976.

DIAS, G.K.A., 2017. Evolvement of Computer Aided Software Engineering (CASE) Tools: A User Experience. Int. J. Comput. Sci. Softw. Eng. IJCSSE Issue 3 Volume 6, 55–60.

Elmasri, R., Navathe, Shamkant B., 2007. Fundamentos de Sistemas de Base de Datos.

Erinle, B., 2015. Performance Testing with Jmeter - Second Edition, Edición: 2. ed. Packt Publishing - ebooks Account, Birmingham, England.

Evans, M., Song, I.-Y., Park, E.K., 1995. A Comparative Analysis of Entity-Relationship Diagrams. J. Comput. Softw. Eng. No4 Volumen 3.

Fielding, R.T., 2000. Architectural Styles and the Design of Network-based Software Architectures.

Franklin, J., Ferguson, R., 2017. Beginning jQuery: From the Basics of jQuery to Writing your Own Plug-ins, 2nd ed. edition. ed. Apress, New York, NY.

García, C., 2010. Enfoque sistémico a la modelación de datos, base para el desarrollo de una herramienta CASE. UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS, Cuba.

Goldstein, A., Lazaris, L., Weyl, E., 2015. HTML5 & CSS3 For The Real World: Powerful HTML5 and CSS3 Techniques You Can Use Today!, 2 edition. ed. SitePoint, Collingwood.

Holmberg, D., Nyberg, V., 2018. Functional and Security Test- ing of a Mobile Client-Server Application p.50.

Krochmalski, J., 2014. IntelliJ IDEA Essentials. Packt Publishing.

Lambert, M., 2016. Learning Bootstrap 4, Second Edi. ed.

Larman, C., 1999. UML y Patrones, 1st ed. Prentice-Hall.

Marrs, T., 2017. JSON at Work: Practical Data Integration for the Web, 1 edition. ed. O'Reilly Media, Sebastopol, CA.

Martin, K., 2016. The Agile Tester 2: Software testing in the agile world, 2nd ed. CreateSpace Independent Publishing Platform.

Morffi, A.R., 2007. Sistema integrado de herramientas de ayuda al diseño de bases de datos en ambientes distribuidos. UNIVERSIDAD CENTRAL “MARTA ABREU” DE LAS VILLAS, Cuba.

Patni, S., 2017. Pro Restful APIs: Design, Build and Integrate with Rest, JSON, XML and JAX-RS. Apress, Berkeley, California.

Pressman, R.S., 2010. Ingeniería de software.Un enfoque práctico, 7ma Edición. ed.

Quintero, J.B., Hernández, D.M., Yanza, A., 2008. DIRECTRICES PARA LA CONSTRUCCIÓN DE ARTEFACTOS DE PERSISTENCIA EN EL PROCESO DE DESARROLLO DE SOFTWARE. Rev. EIA 77–90.

Remón, C.A., Thomas, P., 2010. Análisis de Estimación de Esfuerzo aplicando Puntos de Caso de Uso. VII Workshop Ing. Softw. WIS 577–586.

Rumbaugh, J., Jacobson, I., Booch, G., 2007. El lenguaje unificado de modelado manual de referencia, 2nd. edition. ed.

Sandeep, K.P., 2014. Developing Responsive Web Applications with AJAX and jQuery.

Sandoval, J., 2009. RESTful Java Web Services. Packt Publishing, Birmingham.

Saternos, C., 2014. Client-Server Web Apps with JavaScript and Java: Rich, Scalable, and RESTful, 1 edition. ed. O’Reilly Media, Sebastopol, CA.

Smith, G., 2010. PostgreSQL 9.0 High Performance. Packt Publishing, Birmingham.

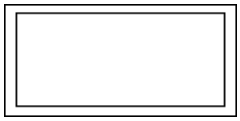
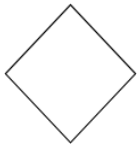
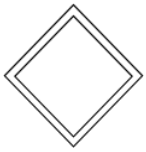
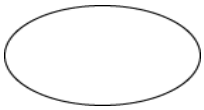
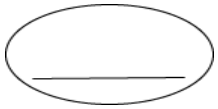
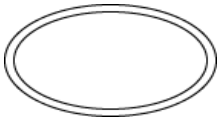
Stephens, R., 2015. Beginning Software Engineering, Edición: 1. ed. Sybex.

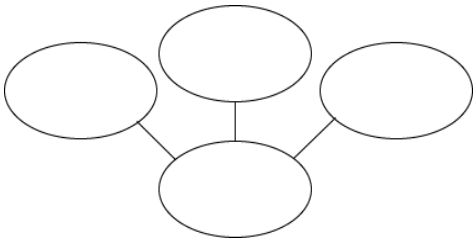
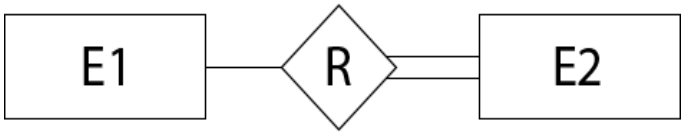
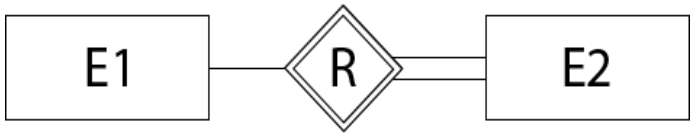
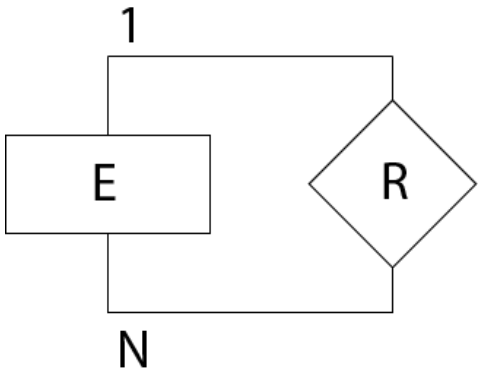
Vukotic, A., Goodwill, J., 2011. Apache Tomcat 7, 1st ed. edition. ed. Apress, New York, NY.

