CONSTRUCCIÓN DE UNA APLICACIÓN WEB PARA LA ELABORACIÓN DE DIAGRAMAS DE CLASE, ORIENTADA BAJO EL MODELO DE SERVICIOS DE INTERNET CLOUD COMPUTING

(Proyecto de Grado Dirigido)

GABRIEL LEONARDO DÍAZ CÁRDENAS

UNIVERSIDAD FRANCISCO DE PAULA SANTANDER

FACULTAD DE INGENIERIAS

PLAN DE ESTUDIOS DE INGENIERIA DE SISTEMAS

SAN JOSÉ DE CÚCUTA

2014

CONSTRUCCIÓN DE UNA APLICACIÓN WEB PARA LA ELABORACIÓN DE DIAGRAMAS DE CLASE, ORIENTADA BAJO EL MODELO DE SERVICIOS DE INTERNET CLOUD COMPUTING.

GABRIEL LEONARDO DÍAZ CÁRDENAS

Código 0152685

Trabajo de grado para ostentar el título de Ingeniero de Sistemas de la Universidad Francisco de Paula Santander

DIRECTOR

MARCO ANTONIO ADARME JAIMES

INGENIERO DE SISTEMAS

MAGISTER EN CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD FRANCISCO DE PAULA SANTANDER

FACULTAD DE INGENIERIAS

PLAN DE ESTUDIOS DE INGENIERIA DE SISTEMAS

SAN JOSÉ DE CÚCUTA

2014

**TABLA DE CONTENIDO**

[LISTADO DE FIGURAS 7](#_Toc385624532)

[LISTADO DE DIAGRAMAS 8](#_Toc385624533)

[LISTADO DE TABLAS 9](#_Toc385624534)

[INTRODUCCIÓN 10](#_Toc385624535)

[1. PRESENTACIÓN DEL PROYECTO 12](#_Toc385624536)

[1.1 TITULO 12](#_Toc385624537)

[1.2 PLANTEAMIENTO DEL PROBLEMA 12](#_Toc385624538)

[1.3 JUSTIFICACIÓN 13](#_Toc385624539)

[1.4 OBJETIVOS 15](#_Toc385624540)

[1.4.1 Objetivo General 15](#_Toc385624541)

[1.4.2 Objetivos Específicos 15](#_Toc385624542)

[1.5 ALCANCES Y DELIMITACIONES 16](#_Toc385624543)

[1.5.1 Alcance 16](#_Toc385624544)

[1.5.2 Delimitaciones 16](#_Toc385624545)

[2. MARCO REFERENCIAL 18](#_Toc385624546)

[2.1 MARCO DE ANTECEDENTES 18](#_Toc385624547)

[2.1.1 GenMyModel 18](#_Toc385624548)

[2.1.2 Draw.io 18](#_Toc385624549)

[2.1.3 Creately 19](#_Toc385624550)

[2.1.4 GWTUML 19](#_Toc385624551)

[2.1.5 JoinJS 20](#_Toc385624552)

[2.2 MARCO CONCEPTUAL 20](#_Toc385624553)

[2.2.1 Herramientas CASE 20](#_Toc385624554)

[2.2.2 UML 21](#_Toc385624555)

[2.2.3 Diagramas de Clase 22](#_Toc385624556)

[2.2.4 Cloud Computing 23](#_Toc385624557)

[2.3 MARCO TEORICO 25](#_Toc385624558)

[2.3.1 Java 25](#_Toc385624559)

[2.3.2 Frameworks 26](#_Toc385624560)

[2.3.3 Extreme Programming (XP) 27](#_Toc385624561)

[2.3.4 HTML, JavaScript y CSS 28](#_Toc385624562)

[2.4 MARCO DE LEGAL 29](#_Toc385624563)

[2.4.1 Acuerdos de uso 29](#_Toc385624564)

[2.4.2 Contrato de licencia de código binario, SUN MICROSYSTEMS 29](#_Toc385624565)

[2.4.3 Contrato de licencia de mxGraph 30](#_Toc385624566)

[2.4.4 GNU General Public Licence 30](#_Toc385624567)

[2.4.5 Propiedad Intelectual 31](#_Toc385624568)

[3. DISEÑO METODOLÓGICO 32](#_Toc385624569)

[3.1 TIPO DE INVESTIGACIÓN 32](#_Toc385624570)

[3.2 FUENTES DE INFORMACIÓN 32](#_Toc385624571)

[3.2.1 Fuentes de información primaria 32](#_Toc385624572)

[3.2.2 Fuentes de información secundaria 32](#_Toc385624573)

[3.3 RECOLECCIÓN Y ANALISIS DE INFORMACIÓN 33](#_Toc385624574)

[4. ANALISIS DE LAS HERRAMIENTAS CASE 35](#_Toc385624575)

[4.1 GENERALIDADES 35](#_Toc385624576)

[4.1.1 Enterprise Architect 35](#_Toc385624577)

[4.1.2 Netbeans UML 38](#_Toc385624578)

[4.1.3 StarUML 40](#_Toc385624579)

[4.1.4 GenMyModel 42](#_Toc385624580)

[4.2 DISEÑO DE DIAGRAMAS DE CLASE 43](#_Toc385624581)

[4.2.1 Características Básicas 43](#_Toc385624582)

[4.2.2 Características Avanzadas 46](#_Toc385624583)

[4.3 MATRIZ DE CARACTERIZACIÓN 48](#_Toc385624584)

[4.4 TABULACIÓN Y ANALISIS 49](#_Toc385624585)

[4.4.1 Tabulación 49](#_Toc385624586)

[4.4.2 Análisis 49](#_Toc385624587)

[5. METODOLOGÍA DE DESARROLLO 50](#_Toc385624588)

[5.1 EXPLORACIÓN 50](#_Toc385624589)

[5.1.1 Roles del Proyecto 50](#_Toc385624590)

[5.1.2 Actores del Sistema 51](#_Toc385624591)

[5.1.3 Historias de Usuario 54](#_Toc385624592)

[5.1.4 Arquitectura de la Aplicación 71](#_Toc385624593)

[5.1.5 Definición de la Tecnología 73](#_Toc385624594)

[5.1.5.1 JSF 73](#_Toc385624595)

[5.1.5.2 Facelets 73](#_Toc385624596)

[5.1.5.3 PrimeFaces 74](#_Toc385624597)

[5.1.5.4 EJB 74](#_Toc385624598)

[5.1.5.5 JPA 75](#_Toc385624599)

[5.1.5.6 MxGraph 76](#_Toc385624600)

[5.1.6 Herramientas de Desarrollo 77](#_Toc385624601)

[5.1.6.1 Glassfish 77](#_Toc385624602)

[5.1.6.2 MySQL 77](#_Toc385624603)

[5.1.6.3 Eclipse 78](#_Toc385624604)

[5.2 PLANIFICACIÓN 78](#_Toc385624605)

[5.2.1 Priorización de Historias de Usuario 78](#_Toc385624606)

[5.2.2 Estimación de Historias de Usuario 79](#_Toc385624607)

[5.2.3 Definición de Iteraciones 80](#_Toc385624608)

[5.2.4 Metáfora del Sistema 81](#_Toc385624609)

[5.3 IMPLEMENTACIÓN 82](#_Toc385624610)

[5.3.1 Patrones de Desarrollo 82](#_Toc385624611)

[5.3.1.1 Modelo Vista Controlador 82](#_Toc385624612)

[5.3.1.2 Intercepting Filter 83](#_Toc385624613)

[5.3.1.3 Front Controller 84](#_Toc385624614)

[5.3.1.4 Session Facade 85](#_Toc385624615)

[5.3.1.5 Singleton 86](#_Toc385624616)

[5.3.1.6 Data Access Object 87](#_Toc385624617)

[5.3.2 Estructura del Proyecto 88](#_Toc385624618)

[5.3.3 Interfaz Gráfica de Usuario 89](#_Toc385624619)

[5.3.4 Iteración 1 92](#_Toc385624620)

[5.3.4.1 Actividades a Realizar 92](#_Toc385624621)

[5.3.4.2 Diagrama de Clases 93](#_Toc385624622)

[5.3.4.3 Modelo de Datos 95](#_Toc385624623)

[5.3.4.4 Interfaz grafica 98](#_Toc385624624)

[5.3.4.5 Controles 100](#_Toc385624625)

[5.3.4.6 Configuración de correo electrónico 100](#_Toc385624626)

[5.3.4.7 Servicios 101](#_Toc385624627)

[5.3.4.8 Pruebas de unidad 111](#_Toc385624628)

[5.3.5 Iteración 2 114](#_Toc385624629)

[5.3.5.1 Actividades a Realizar 114](#_Toc385624630)

[5.3.5.2 Diagrama de Clases 114](#_Toc385624631)

[5.3.5.3 Modelo de Datos 116](#_Toc385624632)

[5.3.5.4 Interfaz grafica 116](#_Toc385624633)

[5.3.5.5 Seguridad 119](#_Toc385624634)

[5.3.5.6 Servicios 121](#_Toc385624635)

[5.3.5.7 Pruebas de unidad 126](#_Toc385624636)

[5.3.6 Iteración 3 127](#_Toc385624637)

[5.3.7 Iteración 4 127](#_Toc385624638)

[5.3.8 Iteración 5 127](#_Toc385624639)

[5.3.9 Iteración 6 127](#_Toc385624640)

[5.3.10 Iteración 7 127](#_Toc385624641)

[5.3.11 Iteración 8 127](#_Toc385624642)

[5.3.12 Iteración 9 127](#_Toc385624643)

[5.4 PRODUCCIÓN 127](#_Toc385624644)

[6. IMPLEMENTACIÓN 129](#_Toc385624645)

[7. PRUEBAS 130](#_Toc385624646)

[8. DOCUMENTACIÓN 131](#_Toc385624647)

[CONCLUSIONES 132](#_Toc385624648)

[RECOMENDACIONES 133](#_Toc385624649)

[BIBLIOGRAFIA 134](#_Toc385624650)

# LISTADO DE FIGURAS

[Figura 1. Jerarquía de los Diagramas de UML 2.0 22](#_Toc385624651)

[Figura 2. Interfaz de usuario principal de las herramientas CASE 44](#_Toc385624652)

[Figura 3. Arquitectura de la aplicación CLASS Modeler 71](#_Toc385624653)

[Figura 4. Arquitectura de la librería mxGraph 76](#_Toc385624654)

[Figura 5. Patrón Modelo-Vista-Controlador 83](#_Toc385624655)

[Figura 6. Estructura y distribución del código de la aplicación CLASS Modeler 88](#_Toc385624656)

[Figura 7. Plantillas principales para la interfaz gráfica de la aplicación 90](#_Toc385624657)

[Figura 8. Jerarquía de las páginas basadas en plantillas 91](#_Toc385624658)

[Figura 9. Interfaz gráfica principal de la aplicación CLASS Modeler 98](#_Toc385624659)

[Figura 10. Interfaz gráfica para la creación de cuenta de diagramador 99](#_Toc385624660)

[Figura 11. Interfaz gráfica para solicitud de reasignación de contraseña 99](#_Toc385624661)

[Figura 12. Interfaz gráfica para reasignar la contraseña 100](#_Toc385624662)

[Figura 13. Resultados de las pruebas unitarias del servicio *UserServiceTest* 113](#_Toc385624663)

[Figura 14. Resultados de las pruebas unitarias del servicio *SecurityServiceTest* 113](#_Toc385624664)

[Figura 15. Resultados de las pruebas unitarias del servicio *EmailServiceTest* 113](#_Toc385624665)

[Figura 16. Formulario de inicio de sesión 117](#_Toc385624666)

[Figura 17. Tablero de control para listar los diagramas del usuario 117](#_Toc385624667)

[Figura 18. Formulario de creación de diagramas. 119](#_Toc385624668)

[Figura 19. Diagrama de secuencia de la historia de usuario H9 124](#_Toc385624669)

[Figura 20. Resultados de las pruebas de unidad del servicio SessionService 126](#_Toc385624670)

[Figura 21. Resultados de las pruebas unitarias del servicio DiagramService 126](#_Toc385624671)

# LISTADO DE DIAGRAMAS

[Diagrama 1. Diagrama de actores de la aplicación CLASS Modeler 53](#_Toc385624672)

[Diagrama 2. Diagrama de clases del dominio de la iteración 1 94](#_Toc385624673)

[Diagrama 3. Modelo de datos para la iteración 1 96](#_Toc385624674)

[Diagrama 4. Diagrama de clases de los servicios para la iteración 1 102](#_Toc385624675)

[Diagrama 5. Diagrama de clases de implementacion para servicos de la iteración 1 103](#_Toc385624676)

[Diagrama 6. Diagrama de secuencia para creación de cuenta de diagramador H1 104](#_Toc385624677)

[Diagrama 7. Diagrama de secuencia para activación de cuenta de diagramador H2 106](#_Toc385624678)

[Diagrama 8. Diagrama de secuencia para solicitud de nueva contraseña H3 108](#_Toc385624679)

[Diagrama 9. Diagrama de secuencia para reasignación de contraseña H3 109](#_Toc385624680)

[Diagrama 10. Diagrama de clases de la iteración 2 115](#_Toc385624681)

[Diagrama 11. Modelo de datos de la iteración 2 116](#_Toc385624682)

[Diagrama 12. Proceso de filtro de peticiones hacia las páginas de la aplicación 120](#_Toc385624683)

[Diagrama 13. Diagrama de clases de los servicios para la iteración 2 121](#_Toc385624684)

[Diagrama 14. Diagrama de secuencia de la historia de usuario H6 122](#_Toc385624685)

# LISTADO DE TABLAS

[Tabla 1. Características generales de Enterprise Architect 38](#_Toc385624725)

[Tabla 2. Características generales de Netbeans UML 39](#_Toc385624726)

[Tabla 3. Características principales de StarUML 41](#_Toc385624727)

[Tabla 4. Características generales de GenMyModel 43](#_Toc385624728)

[Tabla 5. Matriz de características de herramientas CASE para diagramas de clase 48](#_Toc385624729)

[Tabla 6. Especificación del actor *Visitante* 51](#_Toc385624730)

[Tabla 7. Especificación del actor *Usuario* 52](#_Toc385624731)