York, R., 2015. Web Development with jQuery, Edición: 2nd ed. ed. John Wiley & Sons Inc, Indianapolis, Indiana.

Anexos

TABLA 17 ANEXOS: CONSTRUCCIONES DE LA NOTACIÓN ESMARI & NAVATHE

Símbolo	Significado
	**Entidad.**
	**Entidad débil.**
	**Relación.**
	**Relación débil.**
	**Atributo.**
	**Atributo clave.**
	**Atributo multivalor.**
	**Atributo derivado.**

	**Atributo compuesto.**
	**Cardinalidad 1:1 para E1:E2 en R.**
	**Cardinalidad 1:N para E1:E2 en R.**
	**Cardinalidad N:M para E1:E2 en R.**
	**Participación total E2 en R.**
	**Entidad W identificada por una entidad E a través de la relación débil R.**
	**Relación recursiva R.**

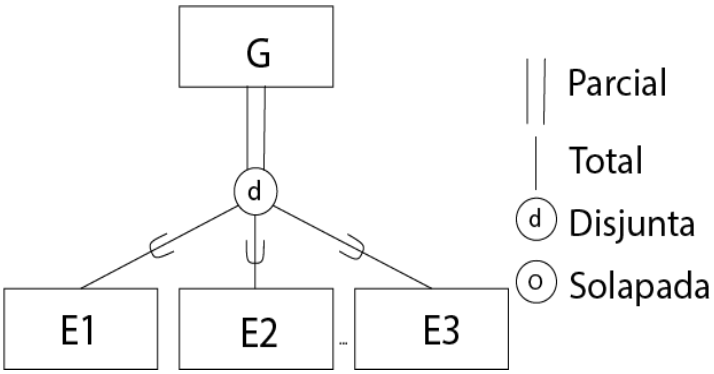
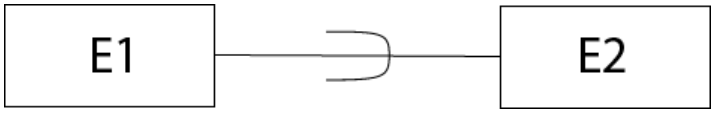
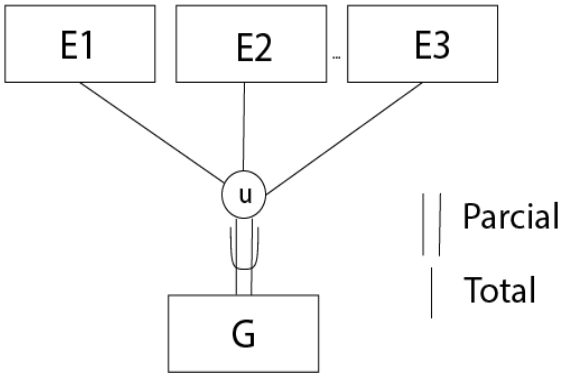
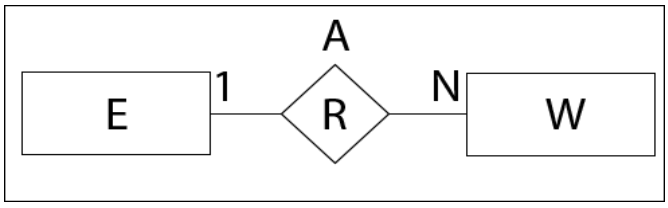
	**Jerarquía de generalización/especialización.**
	**Interrelación de subconjunto (ISA).**
	**Interrelación de categorización.**
	**Agregación.**

TABLA 18 ANEXOS: CASO DE USOS DEL SISTEMA GESTIONAR USUARIO

Caso de Uso:	Gestionar Usuario.
Actores:	Usuario.
Resumen:	El caso de uso inicia cuando un usuario accede a la página de autenticación. El usuario tiene la posibilidad de registrarse y cambiar su contraseña con los datos que el sistema especificara. Para iniciar su sesión el usuario deberá autenticarse.
Precondiciones:	Abrir el navegador y acceder a la página de autenticación.
Referencias:	RF1.1, RF1.2, RF1.3.
Prioridad:	Normal
Flujo Normal de Eventos	
Sección "Registrar cuenta**"**	
Acción de Actor	**Respuesta del Sistema**
1. Accede desde el navegador a la página inicial del sistema, y selecciona la opción Registro.	2. Muestra una interfaz con los siguientes campos: • Usuario • Correo • Contraseña • Repetir Contraseña
3. Inserta los datos especificados por el sistema. 4. Presiona el botón Registrar.	5. Muestra un mensaje al autor de que se encuentra registrado en el sistema. 6. Termina el caso de uso.

Flujos Alternos	
3. El actor no introduce uno de los campos.	
Acción de Actor	**Respuesta del Sistema**
	3.1. Muestra un mensaje de advertencia al actor reflejando que el autor no ha especificado uno de los campos. 3.2. Regresa al paso 3 del flujo de trabajo.
3. El actor introduce datos inválidos en uno de los campos.	
	3.1. Muestra un mensaje de Error al actor reflejando que ha introducido datos inválidos en uno de los campos. 3.2. Regresa al paso 3 del flujo de trabajo.
Prototipo de Interfaz	

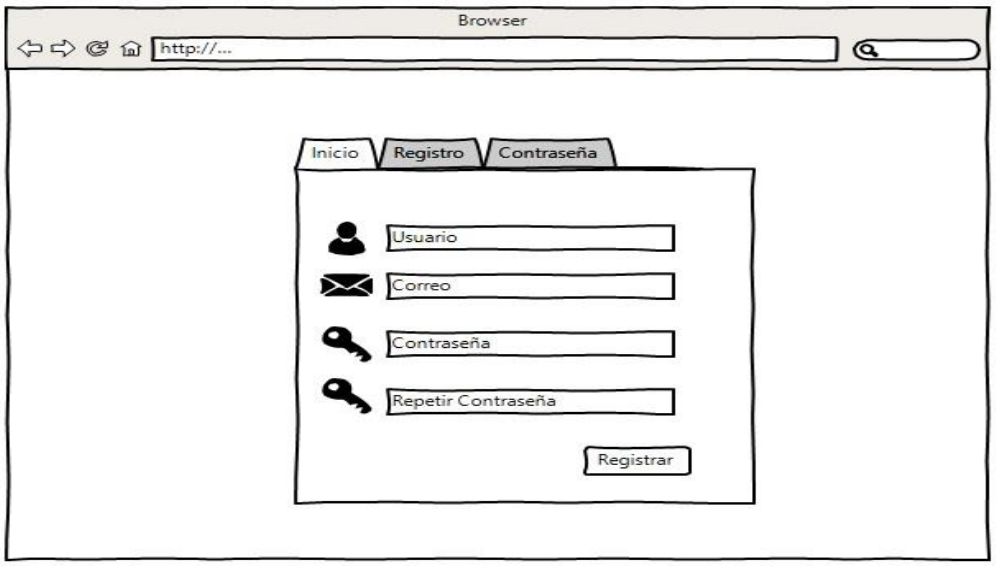


FIGURA 44 PROTOTIPO REGISTRAR CUENTA

Flujo Normal de Eventos

<table>
<tr><th colspan="2">Sección “Cambiar contraseña”</th></tr>
<tr><th>Acción de Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1. Accede desde el navegador a la página inicial del sistema, y selecciona la opción Contraseña.</td><td>2. Muestra una interfaz con los siguientes campos:
Usuario
Contraseña
Repetir Contraseña</td></tr>
<tr><td>3. Inserta los datos especificados por el sistema.
4. Presiona el botón Enviar.</td><td>5. Muestra un mensaje al autor de que ha cambiado su contraseña exitosamente.
6. Termina el caso de uso.</td></tr>
<tr><th colspan="2">Flujos Alternos</th></tr>
<tr><td colspan="2">3. El actor no introduce uno de los campos.</td></tr>
<tr><th>Acción de Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td></td><td>3.1. Muestra un mensaje de advertencia al actor reflejando que el autor no ha especificado uno de los campos.
3.2. Regresa al paso 3 del flujo de trabajo.</td></tr>
<tr><td colspan="2">3. El actor introduce datos inválidos en uno de los campos.</td></tr>
<tr><th>Acción de Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td></td><td>3.1. Muestra un mensaje de Error al actor reflejando que ha introducido datos inválidos en uno de los campos.
3.2. Regresa al paso 3 del flujo de trabajo.</td></tr>
<tr><th colspan="2">Prototipo de Interfaz</th></tr>
</table>

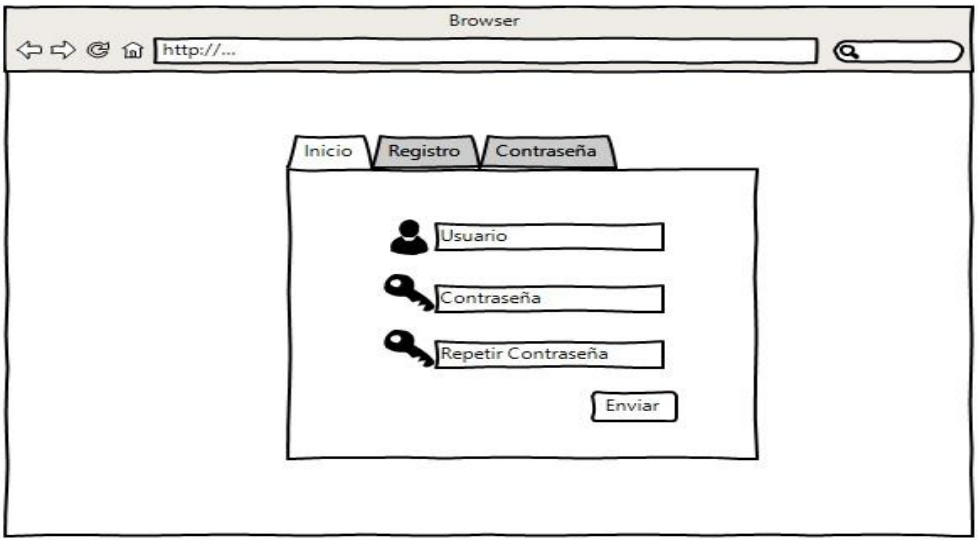


FIGURA 45 PROTOTIPO CAMBIAR CONTRASEÑA

Sección "Autenticar usuario**"**	
Flujo Normal de Eventos	
Acción de Actor	**Respuesta del Sistema**
1. Accede desde el navegador a la página inicial del sistema, y selecciona la opción Inicio.	2. Muestra una interfaz con los siguientes campos: • Usuario • Contraseña
3. Inserta los datos especificados por el sistema. 4. Presiona el botón Inicio.	5. Inicia la sesión del actor. 6. Termina el caso de uso.
Flujos Alternos	
3. El actor no introduce uno de los campos.	
Acción de Actor	**Respuesta del Sistema**
	3.1. Muestra un mensaje de advertencia al actor reflejando que el autor no ha

	especificado uno de los campos. 3.2. Regresa al paso 3 del flujo de trabajo.
5. El actor introduce datos inválidos en uno de los campos.	
Acción de Actor	**Respuesta del Sistema**
	5.1. Muestra un mensaje de Error al actor reflejando que ha introducido datos inválidos en uno de los campos. 5.2. Regresa al paso 3 del flujo de trabajo.
Prototipo de Interfaz	