[Tabla 8. Especificación del actor *Invitado* 52](#_Toc385624732)

[Tabla 9. Especificación del actor *Diagramador* 53](#_Toc385624733)

[Tabla 10. Formato de historia de usuario H1 54](#_Toc385624734)

[Tabla 11. Formato de historia de usuario H2 55](#_Toc385624735)

[Tabla 12. Formato de historia de usuario H5 56](#_Toc385624736)

[Tabla 13. Formato de historia de usuario H4 57](#_Toc385624737)

[Tabla 14. Formato de historia de usuario H5 58](#_Toc385624738)

[Tabla 15. Formato de historia de usuario H6 58](#_Toc385624739)

[Tabla 16. Formato de historia de usuario H7 59](#_Toc385624740)

[Tabla 17. Formato de historia de usuario H8 60](#_Toc385624741)

[Tabla 18. Formato de historia de usuario H9 60](#_Toc385624742)

[Tabla 19. Formato de historia de usuario H10 61](#_Toc385624743)

[Tabla 20. Formato de historia de usuario H11 62](#_Toc385624744)

[Tabla 21. Formato de historia de usuario H12 63](#_Toc385624745)

[Tabla 22. Formato de historia de usuario H13 63](#_Toc385624746)

[Tabla 23. Formato de historia de usuario H14 64](#_Toc385624747)

[Tabla 24. Formato de historia de usuario H15 65](#_Toc385624748)

[Tabla 25. Formato de historia de usuario H16 66](#_Toc385624749)

[Tabla 26. Formato de historia de usuario H17 67](#_Toc385624750)

[Tabla 27. Formato de historia de usuario H18 68](#_Toc385624751)

[Tabla 28. Formato de historia de usuario H19 68](#_Toc385624752)

[Tabla 29. Formato de historia de usuario H17 69](#_Toc385624753)

[Tabla 30. Formato de historia de usuario H21 69](#_Toc385624754)

[Tabla 31. Formato de historia de usuario H22 70](#_Toc385624755)

[Tabla 32. Formato de historia de usuario H23 70](#_Toc385624756)

[Tabla 33. Priorización de historias de usuario 79](#_Toc385624757)

[Tabla 34. Estimación de tiempo para historias de usuario 80](#_Toc385624758)

[Tabla 35. Plan de iteraciones 81](#_Toc385624759)

[Tabla 37. Listado de pruebas de unidad para la iteración 1 111](#_Toc385624760)

[Tabla 38. Listado de pruebas de unidad para la iteración 2 126](#_Toc385624761)

# INTRODUCCIÓN

En el desarrollo de software es común usar herramientas que faciliten las tareas complejas y agilicen el trabajo manual; la mayoría de las empresas dedicadas al desarrollo de software incorporan aplicaciones a su quehacer diario, permitiendo a los trabajadores llevar a cabo su labor de manera más eficiente, conjunta y automatizada.

Este grupo de aplicaciones son llamadas Herramientas CASE, debido a las siglas tomadas del nombre en inglés “Computer Aided Software Engineering”, traducido al español como Ingeniería de Software asistida por computadora. Estas herramientas son la base para la creación de todas las aplicaciones software que vemos hoy en día; cada una se especializa en diferentes aspectos, dependiendo de la, o las fases del proceso de desarrollo que apoyan; algunas permiten controlar prácticamente todo el ciclo de vida de una aplicación desde el análisis de requerimientos hasta el despliegue y mantenimiento.

Con el desarrollo de este proyecto se pretende incorporar el concepto de herramienta CASE en una aplicación web, aprovechando las bondades que brinda el internet y resolviendo una necesidad particular de nuestro ambiente universitario. De este modo los estudiantes podrán disponer de una herramienta básica para su formación y que posea las características necesarias para ser considerada una aplicación en la nube.

El objetivo principal del proyecto es el desarrollo de una herramienta para el modelado de Diagramas de Clase UML, que permita a los estudiantes y profesores, crear y manipular, en línea, los componentes básicos de este tipo de diagrama y llegando hasta obtener archivos de código fuente generados a partir de los elementos del mismo.

*CLASS Modeler* (Modelador de clases), si bien es una herramienta básica para la creación de elementos de diagramas de clase, está basado en el documento de especificación formal del estándar UML en su versión 2.4, utilizando una implementación en lenguaje Java de del meta modelo propuesto por el OMG; el cual comprende un conjunto de clases, que representan los elementos que se pueden crear en cada tipo de diagrama. Esto en conjunto con un sistema de plantillas, permite generar una representación en código fuente del modelo diseñado.

Este documento explica el proceso seguido para la ejecución del proyecto y el desarrollo de la herramienta *CLASS Modeler*; así como las técnicas y conceptos de ingeniería de software utilizados para la creación de este producto.

Los 3 primeros capítulos del documento están destinados a la presentación del proyecto; exponiendo la necesidad detectada, los objetivos del proyecto y la justificación del mismo; además de las bases teóricas y conceptuales necesarias para su ejecución.

El capítulo 4 se enfoca en una investigación sobre el estado del arte de las herramientas CASE y el análisis de las funcionalidades que estas ofrecen a los usuarios, específicamente para la creación de diagramas de clase. Esto sirve de base o suministro para los capítulos subsiguientes, de tal manera que se puedan determinar las características mínimas que debería tener una herramienta de este tipo.

El capítulo 5 comprende la aplicación de la metodología ágil *XP Extreme Programming*, seleccionada para el desarrollo de la herramienta. Siguiendo las pautas y fases de la metodología XP se escriben inicialmente las historias de usuario, que determinan las funcionalidades que el software ofrecerá; se hace una estimación basada en tiempo y prioridad, y se establecen las iteraciones necesarias para el desarrollo completo del software. Adicionalmente se realiza un análisis de la tecnología utilizada para la implementación de la herramienta, incluyendo aspectos de arquitectura y patrones de diseño.

El capítulo 6 describe la ejecución de las iteraciones, las cuales establecen las actividades que se deben realizar, el análisis de las historias de usuario y los aspectos técnicos de implementación e ingeniería de software.

Por último, el capítulo 7 describe las actividades realizadas como parte de la fase de pruebas y verificación de la herramienta CLASS Modeler. Esta sección documenta las actividades de realizadas con estudiantes y profesores del plan de estudios de Ingeniería de Sistema, junto con los resultados de la encuesta aplicada para conocer el nivel de satisfacción y facilidad de uso que ellos perciben mediante la interacción con la herramienta.

# PRESENTACIÓN DEL PROYECTO

## TITULO

CONSTRUCCIÓN DE UNA APLICACIÓN WEB PARA LA ELABORACIÓN DE DIAGRAMAS DE CLASE, ORIENTADA BAJO EL MODELO DE SERVICIOS DE INTERNET CLOUD COMPUTING.

## PLANTEAMIENTO DEL PROBLEMA

En estudios realizados por el grupo de investigación y desarrollo de software (GIDIS) de la Universidad Francisco de Paula Santander, después convertido en un proyecto de grado [1], se ha determinado que unos de los principales factores involucrados en los altos índices de mortalidad y deserción académica en la carrera de Ingeniería de Sistemas de las universidades de Cúcuta, son la falta de dedicación al estudio por parte de los estudiantes, poca comprensión de temas como abstracción y modelamiento de sistemas, y falta de ambientes prácticos donde los estudiantes tuvieran la posibilidad de llevar a cabo actividades para afianzar los conceptos aprendidos en clase. Teniendo esto en cuenta se hace evidente que se necesitan mecanismos para que los estudiantes ejerciten sus habilidades y de esta manera se disminuya esta problemática que ha afectado considerablemente al programa académico.

Dentro del programa de Ingeniería de Sistemas de la UFPS se tiene un clara tendencia e inclinación hacia el desarrollo de software, es claro para muchos estudiantes que este es uno de los puntos fuertes de la carrera y es por ello que muchos continúan su vida profesional por ese camino; por tal motivo desde los primeros semestres se trata de cultivar estas habilidades en ellos; considerando que el modelamiento y abstracción de sistemas son temas de vital importancia, se hace necesario disponer de herramientas automáticas (software) que permitan adquirir competencias en estas áreas y reforzar conceptos aprendidos en clase, pero que solo se consiguen mediante la práctica; además que permitan al estudiante forjar unos cimientos importantes para su vida profesional en aspectos tales como UML, Diagramas de clase, CASE Tools[[1]](#footnote-1) y razonamiento abstracto.

Por otro lado, se es consciente que actualmente ya existen herramientas muy completas que permiten realizar estas labores y que el programa académico ha venido utilizando desde hace algún tiempo (por ejemplo Enterprise Architect[[2]](#footnote-2), Netbeans UML, StarUML, ArgoUML), además que la universidad dispone de licencias de uso para algunas de ellas (por ejemplo, Enterprise Architect que es software privado), y otras son software libre de modo que pueden ser utilizados de cualquier forma; sin embargo, la mayoría de ellas, por no decir todas, son software que se instala de manera nativa, de modo que carecen de portabilidad y facilidad de acceso, aspectos que son muy importantes y atractivos hoy en día en los software de aplicación. Otras requieren ciertas características mínimas de hardware que aunque no es muy común, no siempre están disponibles y son suficientes para soportar que estas aplicaciones se ejecuten de manera adecuada.

Adicional pero no menos importante, se ha notado mediante observaciones indirectas y manifestaciones explicitas de algunos estudiantes, sobre todo en primeros semestres, que muchos de ellos aún tienen dificultades en la configuración de estas aplicaciones, otros expresan no tener computadoras propias o simplemente que los recursos hardware son insuficientes para el funcionamiento adecuado de las aplicaciones, aspectos que evidencian la necesidad planteada anteriormente e impulsan a diseñar modelos de servicios accesibles a través internet, los cuales podrían brindar ciertas flexibilidades y beneficios extras para los usuarios que no tienen las aplicaciones nativas, beneficios tales como facilidad de acceso, portabilidad, autoconfiguración, capacidad de compartir recursos e información.

## JUSTIFICACIÓN

El programa académico de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, tiene como objeto de estudio la “Construcción y Gestión de Sistemas de Información”[[3]](#footnote-3), por esta razón dentro del pensum existen muchos cursos que se enfocan en programación e ingeniería de software; esto significa que hay una clara tendencia a tratar temas como el modelado de aplicaciones, procesos de desarrollo y otros conceptos que hacen parte de esta área de la ingeniería de sistemas; de esta manera consideramos que se necesita que los estudiantes pongan en práctica técnicas y métodos para desarrollar y mantener software de calidad, teniendo siempre en mente que la tecnología existe para ser usada como herramienta facilitadora del trabajo.

Lo anterior sugiere la necesidad de disponer de una herramienta que permita realizar prácticas de laboratorio, más específicamente prácticas de abstracción de diagramas de clase, de modo que se puedan aplicar todos los conceptos aprendidos, bajo un ambiente real. Es por ello que nos hemos enfocado específicamente en este aspecto, de tal manera que se pueda obtener una herramienta adicional que sirva de apoyo al proceso de aprendizaje y permita adquirir con seguridad conceptos vitales y que serán de gran ayuda en la vida profesional de los estudiantes.

De igual manera sería ideal que los docentes puedan disponer de un mecanismo de control y supervisión de las actividades realizadas por sus estudiantes, permitiendo de manera fácil y cómoda la revisión de los trabajos de clase o evaluaciones parciales; por tal motivo se pretende construir una aplicación orientada bajo el modelo de servicios “Cloud Computing”[[4]](#footnote-4), e incorporar comportamientos como los desarrollados por “Google Drive”[[5]](#footnote-5), en donde los usuarios pueden compartir información con otros de manera fácil y cómoda a través de un clic. Aplicando este concepto a la aplicación en construcción, se espera que los estudiantes puedan compartir con su docente los diagramas realizados de modo que el docente pueda consultarlos y corregirlos a través de la aplicación. Además es significativo incentivar el autoaprendizaje, permitiendo a los estudiantes el intercambio de conocimientos e información, de manera que puedan reforzar los conceptos, obviamente mediante algunos mecanismos de control que garanticen un proceso limpio y transparente.

Adicionalmente una motivación importante para el desarrollo de este proyecto es mejorar la infraestructura de servicios y de tecnologías de información disponibles para los estudiantes de la UFPS, mediante la construcción de herramientas innovadoras y que generen conocimientos sobre el desarrollo de aplicaciones web.

Por último, otro aspecto importante es la necesidad de impulsar e innovar en el desarrollo de tecnologías, el compromiso que tiene el plan de estudios de Ingeniería de Sistemas con la universidad y con la comunidad en general, para desarrollar alternativas de solución a problemáticas mediante el uso de la tecnología y generar conocimiento en base a investigación e innovación nos ha llevado a esta iniciativa, esperando propulsar el desarrollo de nuevos productos y herramientas que presten servicios dentro y fuera de la universidad, y que sean realizadas por los mismos estudiantes.

## OBJETIVOS

### Objetivo General

Implementar una aplicación web basada en el paradigma orientado a servicios de Cloud Computing, que permita el diseño y construcción de diagramas de clase UML y que sirva como medio de apoyo para proyectos de software realizados por los estudiantes de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander.

### Objetivos Específicos

* Realizar un estudio de las principales características que poseen las herramientas CASE utilizadas por los estudiantes del programa de Ingeniería de Sistemas, para determinar los servicios que prestan en cuanto al diseño de diagramas de clase.
* Definir las funcionalidades que el software debe poseer, determinando las características específicas, los servicios que debe prestar a los usuarios y los procedimientos que debe llevar a cabo.
* Realizar el modelado del sistema, determinando una solución óptima a los procedimientos definidos, de modo que no se vea afectado el rendimiento y la estabilidad del software a construir.
* Realizar pruebas de funcionamiento que garanticen el comportamiento adecuado del software y que permitan verificar que se cumplan los requerimientos definidos.
* Documentar las pruebas realizadas junto con los procedimientos llevados a cabo y los resultados obtenidos.
* Desplegar la aplicación en un servidor de modo que pueda ser accedido por estudiantes y docentes.