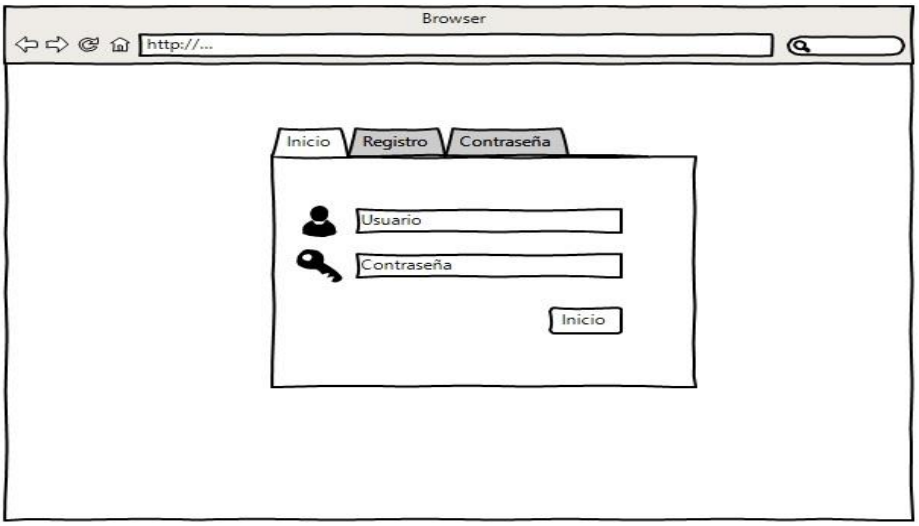


FIGURA 46 PROTOTIPO AUTENTICAR USUARIO

TABLA 19 ANEXOS: CASO DE USOS DEL SISTEMA EXPORTAR MODELO.

Caso de Uso:	Exportar Modelo.
Actores:	Usuario.
Resumen:	El caso de uso inicia cuando un usuario accede a la página de modelación de diagramas. El usuario tiene la posibilidad de crear exportar su modelo a un documento XML.
Precondiciones:	Acceder a la página de modelación de diagramas.
Referencias:	RF4.
Prioridad:	Normal.
Flujo Normal de Eventos	
Acción de Actor	**Respuesta del Sistema**
7. Presiona en la barra de navegación la opción Salvar XML.	8. Muestra la ventana Salvar XML con los siguientes campos: • Nombre del documento
9. Inserta el nombre del documento. 10. Presiona el botón Aceptar.	11. Muestra un mensaje de exportación exitosa. 12. Termina el caso de uso.
Flujos Alternos	
4. El actor no introduce el nombre del documento y presiona el botón Aceptar.	
Acción de Actor	**Respuesta del Sistema**
	4.1. Muestra un mensaje de advertencia al actor reflejando que el autor no ha

	especificado el campo. 4.2. Regresa al paso 3 del flujo de trabajo.
4. El actor presiona el botón Cancelar.	
	4.1. Cierra la ventana Salvar XML. 4.2. Termina el caso de uso..
Prototipo de Interfaz	

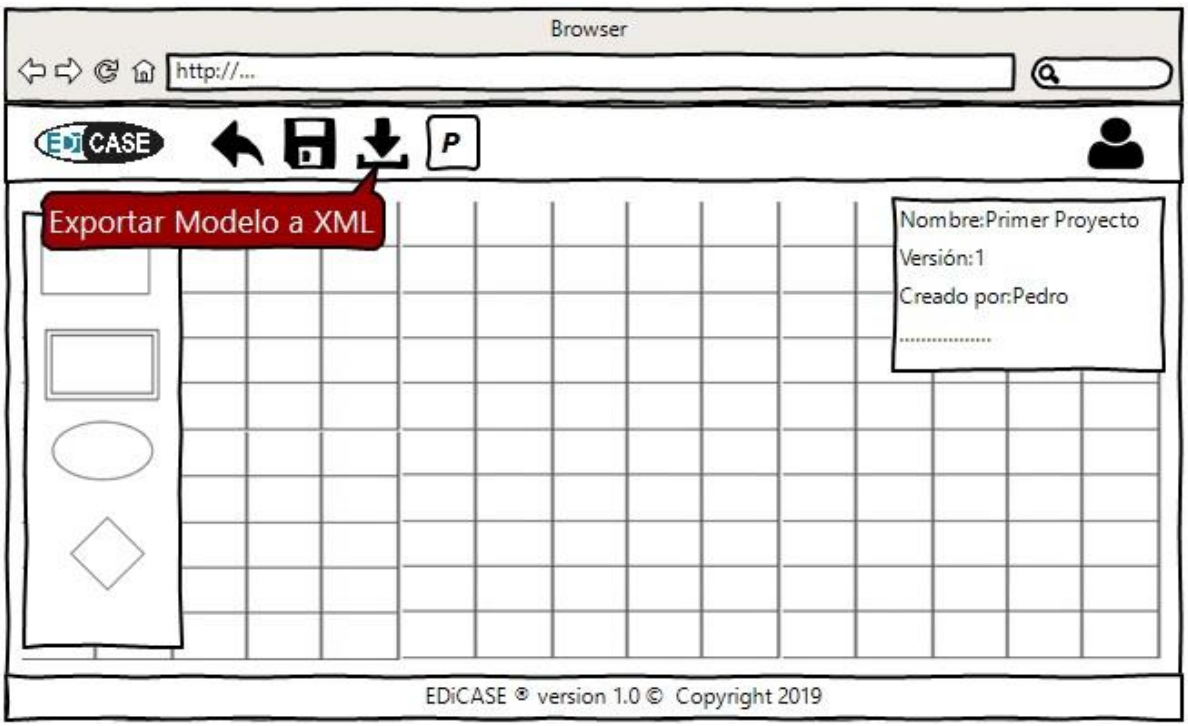


FIGURA 47 PROTOTIPO INTERFAZ EXPORTAR MODELO A XML.

FIGURA 48 PROTOTIPO EXPORTAR MODELO A XML.