## ALCANCES Y DELIMITACIONES

### Alcance

La ejecución de este proyecto pretende alcanzar lo siguiente:

* Obtener un producto software que permita a los usuarios diseñar de manera visual diagramas de clase UML.
* Disponer de una plataforma online que pueda ser utilizada desde cualquier lugar y a cualquier hora a través de internet.
* Disponer de una herramienta que permita a los docentes asesorar de manera continua y cómoda a sus estudiantes en los proyectos desarrollo de software.
* Un software para realizar modelado de objetos que sirva como soporte de los proyectos de software y permita generar código fuente de manera automática, agilizando el proceso de desarrollo.
* Un documento mediante el cual se pueda conocer el funcionamiento básico del software y los servicios que este presta, que sirva como guía a los usuarios sobre la manera en la cual deben utilizar la herramienta (Manual de usuario).

### Delimitaciones

Con el propósito de determinar la viabilidad y posibilidad de desarrollo del proyecto se han establecido las delimitaciones mencionadas a continuación:

* El software será totalmente independiente del Sistema de Información Académico de la Universidad (SIA), de modo que no se validara que los usuarios registrados estén en la base de datos del mismo.
* Solo se podrá realizar diagramas de clase UML, la aplicación no abarca el diseño de otro tipo de diagrama.
* El software no dispondrá de un administrador, de modo que no será necesario disponer de usuarios que realicen configuraciones específicas y únicas en el software.
* El código fuente generado a partir de los diagramas de clase será únicamente en lenguaje Java.
* Considerando que existen varios cursos de programación e Ingeniería de Software dentro del pensum académico actual del programa de Ingeniería de Sistemas, el proyecto no estará dirigido específicamente a un curso en particular, sino que estará disponible para cualquier estudiante en cualquier curso; sin embargo, se tomara una población base de uno o dos cursos para realizar el levantamiento de información. En el diseño metodológico se especifica con más detalle este tema.
* El software no permitirá editar el código fuente generado directamente, para esta labor se hace necesario que el usuario disponga de un editor.

# MARCO REFERENCIAL

## MARCO DE ANTECEDENTES

Después de realizar una investigación de las soluciones existentes a la problemática planteada, se encontraron algunas aplicaciones en el mercado que ofrecen parte de las características que esperamos obtener con el software en construcción. A continuación se mencionan algunas de estas aplicaciones:

### GenMyModel

GenMyModel es una herramienta UML gratuita online con generadores de código para arquitectos de software y desarrolladores. Axellience, que es el editor del software GenMyModel es una compañía Startup[[6]](#footnote-6) Francesa fundada en abril de 2012, después de tres años de investigación y desarrollo.

La aplicación se enfoca en la creación de diagramas de clase, casos de uso y diagramas de actividad; permite personalizar propiedades, tipos y relaciones como un modelador de escritorio. Es una herramienta completamente basada en web, donde no es necesario instalar ninguna aplicación, ni de realizar ninguna configuración [2].

### Draw.io

Draw.io es una aplicación de diagramado online desarrollada usando la librería JavaScript mxGraph creada por la compañía JGraph Ltd. Esta empresa fue fundada en el año 2000 y está dedicada únicamente a la comercialización y soporte de componentes gráficos de visualización desde el 2004. Actualmente se ha impuesto en el mercado con su producto líder *mxGraph*, el cual comprende una librería JavaScript que permite dibujar componentes para diagramas en cualquier navegador web usando HTML, CSS, JavaScript y SVG, además que permite realizar el Backend[[7]](#footnote-7) de las aplicaciones en diferentes lenguajes como Java, PHP y .NET [3].

Draw.io es el sitio de demostración para la tecnología desarrollada por la empresa JGraph y su producto mxGraph, el cual permite dibujar diagramas creativos en el navegador, compartir y editar con otros usuarios en tiempo real.

### Creately

Creately es una herramienta online UML para crear diagramas de Casos de uso, Diagramas de clase, Actividad, Secuencia, Modelos Entidad Relación, entre muchos otros. Esta es una herramienta comercial la cual es propiedad de la empresa Cinergix Pty Ltd., es el primer producto de la empresa y se encuentra actualmente en una etapa madura; es posible acceder a él en modo prueba aunque no se pueden guardar los proyectos y las funciones son un poco limitadas; para poder tener a disposición toda la herramienta es necesario comprar una licencia de uso. Esta herramienta permite la generación de código fuente, documentación, ingeniería inversa, entre otras funciones; características que lo hacen muy llamativo aunque cuenta con la limitante de ser software propietario [4].

### GWTUML

GWTUML es una herramienta open source desarrollada por la empresa francesa Object Direct, dirigido por el ODLabs departament. Esta es una compañía dedicada a la consultoría y el desarrollo de tecnologías de información utilizando metodologías de desarrollo ágil, orientación a objetos y Cloud computing, brindando soporte a proyectos open source.

GWTUML es una herramienta CASE en proceso de desarrollo, la cual permite el modelado visual de diagramas de clase, diagramas de objetos y diagramas de secuencia. El desarrollo de esta herramienta se inició en enero de 2009 y la versión actual aún se encuentra en modo experimental, sin embargo, los desarrolladores han liberado algunas APIs[[8]](#footnote-8) y versiones beta de la aplicación, las cuales pueden ser descargadas y utilizadas en otros proyectos que requieran sus servicios. Adicionalmente existe alguna documentación del código fuente (Javadocs) de modo que los usuarios pueden consultarlos en caso de utilizar las APIs disponibles.

Este proyecto está basado en GWT[[9]](#footnote-9) (Google Web Toolkit) el cual es un Framework para el desarrollo de aplicaciones web en lenguaje java, liberando al desarrollador de la complejidad implícita de varios aspectos de la tecnología AJAX, que a su vez es la herramienta más utilizada hoy en día para la construcción de aplicaciones para la internet

### JoinJS

JoinJS es una librería JavaScript para crear diagramas totalmente interactivos en la web. Esta librería puede ser usada tanto para implementar herramientas de diagramado o para simplemente publicar diagramas online. Página principal de la herramienta [5].

Las principales características de esta librería son:

* Permite conectar objetos vectoriales con varios tipos de líneas y flechas.
* Interacción con objetos y conectores.
* Personalizar eventos y manejadores para mouse, teclado, entre otros.
* Elementos pre construidos para diagramas comúnmente usados (Entidad relación, UML, Org Chart, entre otros).
* Serialización (formato JSON, exportación a SVG en navegadores que lo permiten).
* Soporte para la mayoría de navegadores en versiones antiguas.

## MARCO CONCEPTUAL

### Herramientas CASE

Se puede definir a las Herramientas CASE (Computer Aided Software Engineering) como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un Software [6].

La realización de un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Las Herramientas CASE fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de software.

La mejor razón para la creación de estas herramientas fue el incremento en la velocidad de desarrollo de los sistemas. Por esto, las compañías pudieron desarrollar sistemas sin encarar el problema de tener cambios en las necesidades del negocio, antes de finalizar el proceso de desarrollo. Las herramientas CASE también permiten a los analistas tener más tiempo para el análisis y diseño y minimizar el tiempo para codificar y probar. Con un CASE integrado, las organizaciones pueden desarrollar rápidamente sistemas de mejor calidad para soportar procesos críticos del negocio y asistir en el desarrollo y promoción intensiva de la información de productos y servicios.

Estas herramientas pueden proveer muchos beneficios en todas las etapas del proceso de desarrollo de software, algunos de estos beneficios son:

* Verificar el uso de todos los elementos en el sistema diseñado.
* Automatizar el dibujo de diagramas.
* Ayudar en la documentación del sistema.
* Ayudar en la creación de relaciones en la Base de Datos.
* Generar estructuras de código.

Las herramientas CASE, en función de las fases del ciclo de vida abarcadas, se pueden agrupar de la forma siguiente:

* Herramientas integradas, I-CASE (Integrated CASE, CASE integrado): abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.
* Herramientas de alto nivel, U-CASE (Upper CASE - CASE superior) o Front-End, orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo: análisis y diseño.
* Herramientas de bajo nivel, L-CASE (Lower CASE - CASE inferior) o Back-End, dirigidas a las últimas fases del desarrollo: construcción e implantación.

### UML

El UML (Lenguaje Unificado de Modelado) es una de las herramientas más emocionantes en el mundo actual de desarrollo de sistemas. Esto se debe a que permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas [6].

La comunicación de la idea es de suma importancia. Antes del advenimiento del UML, el desarrollo de sistemas era, con frecuencia, una propuesta al azar. Los analistas de sistemas intentaban evaluar los requerimientos de sus clientes, generar un análisis de requerimientos en algún tipo de notación que ellos mismos comprendieran (aunque el cliente no lo comprendiera), dar tal análisis a uno o más programadores y esperar que el producto final cumpliese con lo que el cliente deseaba.

Dado que el desarrollo de sistemas es una actividad humana, hay muchas posibilidades de cometer errores en cualquier etapa del proceso, por ejemplo, el analista pudo haber malentendido al cliente, es decir, probablemente produjo un documento que el cliente no pudo comprender. Tal vez ese documento tampoco fue comprendido por los programadores quienes, por ende, pudieron generar un programa difícil de utilizar y no generar una solución al problema original del cliente.

UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que UML es un lenguaje, cuenta con reglas para combinar tales elementos entre los cuales encontramos: diagrama de clases, diagrama de objetos, diagrama de casos de uso, diagrama de estados, diagrama de secuencias, diagrama de actividades, diagrama de colaboraciones, diagrama de componentes y diagrama de distribución.

El estándar ofrece una variedad de diagramas, cada uno representa una vista del sistema en construcción, con diferente nivel de abstracción. La figura 1 muestra la jerarquía que existe en los diferentes diagramas del estándar UML versión 2.0.



Figura . Jerarquía de los Diagramas de UML 2.0

### Diagramas de Clase

Los diagramas de clases muestran un conjunto de clases, interfaces y colaboraciones, así como sus relaciones, estos se utilizan para modelar la vista de diseño estática de un sistema, es por ello que están incluidos dentro de los diagramas estructurales del lenguaje UML.

El representar la vista estática del sistema, implica principalmente, modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas. Los diagramas de clases también son la base para un par de diagramas relacionados, los diagramas de componentes y los diagramas de despliegue. Los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino que también para construir sistemas ejecutables aplicando ingeniería directa e inversa [7].

Un diagrama de clases es un diagrama que muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones. Al igual que otros diagramas los diagramas de clases pueden contener notas y restricciones. También pueden contener paquetes o subsistemas, los cuales se usan para agrupar los elementos de un modelo en partes más grandes. A veces se colocarán instancias en los diagramas de clases, especialmente cuando se quiera mostrar el tipo (posiblemente dinámico) de una instancia.

### Cloud Computing

La nube (Cloud Computing) es el procesamiento masivo de datos y almacenamiento de información en servidores, ubicados en cualquier parte del mundo,  conectados a una conexión de internet y a los que se puede acceder desde cualquier dispositivo: computador, Smartphone, tabletas, etc.

En los años 70’s John McCarthy, famoso científico e informático ganador del Premio Turing, expresó: “algún día la computación podrá ser organizada como un servicio público”. El primer uso académico de la expresión “computación en nube”, fue durante una conferencia de RamnathChellappa (actual profesor de la EmoryUniversity) en 1997. Desde ese momento el famoso concepto de la nube ha venido trascendiendo y hoy en día estamos entiendo con más claridad a qué se referían estos visionarios de la tecnología.

EL NIST (National Institute of Standards and Technology) de EE.UU lo define como: “Cloud Computing es un modelo para permitir el acceso adecuado y bajo demanda a un conjunto de recursos de cómputo configurables (ejemplos: redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente provistos y puestos a disposición del cliente con un mínimo esfuerzo de gestión y de interacción con el proveedor del servicio”.

Para los usuarios finales, el cómputo cloud significa que no existen los costos de adquisición de hardware, ni el manejo de las licencias de software o de actualizaciones, ni nuevos empleados o consultores que contratar, ni instalaciones que rentar, ni costos de capital de ninguna clase — ni costos ocultos. Sólo una tarifa medida por uso o una cuota fija de subscripción. Use sólo lo que quiera, pague sólo lo que usa.

**Modelos de servicios**

* Software como servicio: El software como servicio (en inglés software as a service, SaaS) se encuentra en la capa más alta y caracteriza una aplicación completa ofrecida como un servicio, en demanda, vía multi-tendencia que significa una sola instancia del software que corre en la infraestructura del proveedor y sirve a múltiples organizaciones de clientes. Un ejemplo es la plataforma MS Office como servicio SaaS con su denominación de Microsoft Office 365, que incluye versiones online de la mayoría de las aplicaciones de esta suite ofimática de Microsoft.
* Plataforma como servicio: La capa del medio, que es la plataforma como servicio (en inglés platform as a service, PaaS), es la encapsulación de una abstracción de un ambiente de desarrollo y el empaquetamiento de una serie de módulos o complementos que proporcionan, normalmente, una funcionalidad horizontal (persistencia de datos, autenticación, mensajería, etc.). De esta forma, un arquetipo de plataforma como servicio podría consistir en un entorno conteniendo una pila básica de sistemas, componentes o APIs pres configurados y listos para integrarse sobre una tecnología concreta de desarrollo (por ejemplo, un sistema Linux, un servidor web, y un ambiente de programación como Perl o Ruby). Un ejemplo de este tipo de nube es Google App Engine que sirven aplicaciones de Google y Microsoft.
* Infraestructura como servicio: La infraestructura como servicio (infrastructure as a service, IaaS) -también llamado en algunos casos hardware as a service, HaaS)6 se encuentra en la capa inferior y es un medio de entregar almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red. Servidores, sistemas de almacenamiento, conexiones, enrutadores, y otros sistemas se concentran (por ejemplo a través de la tecnología de virtualización) para manejar tipos específicos de cargas de trabajo, desde procesamiento en lotes (batch) hasta aumento de servidor/almacenamiento durante las cargas pico.

**Tipos de nubes**

* Las nubes públicas se manejan por terceras partes, y los trabajos de muchos clientes diferentes pueden estar mezclados en los servidores, los sistemas de almacenamiento y otras infraestructuras de la nube. Los usuarios finales no conocen qué trabajos de otros clientes pueden estar corriendo en el mismo servidor, red, discos como los suyos propios.
* Las nubes privadas son una buena opción para las compañías que necesitan alta protección de datos y ediciones a nivel de servicio. Las nubes privadas están en una infraestructura en-demanda manejada por un solo cliente que controla qué aplicaciones debe correr y dónde. Son propietarios del servidor, red, y disco y pueden decidir qué usuarios están autorizados a utilizar la infraestructura.

Las nubes híbridas combinan los modelos de nubes públicas y privadas. Usted es propietario de unas partes y comparte otras, aunque de una manera controlada. Las nubes híbridas ofrecen la promesa del escalado aprovisionada externamente, en-demanda, pero añaden la complejidad de determinar cómo distribuir las aplicaciones a través de estos ambientes diferentes.

## MARCO TEORICO

### Java

El significado de Java tal y como se le conoce en la actualidad es el de un lenguaje de programación y un entorno para ejecución de programas escritos en el lenguaje Java. AI contrario que los compiladores tradicionales, que convierten el código fuente en instrucciones a nivel de máquina, el compilador Java traduce el código fuente Java en instrucciones que son interpretadas por la Máquina Virtual Java (JVM, *Java Virtual Machine)* [8].

**Java Virtual Machine (JVM)**

La JVM es una de las piezas fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al Hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un puente que entiende tanto el bytecode, como el sistema sobre el que se pretende ejecutar. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada en una máquina virtual Java en concreto, siendo ésta la que en última instancia convierte de código bytecode a código nativo del dispositivo final. La gran ventaja de la máquina virtual java es aportar portabilidad al lenguaje de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales java para diferentes arquitecturas y así un programa .class escrito en un Windows puede ser interpretado en un entorno Linux. Tan solo es necesario disponer de dicha máquina virtual para dichos entornos.

**Componentes Java**

*J2EE* es un grupo de especificaciones diseñadas por Sun que permiten la creación de aplicaciones empresariales, esto sería: acceso a base de datos (JDBC), utilización de directorios distribuidos (JNDI), acceso a métodos remotos (RMI/CORBA), funciones de correo electrónico (Java Mail). Java EE también configura algunas especificaciones únicas para componentes Java EE, estas incluyen: Enterprise JavaBeans EJB, Servlets, Portlets (siguiendo la especificación de Portlets Java), Java Server Pages y varias tecnologías de servicios web. Ello permite al desarrollador crear aplicaciones empresariales portables entre plataformas y escalables, a la vez que integrable con tecnologías anteriores[[10]](#footnote-10).

*J2SE* es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java. La Plataforma Java 2 Enterprise Edition incluye todas las clases en el Java SE, además de algunas de las cuales son útiles para programas que se ejecutan en servidores sobre estaciones de trabajo.

*J2ME* es la versión de Java orientada a los dispositivos móviles. Debido a que los dispositivos móviles tienen una potencia de cálculo baja e interfaces de usuario pobres, es necesaria una versión específica de Java destinada a estos dispositivos, ya que el resto de versiones de Java, J2SE o J2EE, no encajan dentro de este esquema. J2ME es por tanto, una versión “reducida” de J2SE.

### Frameworks

En el desarrollo de software, un Framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Dentro de este tipo de aplicaciones, Java juega un papel muy importante actualmente ya que este es uno de los usos más comunes que se le da a este lenguaje de programación, además de que representa uno de los mejores medios para construir dichas aplicaciones.

Aunque no existe una clasificación estándar y formal que se aplique a los Frameworks para aplicaciones web, existen algunos aspectos que se pueden considerar para diferenciar unos de otros. De esta manera y teniendo en cuenta el enfoque de los Frameworks, se pueden mencionar dos tipos para el lenguaje Java:

* **Frameworks de aplicación**: Los Frameworks de aplicación son aquellos que están basados en los Request HTTP, es decir hacen uso del API Servlet. Así mismo, estos Frameworks no se preocupan acerca de los detalles de cómo se renderiza la interfaz de usuario y no hacen distinción entre acciones que solo afectan a la interfaz de usuario y aquellas que requieren ser procesadas por código de la aplicación.
* **Frameworks de Interfaz de usuario**: Estos Frameworks están basados en componentes, se enfocan en los detalles de la interfaz de usuario y no se interesan en cómo es implementado el resto de la aplicación. De igual forma, definen un API detallado para los componentes que formaran la interfaz de usuario con los objetivos de ligar a estos con la lógica de la aplicación adecuada, determinar qué acciones de los usuarios resultaran eventos de la interfaz y como serán manejados estos últimos, entre otras finalidades.

De entre todos los Frameworks para el desarrollo en Java, sobresalen los siguientes:

* **Struts**: Framework de aplicación open source, este fue el primer Framework en salir para el desarrollo web en Java.
* **Java Server Faces (JSF)**: Framework de interfaz de usuario, es el estándar de Java actualmente para el desarrollo web.
* **Spring**: Framework de aplicación muy popular últimamente.

### Extreme Programming (XP)

Metodología ágil basada en cuatro principios: simplicidad, comunicación, retroalimentación y valor. Además, orientada por pruebas y refactorización, se diseña e implementan las pruebas antes de programar la funcionalidad, el programador crea sus propias Pruebas de Unidad[[11]](#footnote-11).

Este método es típicamente atribuido a Kent Beck, Ron Jeffries y Ward Cinningham. El objetivo de XP son grupos pequeños y medianos de construcción de software en donde los requisitos aún son muy ambiguos, cambian rápidamente o son de alto riesgo. XP busca la satisfacción del cliente tratando de mantener durante todo el tiempo su confianza en el producto. Además, sugiere que el lugar de trabajo sea una sala amplia, si es posible sin divisiones (en el centro los programadores, en la periferia los equipos individuales). Una ventaja del espacio abierto es el incremento en la comunicación y el proporcionar una agenda dinámica en el entorno de cada proyecto [9].

**Ciclo de Vida de XP**

El ciclo de vida de XP se enfatiza en el carácter interactivo e incremental del desarrollo, en donde una iteración de desarrollo es un período de tiempo en el que se realiza un conjunto de funcionalidades determinadas que en el caso de XP corresponden a un conjunto de historias de usuarios.

**Fase de Exploración**: En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

**Fase de Planeación**: Se priorizan las historias de usuario y se acuerda el alcance del Release[[12]](#footnote-12). Los programadores estiman cuánto esfuerzo requiere cada historia y a partir de allí se define el cronograma. La duración del cronograma del primer Release no excede normalmente dos meses. La fase de planeamiento toma un par de días. Se deben incluir varias iteraciones para lograr un Release. El cronograma fijado en la etapa de planeamiento se realiza a un número de iteraciones, cada una toma de una a cuatro semanas en ejecución. La primera iteración crea un sistema con la arquitectura del sistema completo. Esto es alcanzado seleccionando las historias que harán cumplir la construcción de la estructura para el sistema completo. El cliente decide las historias que se seleccionarán para cada iteración. Las pruebas funcionales creadas por el cliente se ejecutan al final de cada iteración. Al final de la última iteración el sistema está listo para producción.

**Fase de Producción**: Requiere prueba y comprobación extra del funcionamiento del sistema antes de que éste se pueda liberar al cliente. En esta fase, los nuevos cambios pueden todavía ser encontrados y debe tomarse la decisión de si se incluyen o no en el Release actual. Durante esta fase, las iteraciones pueden ser aceleradas de una a tres semanas. Las ideas y las sugerencias pospuestas se documentan para una puesta en práctica posterior por ejemplo en la fase de mantenimiento. Después de que se realice el primer Release productivo para uso del cliente, el proyecto de XP debe mantener el funcionamiento del sistema mientras que realiza nuevas iteraciones.

**Fase de Mantenimiento**: Requiere de un mayor esfuerzo para satisfacer también las tareas del cliente. Así, la velocidad del desarrollo puede desacelerar después de que el sistema esté en la producción. La fase de mantenimiento puede requerir la incorporación de nueva gente y cambiar la estructura del equipo.

**Fase de Muerte**: Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

### HTML, JavaScript y CSS

Originalmente, las páginas HTML sólo incluían información sobre sus contenidos de texto e imágenes. Con el desarrollo del estándar HTML, las páginas empezaron a incluir también información sobre el aspecto de sus contenidos: tipos de letra, colores y márgenes. La posterior aparición de tecnologías como JavaScript, provocaron que las páginas HTML también incluyeran el código de las aplicaciones (llamadas scripts) que se utilizan para crear páginas web dinámicas.

Incluir en una misma página HTML los contenidos, el diseño y la programación complica en exceso su mantenimiento. Normalmente, los contenidos y el diseño de la página web son responsabilidad de diferentes personas, por lo que es conveniente separarlos. CSS es el mecanismo que permite separar los contenidos definidos mediante XHTML y el aspecto que deben presentar esos contenidos.

Una ventaja añadida de la separación de los contenidos y su presentación es que los documentos XHTML creados son más flexibles, ya que se adaptan mejor a las diferentes plataformas: pantallas de ordenador, pantallas de dispositivos móviles, impresoras y dispositivos utilizados por personas discapacitadas.

De esta forma, utilizando exclusivamente XHTML se crean páginas web "feas" pero correctas. Aplicando CSS, se pueden crear páginas "bonitas" a partir de las páginas XHTML correctas.

## MARCO DE LEGAL

### Acuerdos de uso

El software, resultado de la ejecución del proyecto, se regirá bajo los marcos legales fijados por la licencia pública Creative Common (LPCC) que dicta las siguientes condiciones de uso, encontradas más detalladamente en el documento Atribución No Comercial Compartir Igual 2.5 (Colombia) [10]:

* Se posee la libertad de compartir la obra, entendiendo compartir como la capacidad de copiar, distribuir, ejecutar y comunicar públicamente la obra.
* Además puede llevar a cabo obras derivadas de la original.
* Es obligatorio reconocer los créditos de la obra de manera especificada por el autor o el licenciante, pero no de manera que sugiera que tiene el apoyo de este último, o que apoya el uso que hacen a su obra.
* No puede ser utilizada para uso comercial.
* Si se altera o transforma, o genera un derivado a partir de esta obra, solo puede ser distribuida bajo una licencia idéntica a la presente.

### Contrato de licencia de código binario, SUN MICROSYSTEMS

Licencia de uso de software concedidos por Sun Microsystems, para la utilización de Java SE, siempre y cuando sean aceptados los términos de licencia, expuestos en el documento SUN MICROSYSTEMS, INC. CONTRATO DE LICENCIA DE CÓDIGO BINARIO [11].

“SUN MICROSYSTEMS, INC. (EN ADELANTE DENOMINADO “SUN”) LE CONCEDE LA LICENCIA DEL SOFTWARE DEFINIDO A CONTINUACIÓN ÚNICAMENTE CON LA CONDICIÓN DE QUE USTED ACEPTE TODOS LOS TÉRMINOS ESTIPULADOS EN EL PRESENTE CONTRATO DE LICENCIA DE CÓDIGO BINARIO Y TÉRMINOS DE LICENCIA ADICIONALES (EN CONJUNTO DENOMINADOS “CONTRATO”). POR FAVOR, LEA EL CONTRATO DETENIDAMENTE. EL USO DEL SOFTWARE SIGNIFICA QUE HA LEÍDO LAS CONDICIONES Y QUE LAS ACEPTA. SI ACEPTA ESTAS CONDICIONES EN REPRESENTACIÓN DE UNA COMPAÑÍA U OTRA ENTIDAD LEGAL, SIGNIFICA QUE ESTÁ EN POSESIÓN DE LA AUTORIDAD QUE LE PERMITE VINCULAR LA ENTIDAD LEGAL A ESTAS CONDICIONES. SI NO ESTÁ EN POSESIÓN DE DICHA AUTORIDAD O SI NO DESEA QUEDAR VINCULADO A ESTAS CONDICIONES, NO DEBE UTILIZAR EL SOFTWARE EN ESTA INSTALACIÓN NI EN NINGÚN OTRO SOPORTE EN EL QUE SE UBIQUE EL SOFTWARE”.

### Contrato de licencia de mxGraph

MxGraph es una librería Javascript utilizada como parte de la tecnología necesaria para el desarrollo de la herramienta CLASS Modeler. Esta API es privada y tiene derechos de autor, por lo tanto es necesaria la adquisición de una licencia para su uso dentro de instituciones comerciales. Sin embargo en el ámbito académico es posible adquirir una versión de desarrollo, la cual puede ser usada de manera gratuita en proyectos educativos por instituciones del sector. Haciendo la salvedad de que el uso que se le da a la herramienta es con fines netamente educativos y no existe propósito comercial alguno [12].

### GNU General Public Licence

La Licencia Pública General de GNU (GNU GPL, por sus siglas en inglés) es una licencia libre y gratuita con derecho de copia para software y otros tipos de obras.

Las licencias para la mayoría del software y otras obras de índole práctica están diseñadas para privarle de la libertad para distribuir y modificar las obras. Por el contrario, la Licencia Pública General de GNU garantiza la libre distribución y modificación de todas las versiones de un programa, a fin de asegurarle dicha libertad a todos los usuarios. En la Fundación para el Software Libre utilizamos la Licencia Pública General de GNU para la mayoría de nuestro software; también se aplica a cualquier otra obra publicada de esta manera por sus autores. Usted también puede aplicarla a sus programas.

Cuando hablamos de software libre, nos referimos a la libertad, no al precio. Nuestras Licencias Públicas Generales están diseñadas para garantizarle a usted la libertad de distribuir copias de software libre (y cobrar por ellas, si así lo desea), obtener el código fuente, o tener la posibilidad de obtenerlo, modificar el software o utilizar partes del mismo en nuevos programas libres, y saber que puede hacer estas cosas [13].

### Propiedad Intelectual

De acuerdo con lo estimulado en el artículo 156 del acuerdo 065 del 26 de 1996, correspondiente al estatuto estudiante de la Universidad Francisco de Paula Santander, el cual dicta de la siguiente manera: “Los trabajos de grado son propiedad intelectual de la Universidad y su uso estará sujeto a las normas que para tal fin estén vigentes”[[13]](#footnote-13).

# DISEÑO METODOLÓGICO

El diseño metodológico del proyecto está enmarcado en la investigación aplicada y se fundamentará en la ingeniería de software, de ella se toman los conceptos, metodologías y técnicas propias para la construcción de la aplicación.