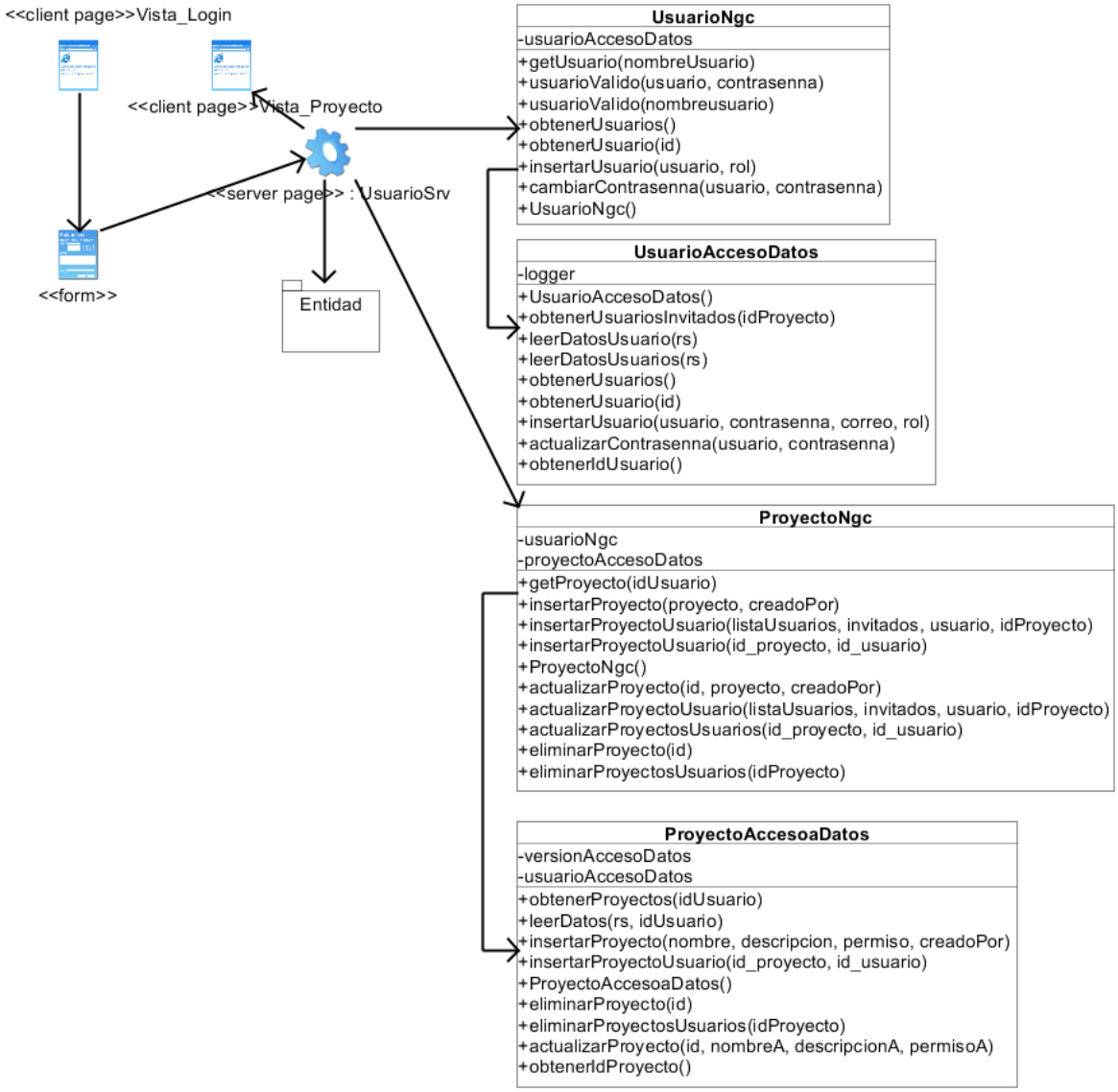


FIGURA 49 DIAGRAMA DE CLASE DE DISEÑO: AUTENTICAR USUARIO

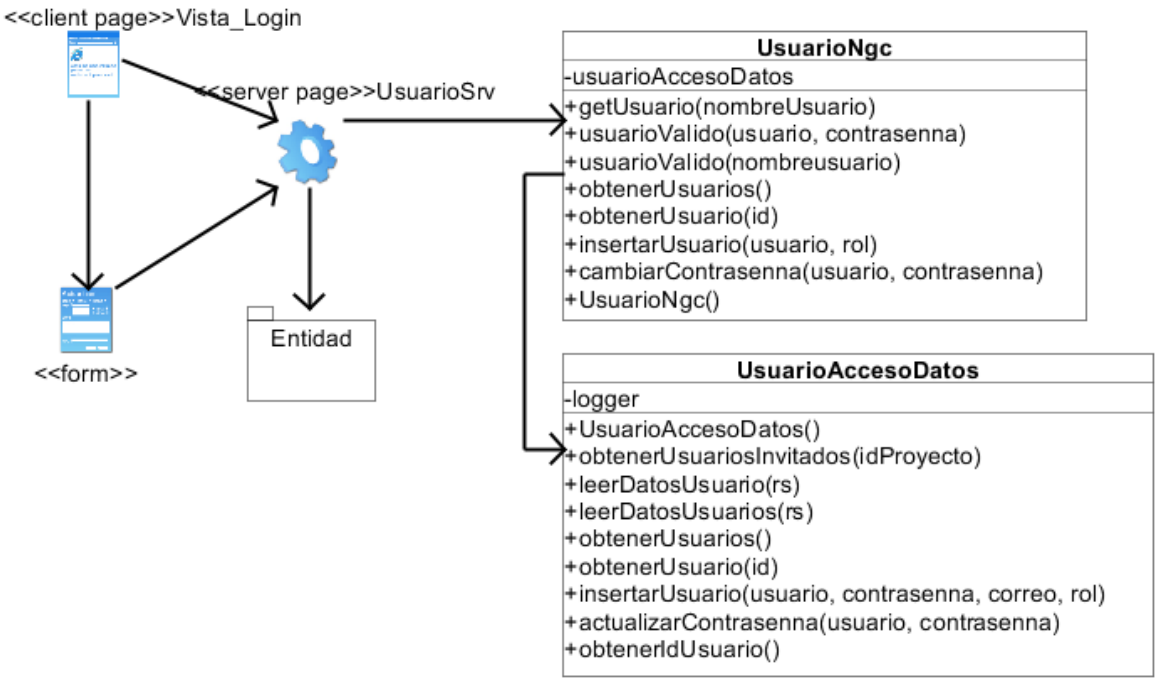


FIGURA 50 DIAGRAMA DE CLASE DE DISEÑO: REGISTRAR USUARIO

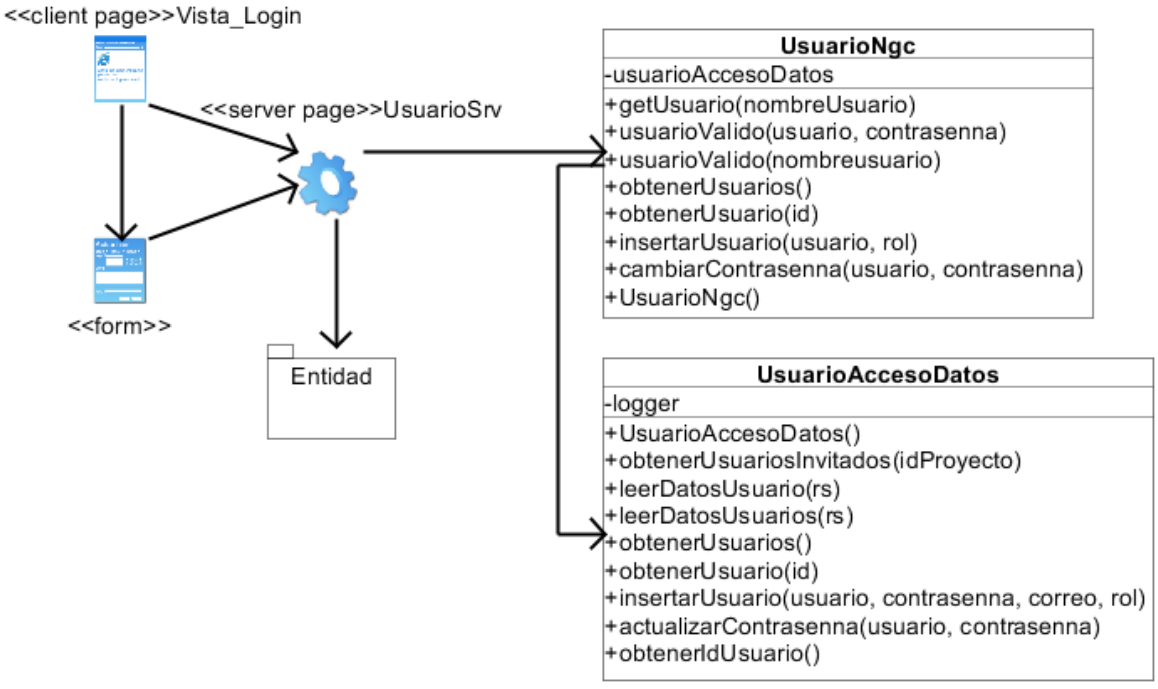