## TIPO DE INVESTIGACIÓN

Según la problemática planteada, el proyecto pretende dar solución a una necesidad vista dentro del programa de Ingeniería de Sistemas de la UFPS, de tal manera que se englobará el proyecto como una INVESTIGACIÓN APLICADA en términos del desarrollo de software que permita solventar necesidades existentes dentro de la organización.

## FUENTES DE INFORMACIÓN

### Fuentes de información primaria

Teniendo en cuenta que la problemática afecta a la comunidad perteneciente al programa de Ingeniería de Sistemas, se consideran las siguientes fuentes de información primaria:

* Estudiantes que están cursando las asignaturas, “Programación Orientada a Objetos I” y “Programación Orientada a Objetos II”, las cuales se enfocan principalmente en el modelado de aplicaciones. Estos son cursos de 2º y 3º semestre respectivamente según el pensum 1155 [14]. No obstante, a manera de observación indirecta se tomaran otros cursos que puedan usar el producto en construcción.
* Docentes del programa Ingeniería de Sistemas, los cuales son fuente de información de vital importancia, ya que ellos son los que acompañan el proceso de aprendizaje de los estudiantes.

### Fuentes de información secundaria

Estas fuentes hacen referencia a estudios, técnicas, teorías, documentos y toda clase de información existente sobre el tema en particular de modo que se establecen las siguientes:

* Herramientas CASE existentes, comerciales o de código abierto, que permitan el diseño de diagramas de clase. Se tomará en cuenta las herramientas utilizadas actualmente por los estudiantes, de modo que se pueda determinar que funcionalidad o que servicios ofrecen.
* Documento de Especificación del estándar UML para Diagramas de Clase.
* Documentación de proyectos de software online que permitan diseñar diagramas de clase.

Libros enfocados en temas como: Cloud Computing, Ingeniería de Software, Lenguaje Unificado de Modelado, Procesos de desarrollo de software, desarrollo de aplicaciones Web, desarrollo de aplicaciones con J2EE, AJAX.

## RECOLECCIÓN Y ANALISIS DE INFORMACIÓN

El desarrollo del proyecto requiere obtener información general sobre la construcción de diagramas de clase, esto implica conocer a fondo conceptos tales como: Clases, Interfaces, Relaciones, Navegabilidad, entre otros; además de todos los elementos de los cuales dispone UML para el diseño de este tipo de diagramas. Para esto se realizará una investigación profunda sobre el tema, consultado bibliografía existente y diferentes autores.

Adicionalmente se necesita determinar las funcionalidades básicas que poseen las herramientas CASE utilizadas actualmente por los estudiantes de la UFPS, de manera que se pueda conocer qué procedimientos, servicios y características poseen con respecto al diseño de diagramas de clase; esto se hace con el objetivo de determinar los requerimientos de funcionalidad mínimos que debe poseer el software en construir y los servicios que este debe ofrecer. De cierta manera, mediante observación indirecta y a manera de tradición se conocen las principales herramientas CASE utilizadas por estudiantes y docentes de Ingeniería de Sistemas de la UFPS (Enterprise Architect, ArgoUML, StarUML), y considerando que el programa académico dispone de algunas licencias de uso para este tipo de herramientas (como es el caso de Enterprise Architect), se procederá a realizar un análisis concreto de estas herramientas específicamente. De igual manera y como se mencionó anteriormente, este análisis servirá como punto de partida del proyecto y además ayudará a tener un idea clara de la aplicación que se desea llegar a construir.

El procedimiento utilizado para la recolección de información en este caso será de manera literaria, mediante consultas e investigaciones de material bibliográfico, logrando una apropiación de conceptos y técnicas importantes para la construcción del proyecto.

Por último es necesario aclarar que este proyecto no pretende dar respuestas a incógnitas o hipótesis que se tengan sobre algún fenómeno académico o social dentro de la Universidad, sino que se encargara de suplir una necesidad detectada dentro del establecimiento.

Para el análisis de la información se procederá a clasificar los datos según su naturaleza, determinando los aspectos importantes para cada fase del desarrollo y la información necesaria para construir los componentes en cada una de ellas.

El análisis de la información consistirá en evaluar los datos obtenidos de las fuentes mencionadas, documentación de las herramientas CASE existentes, manuales de usuario, análisis del funcionamiento de las aplicaciones utilizadas actualmente, literatura especifica del caso, y en general, fuentes que sean cercanas al tema. Se realizará un proceso de clasificación y agrupamiento de los datos, de manera que se pueda mediante ellos, apoyar la siguiente fase del desarrollo y que sean las directrices de todo el proyecto en general, de modo que mediante estos se puedan tomar decisiones importantes para el desarrollo.

# ANALISIS DE LAS HERRAMIENTAS CASE

Con el objetivo de obtener información importante para la realización del proyecto y con el ánimo de iniciar el proceso de levantamiento de requerimientos se ha realizado un análisis básico de algunas de las herramientas CASE utilizadas dentro del plan de estudios de Ingeniería de Sistemas para realizar modelado UML, de modo que se pueda establecer cuáles son las principales características y servicios que estas ofrecen a los usuarios en cuanto al diseño de *Diagramas de Clase* y que puedan ser incluidos en las funcionalidades del software en construcción.

De modo aleatorio se han seleccionado varias de las herramientas más conocidas y que a manera de observación se ha determinado que son ampliamente utilizadas por estudiantes y docentes del plan de estudios; adicionalmente se ha seleccionado una herramienta similar a la que está en construcción, de modo que podamos conocer qué características ofrece a los usuarios mediante una interfaz web.

Las herramientas seleccionadas son:

* *Enterprise Architect* que es un software comercial conocido a nivel mundial como una gran suite para modelamiento.
* *Netbeans UML* que es una extensión del IDE y por lo tanto, software libre.
* *StarUML* que es software libre y una buena herramienta de modelado UML.
* *GenMyModel* que es una herramienta gratuita de modelado web similar a la herramienta en construcción.

A continuación se describirá cada uno de ellos, resaltando en detalle cada una de sus características:

## GENERALIDADES

### Enterprise Architect

* **Resumen:** Enterprise Architect de Sparx Systems es una herramienta CASE para el diseño y construcción de sistemas de software, para el modelado de procesos de negocios, y para objetivos de modelado más generalizados.

EA es una herramienta de análisis y diseño UML comprensivo, que cubre el desarrollo de software desde la obtención de los requisitos, diseño del modelo, pruebas, cambio de control y mantenimiento para la implementación, con completa trazabilidad. Es una herramienta visual con múltiple-usuario con un gran establecimiento de características ayudando a los analistas, probadores, administradores de proyectos, personal del control de calidad y desarrolladores alrededor del mundo a construir y documentar, software sostenible. La información consignada a continuación fue tomada de la documentación explicita de la herramienta [15].

* **Autor:** Enterprise Architect es el producto insignia de la empresa Sparx Systems, esta es una compañía australiana dedicada a la creación de herramientas para el modelado UML, fundada en 1996 por Geoffrey Sparx y situada en Creswick una pequeña población de Victoria, Australia. La primera versión de Enterprise Architect fue liberada en el año 2000 y fue originalmente diseñada como una herramienta de modelado UML para la versión 1.1 del estándar, de allí en adelante el producto ha evolucionado para soportar otras especificaciones OMG[[14]](#footnote-14) UML como 1.3, 2.0, 2.1, 2.3 y 2.4.1.
* **Tipo de Licencia:** Esta es una herramienta comercial que ofrece diferentes ediciones en base a las necesidades del usuario o las organizaciones que usan la aplicación. Durante el tiempo de vida de la aplicación se han liberado alrededor de 20 versiones, desde la versión 1.0 hasta la más reciente 9.3 liberada en Agosto de 2012, cada una con mejoras y características importantes según los cambios que ha sufrido el estándar UML y la adopción de algunas tecnologías.

Enterprise Architect se encuentra disponible en diferentes ediciones, según las necesidades del usuario o de las organizaciones que usan la aplicación, de modo que se pueden adquirir diferentes tipos de licencias para diferentes usos.

* **Características Principales:** Enterprise Architect posee un gran abanico de características, todas encaminadas al desarrollo y mantenimiento de proyectos de software, aunque también brinda soporte para otro tipo de proyectos.

La información aquí consignada fue tomada de la documentación explicita de Enterprise Architect, disponible en [15], además se usó parte de la información del “Contenido de Ayuda” que viene incluido en la versión de prueba del software.

| **Característica** | **Descripción** |
| --- | --- |
| Plataforma | Diseñado para ser usado bajo sistemas operativos Windows, sin embargo puede ser instalado y utilizado mediante emuladores como CrossOver en sistemas Linux o MAC. CrossOver es una herramienta propietaria basada en una tecnología open source llamada Wine, el cual es una API Win32 re-desarrollada para sistemas operativos basados en Unix como Mac OS X o Linux. |
| Versión de UML | UML 2.4.1 Liberada en agosto de 2011. |
| Diagramas UML | Soporte para todos los tipos de diagramas UML tanto estructurales como comportamentales. Algunos diagramas adicionales no UML: Análisis, Requerimientos, Bases de Datos, Mantenimiento, Interfaces de Usuario, Interacción y modelos del negocio. |
| Tecnologías MDG | Soporta tecnologías como ICONIX, ArcGIS, BPMN, SMOF, SoaML, SysML. |
| Fases del desarrollo | Modelamiento del Negocio, Ingeniería de Requerimientos, Análisis, Diseño, Implementación, Pruebas, Mantenimiento, Gestión de la Calidad. |
| Estándar XMI | Soporte completo del estándar en su versión 2.1, permite exportar e importar proyectos en archivos .XMI de modo que puedan ser llevados a otras herramientas de modelado. |
| Proyectos y equipos de trabajo | Permite generar repositorios de servidor DBMS, de modo que se pueda trabajar de manera concurrente en proyectos de gran envergadura y con gran cantidad de usuarios. Adicionalmente permite integración con repositorios de control de versiones como Subversion, CVS y SCC. |
| Importación / Exportación | Exportación: Existen varias formas de exportar proyectos, diagramas o cualquier otro tipo de artefacto creado dentro del software, archivos XML, XMI, Código fuente, imágenes planas de los diagramas en formato PNG, JPG, GIF, entre otros. Importación: Desde XML, XMI, archivos CSV, desde código fuente en varios lenguajes, desde archivos binarios como JARs o ensamblados .NET. |
| Ingeniería Directa e Inversa | Enterprise Architect soporta la generación directa e ingeniería reversa de código fuente para muchos lenguajes populares incluyendo C++, C#, Java, Delphi, VB.Net, Visual Basic, ActionScript, Python y PHP. Con un editor de código fuente “resaltador de sintaxis” integrado. |
| Plantillas de Código | Las plantillas de generación de código le permiten personalizar el código fuente generado de acuerdo a las especificaciones de su compañía. |
| Edición de Código | Enterprise Architect le permite rápidamente navegar y explorar su código fuente del modelo en el mismo ambiente. |
| Documentación generada | Soporte completo de WYSIWYG manejador de plantillas para generar archivos RTF. |
| Generación de Imágenes | Permite generar imágenes planas en varios formatos como PNG, JPG, BMP y GIF. Estos son útiles a la hora de documentar el proyecto. |
| Distribución Automática de Elementos | Esta característica se refiere a la capacidad de la herramienta de organizar los elementos de un diagrama de manera automática, siguiendo una distribución determinada. |
| Soporte de Versiones | Soporte para servidores Subversion SVN. |

Tabla . Características generales de Enterprise Architect

### Netbeans UML

* **Resumen:** Netbeans UML es un componente que se agrega como parte del conjunto de características que ofrece el entorno de desarrollo integrado IDE. Básicamente se trata de un ambiente para el diseño de diagramas UML que incluye características de ingeniería directa e inversa. Esta herramienta permite la creación de 8 tipos de diagramas UML: Actividad, Clase, Secuencia, Estado y Casos de uso; así como algunos diagramas de dependencia para un clasificador en el diagrama: Generalizaciones, Asociaciones, Implementaciones y Operaciones.

Según sus creadores, con el uso de Netbeans UML, los diseñadores se enfocan en el diseño de la aplicación, y los desarrolladores en el código. Primero, diseñadores y analistas diseñan aplicaciones usando UML, un lenguaje de modelado estándar, los desarrolladores generan el código fuente desde el modelo UML. Inclusive permite sincronización en ambos sentidos, es posible actualizar el modelo con los cambios hechos en el código fuente [16].

* **Autor:** La extensión fue desarrollada inicialmente de manera personal por el desarrollador Sergey B. Petrov, quien implementó la herramienta para la versión 5.5 de Netbeans IDE. La herramienta entró en una etapa de descontinuación, debido a que no se desarrollaron más actualizaciones ni se siguió dando soporte al módulo para nuevas versiones del IDE, tanto así que las versiones posteriores a Netbeans 5.5 (6+, 7+), no permitían instalar el modulo desde el centro de actualizaciones, lo cual obliga a los desarrolladores a instalarlo, bajo su responsabilidad, mediante el uso manual de la URL de descarga. Actualmente la herramienta fue acogida por una comunidad quienes han trabajado para permitir la instalación en versiones nuevas del IDE como la 7.3 y 7.4.
* **Tipo de Licencia:** La herramienta esta licenciada bajo la norma CDDL *Common Development and Distribution License* (Licencia Común de Desarrollo y Distribución), la cual es una licencia de código abierto y libre, producida por Sun Microsystems, basada en la Mozilla Public License MPL versión 1.1.
* **Características Principales**

| **Característica** | **Descripción** |
| --- | --- |
| Diagramas UML | Actividad, Clase, Secuencia, Estado y Casos de Uso. |
| Ingeniería directa | Permite la generación de código directamente desde el diagrama UML, generando un proyecto dentro del mismo IDE y copiando las clases, interfaces y enumeraciones dentro de dicho proyecto, |
| Ingeniería inversa | De igual manera la herramienta permite la sincronización y generación del modelo a partir de código fuente existente. Posee un asistente para ayudar al usuario en esta labor. |
| Generación de Javadoc | Permite generar un reporte HTML con la documentación en Javadoc de las clases y demás elementos modelados. |
| Patrones de diseño | Permite crear de manera asistida diagramas UML basados en patrones de diseño conocidos, tales como GoF [17] y algunas tecnologías como EJB [18]. Posee un asistente para guiar al usuario a través del proceso. |
| Distribución Automática | Permite organizar automáticamente los elementos del diagrama en una distribución jerárquica, es decir. |

Tabla . Características generales de Netbeans UML

Debido a que la documentación de la herramienta es muy escasa, solamente fue posible determinar las características a grandes rasgos, sin entran en detalles específicos. La mayoría de estas características fueron detectadas a manera de observación y mediante el uso de la herramienta. No existe una documentación formal, y cabe mencionar que la herramienta actualmente se encuentra obsoleta ya que no han surgido nuevos desarrollos de la misma. Inclusive las nuevas versiones de Netbeans (7+) no permiten la instalación del complemento desde el asistente, es necesario agregar manualmente la URL de descarga para poder instalarlo, además que consta de varios errores debido a incompatibilidad con el mismo IDE.

### StarUML

* **Resumen:** StarUML™ es una plataforma de modelado de software que soporta UML (Unified Modeling Language). Se basa en la versión 1.4 de UML versión y proporciona 11 tipos diferentes de diagrama, y acepta la notación UML 2.0. Apoya activamente el enfoque de MDA (Model Driven Architecture), haciendo hincapié en el concepto de perfil UML. StarUML destaca en la personalización del entorno del usuario y tiene una alta extensibilidad en su funcionalidad.
* **Autor:** Inicialmente el proyecto fue liderado por la empresa Plastic Software Inc, quienes en el año 1997 liberaron la primera versión llamada “Plastic” o “Agora Plastic”, luego de algunos años fue rediseñada completamente y pasó a conocerse como StarUML. Desde el 2005 fue tomado por una comunidad de desarrolladores y hasta el momento no se han conocido nuevas versiones del software desde la liberación de la versión 5 en el año 2005.

StarUML fue programado en lenguaje Delphi, la cual fue una de las razones por las cuales el proyecto fue abandonado por un largo tiempo. Sin embargo, la comunidad ha tratado de revivirlo al intentar hacer una migración al lenguaje Java e incorporándolo al IDE Eclipse.

* **Tipo de Licencia:** Es una herramienta UML open source, licenciada bajo una versión modificada del acuerdo GNU GPL.
* **Características Principales:** La siguiente tabla muestra las características generales de esta herramienta de modelado. Esta información fue tomada explícitamente de la guía de usuario de StarUML [19].

| **Característica** | **Descripción** |
| --- | --- |
| Estándar UML | A la notación UML 2.0, sobre la base de un meta modelo robusto. |
| Formato de modelo Open Source | A diferencia de muchos productos existentes que gestionan sus propios modelos de formato heredado ineficiente, StarUML gestiona todos los archivos en el formato XML estándar. Códigos escritos en estructuras fáciles de leer y sus formatos se pueden modificar convenientemente usando un analizador XML. Dado que XML es un estándar mundial, este es sin duda una gran ventaja. |
| Soporte real de MDA | StarUML soporta UML Profile esto maximiza la extensibilidad de UML, haciendo el modelado de aplicaciones posibles, incluso en áreas como las finanzas, la defensa, el comercio electrónico, los seguros y la aeronáutica. Verdaderamente modelos independientes de plataforma (PIM) pueden ser creados, y modelo específicos de la plataforma (PSM) y códigos ejecutables pueden ser generados automaticamente. |
| Aplicación de metodologías y plataformas | StarUML manipula el concepto de enfoque, creando de ambientes que se adaptan a cualquier metodología o proceso. No sólo los modelos de Frameworks de aplicación para plataformas como. NET y J2EE, sino también las estructuras básicas de los modelos de software (por ejemplo, 4+1 View Model, etc.) se puede definir fácilmente. |
| Extensibilidad | Todas las funciones de las herramientas de StarUML están automatizadas acorde a Microsoft COM. Cualquier lenguaje que soporte COM (Visual Basic Script, JavaScript, VB, Delphi, C++, C#, VB.NET, Python, etc.) puede ser usado para manejar StarUML y desarrollar elementos integrados Add-In. |
| Función de verificación de modelos de software | Los usuarios pueden cometer muchos errores durante el modelamiento de software. Estos errores pueden ser muy costosos si permanecen hasta la etapa final de codificación. Con el objetivo de prevenir este problema, StarUML automáticamente verifica los modelos de software desarrollados por el usuario, facilitando descubrir de manera temprana errores, permitiendo software más completo y con menos faltas. |
| Add-Ins útiles. | StarUML incluye algunos Add-Ins útiles con varias funcionalidades. Estos generan código fuente en varios lenguajes de programación y convierten código fuente en modelos, importan archivos de Rational Rose, intercambian información de modelos con otras herramientas usando XMI, y soportan diseño de patrones. Estos Add-Ins proporcionan reusabilidad adicional, productividad, flexibilidad e interoperabilidad para la información de modelado. |
| Ingeniería Directa / Ingeniería Inversa | Para la generación de código StarUML dispone de un asistente (Wizard) para el proceso. Los archivos de código generado por defecto pueden ser lenguaje C++, C# o Java, sin embargo es posible añadir otros lenguajes instalando los Add-Ins necesarios para este propósito. |
| Formatos de Textos | La herramienta permite definir formatos de textos para los elementos, estos se aplican a los nombres de las clases, o nombres de los atributos y métodos. Adicionalmente también permite establecer el tipo de fuente usada, los colores de las líneas y algunas otras configuraciones. |
| Importación / Exportación | La herramienta dispone de varios mecanismos de importación y exportación. En el caso de importación permite realizar ingeniería inversa de otras herramientas como Rational Rose o desde archivos XMI que contienen la información del diagrama o modelo |

Tabla . Características principales de StarUML

### GenMyModel

* **Resumen:** GenMyModel es una herramienta de modelado UML basada en web que permite soporte para colaboración de equipos. Posee la ventaja de ser una herramienta multiplataforma debido a su naturaleza orientada al internet. La herramienta está enfocada en el tipo de servicio SaaS impulsado por el Cloud Computing.
* **Autor:** Axellience es la empresa propietaria del software GenMyModel, como se mencionó en el Marco de Antecedentes, esta empresa es una Start-Up francesa fundada en el año 2012. Los fundadores de la empresa son: *Alexis Muller*, quien tiene un doctorado en ciencias de la computación en el campo de ingeniería de software especializado en ingeniería dirigida por modelos; *Thomas Legrand*, es ingeniero en computación experto en métodos y herramientas de desarrollo, actualmente coordina y soporta el equipo de desarrollo; y por último *Stephane Deveaux*, quien es magister en economía, y durante 10 años ha estado involucrado en la gestión de ventas en ambientes B2B.
* **Tipo de Licencia:** Es una herramienta gratuita que puede ser accedida y utilizada de manera libre, sin embargo no es de código abierto.
* **Características Principales**

| **Característica** | **Descripción** |
| --- | --- |
| Diagramas UML | Permite crear diagramas de clase, casos de uso y de actividad. |
| Arquitectura | La herramienta permite modelos que representan la arquitectura de una aplicación. Los modelos cumplen con la especificación UML 2.0 y son compatibles con otras herramientas UML 2. |
| No Instalación | Solamente es necesario disponer de un navegador web soportado: Chrome, FF, IE > 9, Opera o Safari. |
| Recursos compartidos | Después de crear un modelo UML, los usuarios pueden compartirlo con sus colaboradores. |
| Trabajo en equipo | GenMyModel permite a los desarrolladores de software alrededor del mundo trabajar sobre el mismo modelo mediante la edición colaborativa. Para esto es necesario compartir el modelo con otros usuarios y en cuestión de segundos se puede trabajar de manera conjunta, adicionalmente estos pueden chatear para realizar diseños en tiempo real. |
| Generación de código online | Provee generadores de código en lenguaje Java, SQL y Spring. Por otro lado permite exportar código a plataformas online como Github[[15]](#footnote-15). |
| Generar imágenes | Exporta imágenes en formato SVG y JPEG. |
| Reportes | Permite generar reportes en archivos PDF del modelo diseñado. |
| Comunidad | La herramienta gira en torno a una comunidad global que utiliza sus servicios, proponiendo nuevas características y sirviendo de testers para verificar su funcionamiento. |

Tabla . Características generales de GenMyModel

## DISEÑO DE DIAGRAMAS DE CLASE

A continuación se mencionan las características principales descritas anteriormente y se realiza una comparación de las herramientas basada en cada una de ellas. Esto nos permitirá conocer de manera más detallada el funcionamiento externo de cada herramienta y la manera en la cual el usuario interactúa con estas aplicaciones; lo cual es importante para el posterior diseño de interfaces de usuario y la implementación de las funcionalidades propias de la herramienta.

### Características Básicas

* **Interfaz de Usuario:** La interfaz de usuario es un aspecto muy importante para este tipo de aplicaciones, es por ello que decidimos analizar este aspecto comparando la distribución de la pantalla principal del diseñador en cada herramienta; a continuación se muestra una imagen de la interfaz principal de las herramientas CASE analizadas.

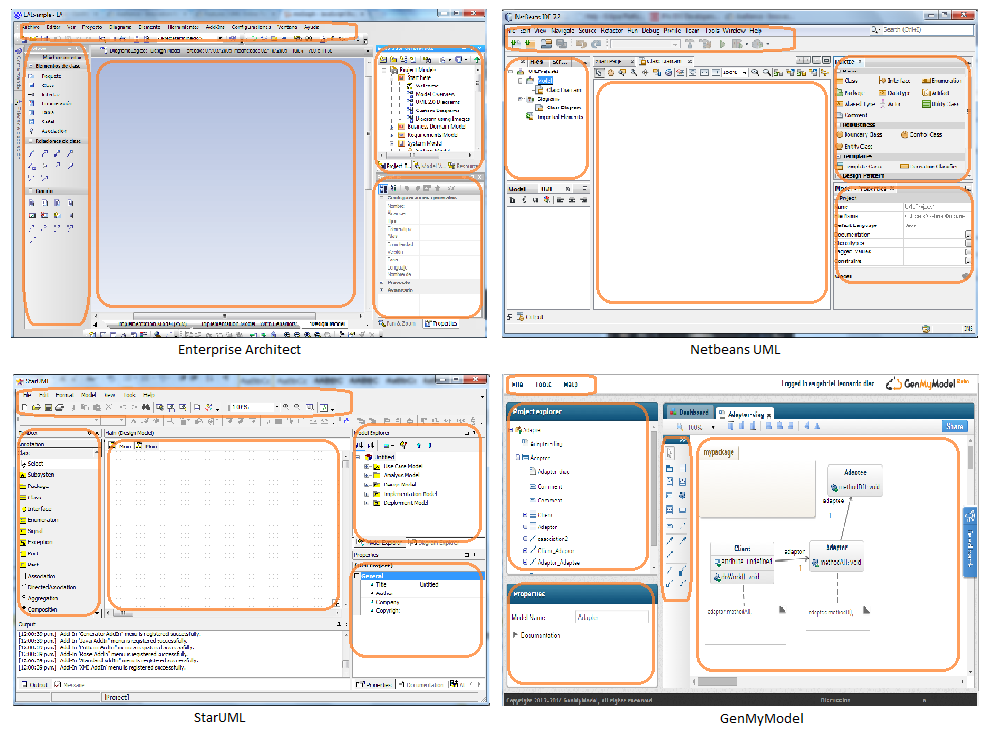


Figura . Interfaz de usuario principal de las herramientas CASE

Como se puede observar en la figura anterior, todas las herramientas CASE poseen una gran similitud en los controles GUI que ofrecen al usuario. La pantalla principal consta básicamente de las mismas secciones o paneles: Barra de menú y herramientas, Paleta de componentes, Panel del Árbol del Proyecto, Panel de Propiedades y el Área de trabajo en la parte central. Quizás la única diferencia es la distribución de los paneles mencionados anteriormente, debido a que cada herramienta los ubica en su posición preferida. Sin embargo la mayoría de ellos son ajustables, y en algunos casos de pueden ocultar para permitirle al usuario un área de trabajo mucho más grande.

* **Panel de Propiedades:** El panel de propiedades permite editar de manera rápida la información del elemento seleccionado en el área de trabajo. Este panel se compone de una tabla en donde cada fila representa una propiedad del elemento, tales como el nombre y visibilidad de una *Clase* o *Interfaz*. Adicionalmente las propiedades pueden estar agrupadas por secciones; por ejemplo, es común tener una sección con las propiedades generales, y otras con propiedades avanzadas o especificas del elemento. Estas propiedades corresponden a las definidas por el metamodelo UML, que a su vez establece los valores válidos para cada propiedad. Cabe mencionar que cuando no hay ningún elemento seleccionado en el área de trabajo, el panel de propiedades se deshabilita, de tal manera que el usuario sepa que no puede realizar ninguna operación allí.
* **Paleta de Componentes:** Este panel contiene los componentes o elementos que se pueden agregar al diagrama que se encuentra en edición. Los elementos típicos de diagramas de clase que las herramientas ofrecen son: *Clase*, *Interfaz*, *Paquete*, *Enumeración*, *Tipo de Dato*, *Comentario*; adicionalmente, se encuentran los diferentes tipos de relaciones: *Asociación*, *Composición*, *Agregación*, *Generalización*, *Realización*, *Dependencia*. Los elementos son añadidos al diagrama desde la paleta de componentes, comúnmente se puede arrastrar y soltarlos en el área de trabajo, o seleccionándolos y luego haciendo clic en un espacio libre.
* **Árbol del Proyecto:** Este panel muestra un árbol que contiene todos los elementos agregados al proyecto que se está trabajando. Es común que herramientas CASE suite, que permiten realizar varios tipos de diagramas y que abarcan todas las fases del proceso de desarrollo, contengan este control, ya que permite clasificar los artefactos diseñados y facilita su ubicación.