FIGURA 51 DIAGRAMA DE CLASE DE DISEÑO: CAMBIAR CONTRASEÑA

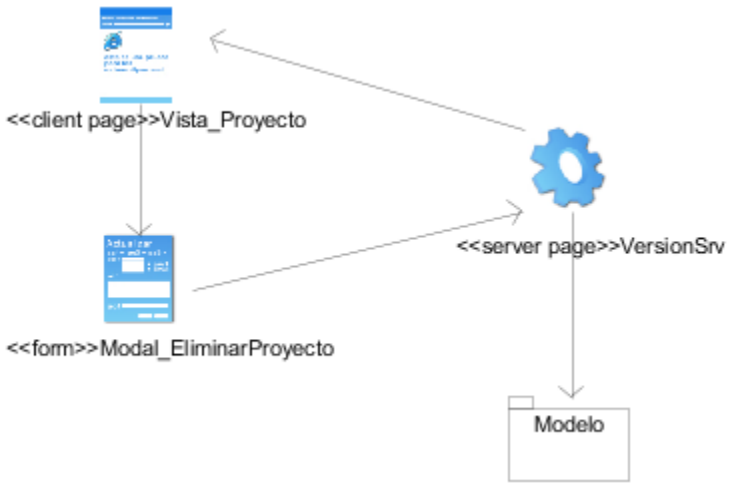


FIGURA 52 DIAGRAMA DE CLASE DE DISEÑO: EXPORTAR MODELO

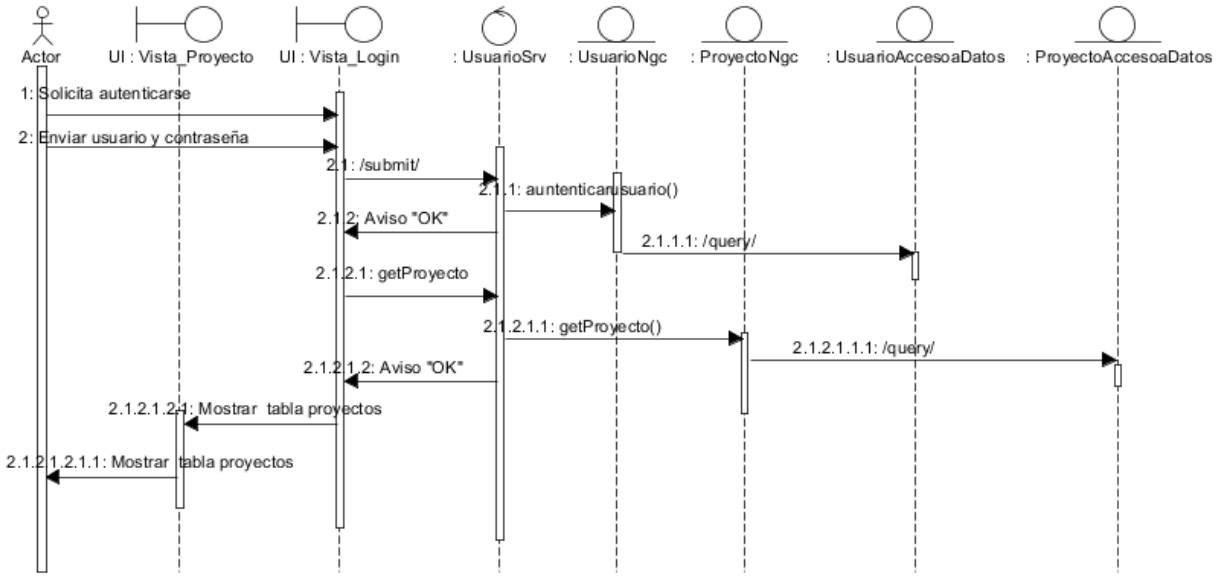


FIGURA 53 DIAGRAMA DE SECUENCIA SECCIÓN: AUTENTICAR USUARIO

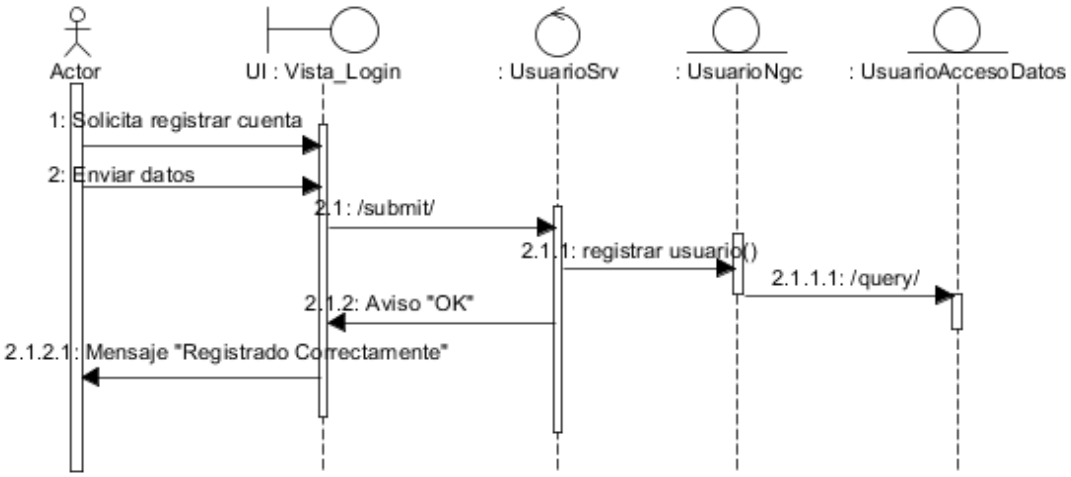