* **Edición de Diagramas:** La edición de diagramas comprende acciones como crear, editar, copiar, eliminar o guardar diagramas. Comúnmente para aplicaciones basadas en Desktop, la herramienta crea un archivo de proyecto que alberga todos los diagramas diseñados dentro del proyecto. Es posible realizar la edición de varios diagramas, inclusive de diferentes tipos, al mismo tiempo; mediante el uso de pestañas que representan cada una un diagrama abierto.
* **Edición de Elementos:** Los elementos se pueden añadir desde la paleta hasta el área de trabajo; sin embargo no es posible tener dos veces el mismo elemento en el mismo diagrama. Los elementos pueden ser removidos del diagrama, pero no necesariamente son borrados del proyecto.
* **Edición de Atributos:** La edición de atributos se realiza de manera común entre los modeladores mediante un dialogo emergente que contiene una tabla o listado de todos los atributos de un clasificador[[16]](#footnote-16). Dentro de dicho dialogo, se puede crear, editar, eliminar y en algunos casos reordenas los atributos. Un aspecto importante que se encontró a la hora de agregar atributos a una clase, fue la selección de colecciones como tipo de dato del mismo, ya que por lo general las herramientas no soportan este tipo de funciones, debido a que intentan funcionar de manera estándar para cualquier lenguaje de programación. En el caso de las enumeraciones, este mismo panel sirve para editar los *Literales* de dicho elemento.
* **Edición de Operaciones:** Al igual que con la edición de atributos, para la edición de operaciones se dispone de un dialogo emergente que contiene la tabla de operaciones creadas para un clasificador. Adicionalmente a la edición de la información básica de la operación, el dialogo también permite gestionar los parámetros de dicha operación. Cada operación debe disponer de información básica, tal como: nombre, visibilidad, tipo de retorno y parámetros; es posible también configurar algunos modificadores que determinan el funcionamiento de la operación: estático, final, y en algunos casos determinar el nivel de concurrencia que este soporta.
* **Edición de Relaciones:**
* **Generación de Código**
* **Generación de Imágenes**
* **Edición General:** Las herramientas de edición hacen referencia a funcionalidades comunes de cualquier editor, tales como: Copiar, Cortar, Pegar, Selección, Eliminar, Zoom, Deshacer, Rehacer, Orden Z, Formatos de textos.

### Características Avanzadas

* **Plantillas de Código fuente**
* **Distribución automática de elementos**
* **Trabajo en Equipos**
* **Soporte de Versiones**
* **Importación/Exportación de XMI**
* **Documentación Auto Generada**

* **Vista en Miniatura**
* **Multiplataforma**

## MATRIZ DE CARACTERIZACIÓN

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Característica** | | **Enterprise Architect** | **Netbeans UML** | **StarUML** | **GenMyModel** |
| Edición | Portapapeles (Copiar, Cortar, Pegar) | X | X | X | X |
| Arrastrar y Soltar (D&D) | X | X | X | X |
| Formato de Textos | X | X |  |  |
| Ordenamiento (Traer al frente, enviar atrás) | X | X | X | X |
| Renombrado inmediato | X | X | X | X |
| Alinear elementos | X | X | X | X |
| Zoom | X | X | X | X |
| Vista Miniatura | X |  |  |  |
| Atajos de teclado | X | X | X | X |
| Deshacer-Rehacer | X | X |  | X |
| Generación de Código Fuente | | X | X | X | X |
| Ingeniería Inversa | | X | X | X |  |
| Importación/Exportación XMI | | X | X | X | X |
| Generar imágenes a partir del diagrama | | X | X | X | X |
| Sincronización con código fuente | | X |  |  |  |
| Edición de código fuente | | X |  |  |  |
| Control de versiones | | X |  |  |  |
| Generación de documentación a partir del diagrama | | X | X |  |  |
| Distribución de elementos automáticamente | | X | X |  |  |
| Plantillas de código para lenguajes adicionales. | | X | X |  |  |

Tabla . Matriz de características de herramientas CASE para diagramas de clase

## TABULACIÓN Y ANALISIS

### Tabulación

### Análisis

# METODOLOGÍA DE DESARROLLO

La metodología de desarrollo seleccionada para el proyecto fue *eXtreme Programming* (XP), debido a su naturaleza y al tamaño del equipo de trabajo, se puede adaptar fácilmente al desarrollo del proyecto; considerando que brinda flexibilidad y la posibilidad de adaptación a los cambios que pueden surgir en el transcurso del proyecto.

## EXPLORACIÓN

### Roles del Proyecto

El desarrollo del proyecto será llevado a cabo por un grupo pequeño de personas, sin embargo cada una tiene una labor específica e importante para la correcta ejecución del mismo. A continuación se describen los roles del proyecto y su participación en el proyecto:

* **Desarrollador**: Gabriel Leonardo Díaz Cárdenas, estudiante de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander.
* **Cliente**: En un proceso de desarrollo ágil es de vital importancia para el desarrollo del proyecto, sin embargo en nuestro caso no tenemos un ente o persona específica a la cual se le entregará la aplicación, en su lugar tenemos un grupo general de personas a las cuales irá dirigido el proyecto:

“Estudiantes del Plan de Estudios de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, en especial, aquellos que cursan materias como Programación Orientada a Objetos I y Programación Orientada a Objetos II. Adicionalmente los Docentes que hacen parte de estos cursos del pensum académico”.

* **Director de Proyecto**: Marco Antonio Adarme Jaimes, Msc. en Computación. Director general del proyecto quien estará controlando su ejecución y orientando el proceso de implementación, además quien estará verificando que se cumplan los objetivos planteados al inicio del proyecto.
* **Asegurador de la Calidad**: Las labores de verificación y pruebas del funcionamiento de la aplicación estarán a cargo del Msc. Margo Antonio Adarme Jaimes, a quien se le entregaran las liberaciones de cada iteración (mencionadas más adelante) y quien dará el visto de aprobación.
* **Consultor**: Este rol es asumido por personas externas, expertos en algunos de los temas que vamos a tratar durante la ejecución del proyecto. A continuación se mencionan las personas que ejercen este rol en el proyecto:
  + **Gaudenz Alder**: Cofundador de la empresa JGraph Ltd, creadora de la librería JavaScript mxGraph utilizada para el funcionamiento de este proyecto. Gaudenz inició JGraph como su proyecto de tesis para el instituto tecnológico de Suiza, Zurich en el 2000.
  + **PrimeFaces Community**: Este agente externo está compuesto por un grupo de desarrolladores alrededor del mundo que se encargan de resolver problemas relacionados con el uso del Framework JSF PrimeFaces, y permiten a usuarios novatos aprender fácilmente el uso de la herramienta.

### Actores del Sistema

En UML se define un *actor* como aquella persona, proceso o cosa que interactúa con un sistema, subsistema o clase [7]. Para nuestro proceso de desarrollo es de vital importancia determinar cuáles son los actores de nuestro sistema, de modo que se pueda tener una idea clara de las necesidades conjuntas y particulares de cada uno de ellos.

Cabe mencionar que dentro de la aplicación no se dispone de un usuario administrador, ya que no existen tareas o configuraciones especiales que se tengan que hacer de manera controlada. Cada usuario es capaz de interactuar con la aplicación de la misma manera y todos ellos disponen de los mismos privilegios. Esto evita la necesidad de tener a una persona realizando configuraciones en la aplicación y la libera de este tipo de dependencia.

|  |  |  |
| --- | --- | --- |
| Visitante.bmp | **VISITANTE** | |
| Representa cualquier persona que llega a la aplicación en búsqueda de información, ya sea de manera accidental o voluntaria. Este actor no tiene una representación real dentro de la aplicación y solamente es manejado de manera conceptual, en efecto cualquier persona inicia siendo un visitante de la aplicación. | |
| **Clase** | --- |
| **Súper Clase** |  |
| **Interacción con la aplicación** | | |
| * Búsqueda de Información o curiosidad. * Registrarse en el sistema. | | |

Tabla . Especificación del actor *Visitante*

|  |  |  |
| --- | --- | --- |
| Visitante.bmp | **USUARIO** | |
| Representa una persona que puede iniciar una sesión para diseñar un diagrama dentro de la aplicación. Este es un actor abstracto y necesita ser redefinido en una representación concreta. Dentro de la aplicación Usuario es una interfaz implementada por dos clases. | |
| **Clase** | *User* |
| **Súper Clase** |  |
| **Interacción con la aplicación** | | |
| * Inicio de Sesión * Cerrar Sesión * Diseñar diagrama (con todas sus características) * Generar Código * Generar Imagen | | |

Tabla . Especificación del actor *Usuario*

|  |  |  |
| --- | --- | --- |
| Visitante.bmp | **INVITADO** | |
| Un invitado en la aplicación es la representación de un usuario que utiliza la opción de *Ver Demostración* e inicia sesión dentro de la aplicación sin registro previo. Este actor posee una representación real dentro de la aplicación, sin embargo sus privilegios son limitados y no tiene acceso a algunas funcionalidades. | |
| **Clase** | *Guest* |
| **Súper Clase** | *User* |
| **Interacción con la aplicación** | | |
| * Todas las interacciones heredadas * Ver demostración: Esta opción puede ser usada por cualquier visitante, sin embargo en el momento de ingresar a la aplicación dicho visitante se convierte en un usuario invitado. | | |

Tabla . Especificación del actor *Invitado*

|  |  |  |
| --- | --- | --- |
| Visitante.bmp | **DIAGRAMADOR** | |
| Representa una persona registrada en la aplicación mediante el formulario designado para tal labor. A esta persona se le asigna una cuenta usuario mediante su dirección de correo electrónico, y por medio de la cual ingresa a la aplicación de manera controlada para hacer uso de todas las funcionalidades que están a su disposición. | |
| **Clase** | *Diagrammer* |
| **Súper Clase** | *User* |
| **Interacción con la aplicación** | | |
| * Todas las interacciones heredadas * Ver tablero de control * Gestionar diagramas (Crear, Editar, Borrar, Copiar) * Compartir diagramas * Cambiar Privilegio * Quitar Privilegio * Gestionar Perfil * Cambiar Contraseña * Recuperar contraseña * Edición concurrente de diagramas | | |

Tabla . Especificación del actor *Diagramador*



Diagrama . Diagrama de actores de la aplicación CLASS Modeler

### Historias de Usuario

A continuación se describen las historias de usuario recolectadas para la implementación del proyecto. El siguiente formato [17] dispone de la información correspondiente a cada historia de usuario, incluyendo la prioridad, la estimación de tiempo y la iteración a la cual fue asignada (las iteraciones son mencionadas en la sección de *Planificación*)

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H1 | **Nombre de historia:** Crear cuenta de diagramador | |
| **Usuario:** Visitante | | **Iteración Asignada:**  1 |
| **Prioridad:** Media | | **Estimación:** 10 días |
| **Descripción:**  Un visitante puede crear una cuenta de diagramador para iniciar sesión en el sistema. Para ello, dispone de un formulario de registro donde se captura información básica del visitante, como los nombres, apellidos, correo electrónico, género y contraseña. | | |
| **Reglas u Observaciones:**   * La dirección de correo electrónico de cada diagramador debe ser única en el sistema, de modo que la aplicación debe controlar y verificar que se cumpla esta condición. El correo electrónico es usado como identificador de la cuenta. * La contraseña ingresada debe tener mínimo 5 caracteres de longitud y máximo 20. Puede incluir cualquier combinación de letras y números. * El correo electrónico es único para todo el sistema, por lo tanto un diagramador no puede registrarse con un correo electrónico ya usado por otra persona. * El género se usa para asignar una imagen por defecto al diagramador una vez crea su cuenta, esta imagen puede ser cambiada a gusto de la persona en cualquier momento. * Una vez creada la cuenta de diagramador, esta permanece INACTIVA hasta que se realice la activación respectiva mediante el correo de confirmación. Este correo se envía a la dirección ingresada al momento de crear la cuenta. * Al momento de crear la cuenta de diagramador el sistema debe generar un código de verificación encriptado, el cual se envía junto con el correo de activación y sirve para validar los datos al momento de ejecutar el proceso. * El código de activación debe ser único por cada solicitud. * El diagramador no puede iniciar sesión en el sistema hasta que active su cuenta. | | |

Tabla . Formato de historia de usuario H1

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H2 | **Nombre de historia:** Activar cuenta de diagramador | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 1 |
| **Prioridad:** Media | | **Estimación:** 5 días |
| **Descripción:**  Este es un requerimiento para poder usar la aplicación una vez se creada la cuenta de diagramador. La activación de la cuenta se hace mediante un link que se envía a la dirección de correo que el visitante ingresó al momento de su registro, este link contiene la información necesaria para validar los datos y un código único encriptado para brindar seguridad. | | |
| **Reglas u Observaciones:**   * El código de activación tiene un periodo de vigencia de 2 días, por lo tanto si la persona no realiza la activación dentro de ese tiempo el código debe caducar y se debe generar uno nuevo. * El código de activación es válido solamente una vez. Por lo tanto una vez utilizado ya se será válido nunca más. * Las cuentas de diagramador pueden tener 3 estados diferentes: INACTIVA el cual se obtiene cuando recién se crea la cuenta de diagramador, ACTIVADA que es el estado normal de una cuenta vigente y DESACTIVADA el cual indica que la cuenta esta temporalmente invalida y no puede ser usada, esto último hecho por voluntad del diagramador. * La cuenta podrá ser activada únicamente si esta existe y el código de verificación concuerda con el guardado en base de datos y además si el código de verificación no ha caducado. * Cuando el código esta vencido la aplicación debe generar uno nuevo y enviarlo automáticamente a la dirección de correo del diagramador, adicionalmente debe informarle que se ha generado un nuevo código. * Una vez realizada la activación de la cuenta, el usuario puede ingresar normalmente a la aplicación por medio de la opción iniciar sesión. | | |

Tabla . Formato de historia de usuario H2

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H3 | **Nombre de historia:** Reasignar contraseña | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 7 |
| **Prioridad:** Baja | | **Estimación:** 7 días |
| **Descripción:**  Un diagramador puede reasignar una nueva contraseña a su cuenta en caso de haber olvidado la anterior. El proceso se realiza por medio de la dirección de correo del diagramador, a donde se le envía un email de confirmación con un link para hacer la reasignación. El link contiene la información necesaria para validar la cuenta y lleva al diagramador a un formulario donde puede asignar una nueva contraseña para su cuenta. | | |
| **Reglas u Observaciones:**   * El diagramador puede solicitar la reasignación de su contraseña mediante la opción *¿Olvidó su contraseña?* en el formulario de inicio de sesión. * Esta opción redirige al diagramador a un formulario donde debe ingresar su dirección de correo. * El sistema valida la existencia de la cuenta y procede a enviar el email de confirmación a la dirección ingresada. El diagramador debe abrir su cuenta de correo y hacer clic en el link enviado. * Las reglas de validación del código de verificación deben ser las mismas que para H2. * Una vez validado el código de verificación se le debe mostrar al diagramador un formulario donde debe ingresar la nueva contraseña y la confirmación, las reglas para la contraseña son las mismas descritas en H1. * Si la validación del código no es satisfactoria se debe mostrar un mensaje de error al usuario. * Se solicita la nueva contraseña y una confirmación, estas dos deben coincidir. * La contraseña anterior no puede ser recuperada y no es necesario validar que la nueva contraseña sea diferente a la anterior. * Una vez los datos sean validados se procede a cambiar la contraseña actual por la nueva. | | |