FIGURA 54 DIAGRAMA DE SECUENCIA SECCIÓN: REGISTRAR USUARIO

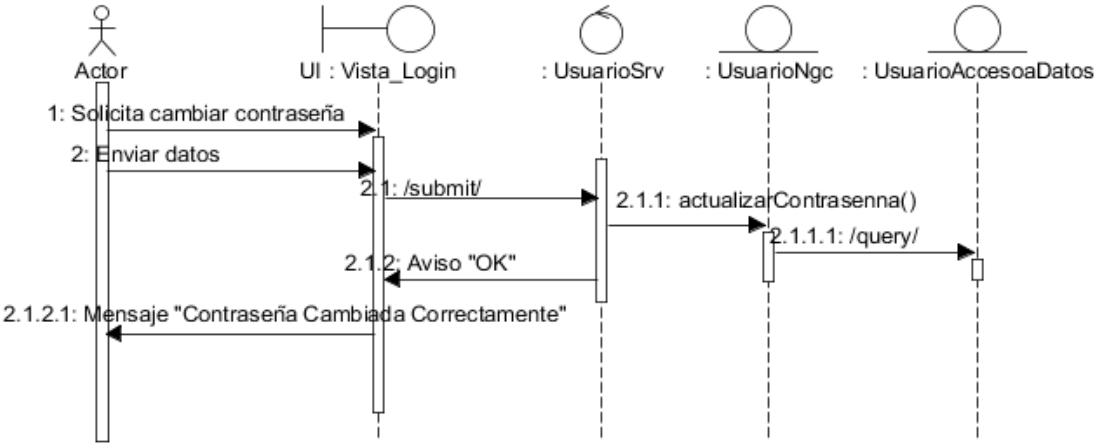


FIGURA 55 DIAGRAMA DE SECUENCIA SECCIÓN: CAMBIAR CONTRASEÑA

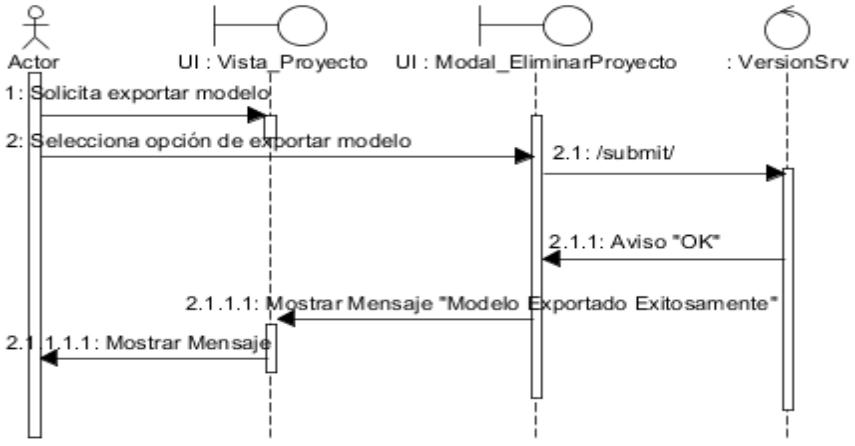


FIGURA 56 DIAGRAMA DE SECUENCIA SECCIÓN: EXPORTAR MODELO

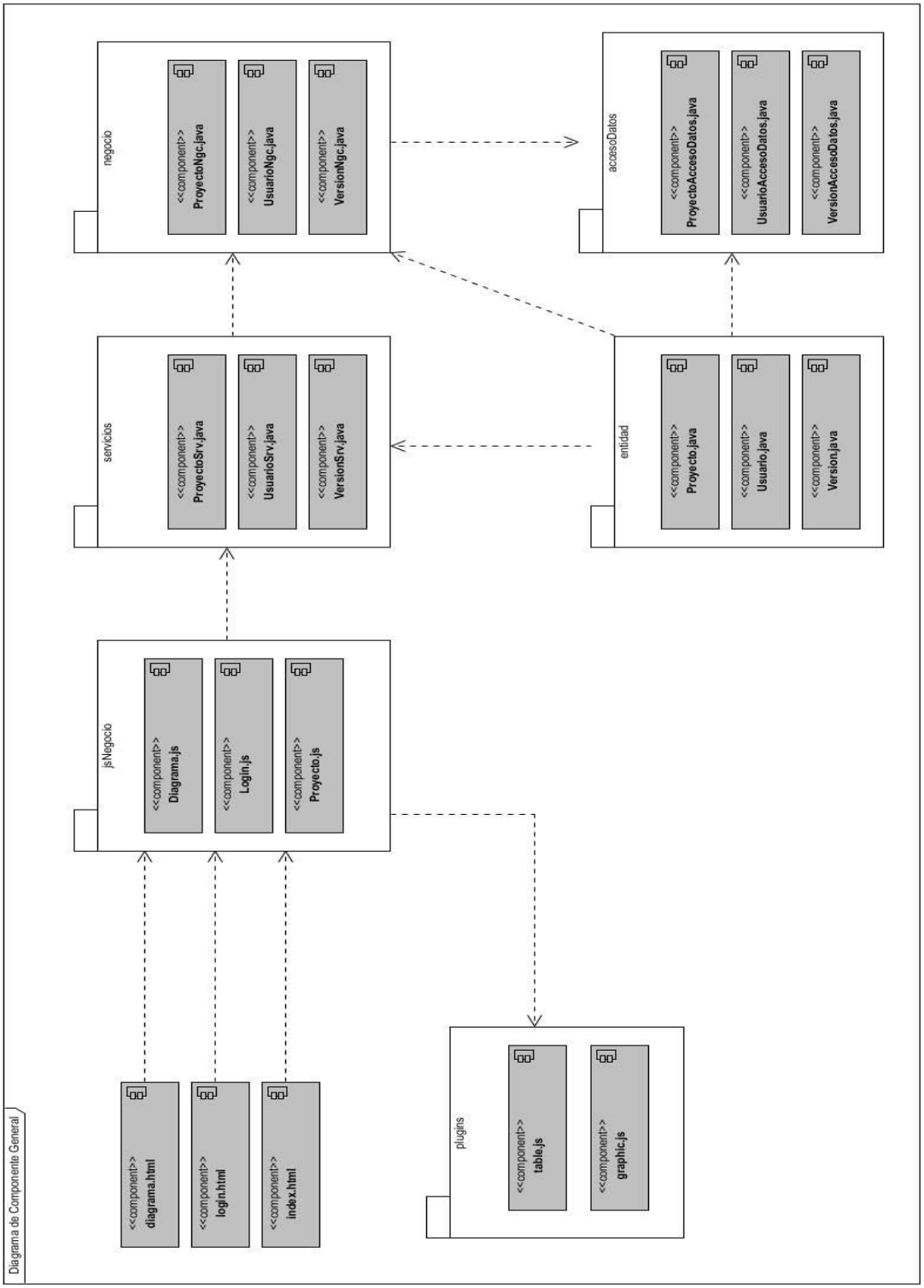


FIGURA 57 DIAGRAMA DE COMPONENTES GENERAL.

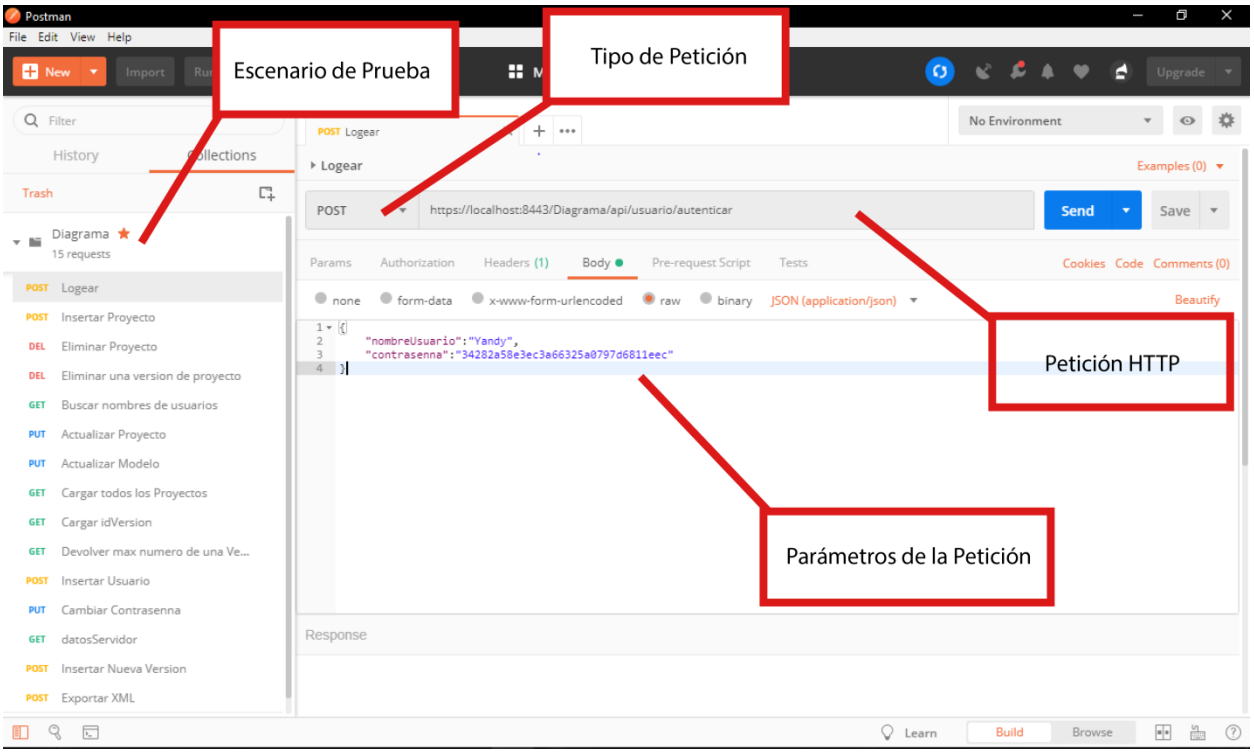


FIGURA 58 INTERFAZ DEL POSTMAN. ESCENARIO DE PRUEBA PARA AUTENTICAR USUARIO.

FIGURA 59 INTERFAZ DEL POSTMAN. RESPUESTA DEL AUTENTICAR USUARIO.