Tabla . Formato de historia de usuario H5

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H4 | **Nombre de historia:** Iniciar y cerrar sesión | |
| **Usuario:** Diagramador/Invitado | | **Iteración Asignada:** 1 |
| **Prioridad:** Media | | **Estimación:** 9 días |
| **Descripción:**  Funcionalidad mediante la cual un usuario ingresa a la aplicación. Esta operación puede ser llevada a cabo de dos maneras diferentes, un usuario registrado (Diagramador) que ingresa a la aplicación con su dirección de correo o un usuario invitado que ingresa a la aplicación sin registro previo (Modo Demostración).  Para los usuarios registrados se dispone de un formulario de ingreso, donde se solicita la dirección de correo electrónico y la contraseña. | | |
| **Reglas u Observaciones:**   * Se deben realizar validaciones básicas, como verificar la existencia de la cuenta, el estado de la misma y la contraseña ingresada. * Para poder iniciar sesión es necesario que la cuenta tenga el estado ACTIVADA, por lo tanto el usuario debe activar su cuenta antes de ingresar (únicamente en el caso de los diagramadores). * Si la cuenta está INACTIVA el usuario no podrá iniciar sesión. * Los usuarios tipo invitado inician sesión para entrar en modo demostración a la aplicación, el cual se realiza sin necesidad de registro previo. * Una vez se inicia sesión, se redirige al usuario a una página de bienvenida, la cual depende del tipo de usuario. Para diagramadores, se redirige al *Tablero de Control*, y en el caso de invitados se redirige directamente al *Diseñador*. * Considerar temas de seguridad, para proteger el acceso no autorizado a páginas de la aplicación que requieren el inicio de sesión. * La opción cerrar sesión está disponible en el menú de opciones del usuario, al cual a su vez se le añadirán algunas otras opciones en las historias de usuario H5 y H6. * Considerar el tiempo de expiración de la sesión. * Cuando se cierra la sesión se termina la edición del diagrama en el caso de estar ubicado en el diagramador. * Considerar el caso de que un usuario cierre el navegador sin haber terminado la sesión de manera manual. | | |

Tabla . Formato de historia de usuario H4

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H5 | **Nombre de historia:** Editar perfil de diagramador | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 7 |
| **Prioridad:** Baja | | **Estimación:** 6 días |
| **Descripción:**  Esta opción permite cambiar la información de registro del diagramador, tal como nombres, apellidos, fecha de nacimiento, sexo y avatar (imagen de identificación). No se puede cambiar la dirección de correo electrónico del usuario. | | |
| **Reglas u Observaciones:**   * Se debe permitir cambiar la información básica del diagramador. * La dirección de correo electrónico no puede ser cambiada debido a que es el identificador de la cuenta dentro del negocio de la aplicación. * Se debe permitir subir archivos al servidor, esto permitirá cambiar la imagen de identificación del diagramador. * Considerar el tamaño máximo de las imágenes subidas. * Las imágenes deben ser alojadas en una carpeta específica dentro del servidor. * En base de datos se almacenará la URL relativa al proyecto dentro del sistema de archivos del servidor. * Es posible descartar el avatar personalizado y utilizar la imagen por defecto. | | |

Tabla . Formato de historia de usuario H5

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H6 | **Nombre de historia:** Cambiar contraseña de cuenta | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 7 |
| **Prioridad:** Baja | | **Estimación:** 5 días |
| **Descripción:**  Permite cambiar la contraseña del usuario, es necesario siempre ingresar la contraseña anterior. El formulario solicita una nueva contraseña y una confirmación para garantizar que el usuario puede recordarla fácilmente. Solo se podrá cambiar la contraseña si la información requerida es correcta, es decir, si la anterior contraseña coincide con la guardada en base de datos y la nueva contraseña y su confirmación son exactamente iguales. | | |
| **Reglas u Observaciones:**   * Se debe solicitar la contraseña anterior para poder cambiarla. * El cambio se hace mediante un formulario que tiene 3 campos, la contraseña anterior, la nueva contraseña y una confirmación de la nueva contraseña. * Las reglas para la nueva contraseña deben ser las mismas ya mencionadas. | | |

Tabla . Formato de historia de usuario H6

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H7 | **Nombre de historia:** Crear diagrama | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | | **Estimación:** 7 días |
| **Descripción:**  Esta opción le permite a un usuario crear un diagrama para sí mismo. Los datos básicos de entrada son solamente el nombre y una descripción opcional del diagrama. No se realiza ninguna validación especial aparte de que exista un nombre. El usuario quien crea el diagrama es asignado como propietario del mismo y puede compartirlo con otros. | | |
| **Reglas u Observaciones:**   * La opción debe localizarse dentro del *Tablero de Control*. * Se debe solicitar el nombre del diagrama y una descripción opcional. * Es necesario almacenar el propietario y el último diagramador que hizo modificaciones sobre el diagrama, así como la fecha de creación y modificación. * Se debe disponer de una tabla en donde el diagramador pueda ver todos los diagramas que tiene creados. El diagramador puede crear tantos diagramas como quiera. * Las columnas de la tabla deben representar la información del diagrama, y es muy importante conocer rápidamente quien fue la última persona que modificó dicho diagrama. * El nombre del diagrama no debe ser necesariamente único, es decir, varios diagramas pueden tener el mismo nombre; el diagramador decide colocar el nombre que quiera. * Debe existir un formulario, preferiblemente dentro de un dialogo modal, que se abra al momento de pulsar el botón “Nuevo Diagrama”. * Sería bueno evitar que la página se recargue al momento de guardar los diagramas, ya que esto molesta al usuario. * La tabla de diagramas debe actualizarse para incluir el nuevo diagrama. * El estado de los botones y la selección en la tabla debe mantenerse. * El nuevo diagrama se creara completamente en blanco, es decir, no contendrá ningún elemento de diagrama de clase por defecto. | | |

Tabla . Formato de historia de usuario H7

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H8 | **Nombre de historia:** Editar diagrama | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 6 |
| **Prioridad:** Media | | **Estimación:** 5 días |
| **Descripción:**  Permite cambiar los datos de creación del diagrama (nombre y descripción). Para poder ejecutar esta opción es necesario que el usuario tenga los privilegios necesarios para ello. Es decir, es necesario que el usuario sea el propietario del diagrama, o que tenga acceso de escritura al diagrama en caso de que fuera compartido por otro usuario. | | |
| **Reglas u Observaciones:**   * Se puede cambiar el nombre y la descripción del diagrama. * Se deben controlar los privilegios del diagramador que intente cambiar los datos sobre el diagrama. * La edición se realiza sobre el diagrama actualmente seleccionado en la tabla. * El botón de edición está inactivo cuando no se tienen privilegios suficientes. | | |

Tabla . Formato de historia de usuario H8

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H9 | **Nombre de historia:** Eliminar diagrama | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 6 |
| **Prioridad:** Media | | **Estimación:** 5 días |
| **Descripción:**  Permite eliminar un diagrama de la lista. | | |
| **Reglas u Observaciones:**   * Se debe confirmar el borrado antes de proceder con la operación. * Los diagramas solo pueden ser borrados por el propietario. * Si un diagramador tiene un diagrama compartido y desea borrarlo, en realidad se le quita el privilegio otorgado y no se borra el diagrama original. * Si el propietario del diagrama decide borrarlo, entonces se le debe notificar que otros diagramadores pueden verse afectados y que estos no podrán ver el diagrama nuevamente. | | |

Tabla . Formato de historia de usuario H9

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H10 | **Nombre de historia:** Copiar diagrama | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 6 |
| **Prioridad:** Baja | | **Estimación:** 5 días |
| **Descripción:**  Esta opción permite copiar un diagrama al cual se tiene acceso, ya sea compartido o propio. Los datos son copiados exactamente pero el propietario de la copia será quien realiza la operación. Los diagramas compartidos pueden ser copiados pero ninguno de los diagramadores podrá ver el nuevo diagrama a menos que se comparta explícitamente. | | |
| **Reglas u Observaciones:**   * Los datos del diagrama se copian intactos, sin embargo es posible cambiar el nombre y la descripción. Se debe abrir el dialogo de creación de diagramas con los campos previamente llenos. * Un diagramador solo puede copiar un diagrama si tiene el privilegio para editarlo, por lo tanto un usuario a quien solamente se le ha otorgado privilegios de lectura no podrá hacer uso de esta opción. * Solo se debe copiar la información específica del diagrama, no se copiaran elementos compartidos, en caso de que el diagrama copiado haya sido compartido con otros diagramadores. | | |

Tabla . Formato de historia de usuario H10

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H11 | **Nombre de historia:** Compartir diagrama | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 5 |
| **Prioridad:** Alta | | **Estimación:** 12 días |
| **Descripción:**  Permite al diagramador compartir diagramas con otros diagramadores registrados en el sistema. Esto permite que posteriormente puedan editar colaborativamente el diagrama compartido. Básicamente, el diagramador selecciona a quienes quiere compartir el diagrama, otorgándoles cierto nivel de acceso.  Los niveles de acceso son ESCRITURA Y LECTURA; dependiendo del privilegio otorgado el diagramador a quien le fue compartido el diagrama, podrá realizar operaciones sobre este. | | |
| **Reglas u Observaciones:**   * Para compartir un diagrama es necesario ser el propietario del mismo. Esta condición debe verificarse al momento de habilitar el botón “Compartir Diagrama”. * Debe existir un dialogo emergente, que contenga el formulario para compartir los diagramas. * El control utilizado para seleccionar los diagramadores debe ser intuitivo y fácil de usar. Además debe permitirle al propietario del diagrama, visualizar fácilmente los diagramadores a los cuales quiere compartir su diagrama. * Los diagramadores pueden ser listados en una tabla o una lista de selección, pero es importante que se puedan seleccionar varios. * Es indispensable contar con un mecanismo de filtrado de los diagramadores; ya sea por nombre o apellidos, de tal manera que no sea necesario buscarlos manualmente uno por uno dentro del listado. * Un diagrama solo puede ser compartido una sola vez al mismo diagramador; por lo tanto, si este ya fue seleccionado anteriormente, no debe aparecer nuevamente en el listado. * Se debe permitir al propietario del diagrama, visualizar los diagramadores con quienes ha compartido el diagrama. De esta manera, es necesario que una vez seleccionado un diagrama en la tabla principal, se carguen inmediatamente los elementos compartidos y se visualicen dentro de otro listado. Esto es comúnmente conocido como formularios Maestro-Detalle. * Este listado puede ser una tabla adicional, que se muestra al lado de la tabla principal. * Es importante observar qué privilegio se le otorgó al diagramador. * Los demás diagramadores pueden ver quienes tienen acceso al diagrama. | | |

Tabla . Formato de historia de usuario H11

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H12 | **Nombre de historia:** Cambiar privilegio otorgado | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 5 |
| **Prioridad:** Media | | **Estimación:** 5 días |
| **Descripción:**  Esta opción le permite al propietario del diagrama modificar el privilegio otorgado a un diagramador. Básicamente el propietario puede cambiar de LECTURA a ESCRITURA o viceversa. | | |
| **Reglas u Observaciones:**   * El propietario del diagrama selecciona dentro de la tabla de elementos compartidos, el diagramador a cual quiere modificar su privilegio; y presiona sobre la opción “Cambiar Privilegio”. * El sistema debe confirmar la operación antes de proceder, una vez se confirme la acción, el sistema procede a cambiar el privilegio. * Una vez se realice el cambio, la tabla de elementos compartidos debe actualizarse y mostrar el nuevo privilegio. * Esta operación solo puede ser realizada por el propietario del diagrama. | | |

Tabla . Formato de historia de usuario H12

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H13 | **Nombre de historia:** Quitar privilegio otorgado | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 5 |
| **Prioridad:** Media | | **Estimación:** 5 días |
| **Descripción:**  Esta opción le permite al propietario del diagrama quitar el privilegio otorgado a otro diagramador. El diagramador no podrá ver nuevamente el diagrama en su listado. | | |
| **Reglas u Observaciones:**   * El propietario del diagrama selecciona dentro de la tabla de elementos compartidos, el diagramador a cual quiere quitar su privilegio; y presiona sobre la opción “Quitar Privilegio”. * El sistema debe confirmar la operación antes de proceder, una vez se confirme la acción, el sistema procede a cambiar el privilegio. * Al igual que con la opción Cambiar privilegio, se debe actualizar automáticamente la tabla. * Cabe aclarar que es necesario evitar que la página se recargue con cada operación, esto evita esperas innecesarias y permite una mejor experiencia de usuario. | | |

Tabla . Formato de historia de usuario H13

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H14 | **Nombre de historia:** Gestionar elementos del diagrama | |
| **Usuario:** Diagramador, Invitado | | **Iteración Asignada:** 2 |
| **Prioridad:** Alta | | **Estimación:** 15 días |
| **Descripción:**  Esta opción se hace parte del conjunto de funcionalidades proporcionadas a la hora de diseñar diagramas de clase. El objetivo es obtener un mecanismo mediante el cual se puedan agregar, editar y eliminar elementos del diagrama de clase.  Adicionalmente esta historia incluye operaciones básicas de edición como: Copiar, Cortar, Pegar, Deshacer-Rehacer, Orden Z, Zoom. | | |
| **Reglas u Observaciones:**   * Inicialmente el diagramador puede debe abrir el diseñador con un diagrama especifico. La operación se realiza haciendo clic sobre el nombre del diagrama en la tabla principal del Tablero de Control, esta operación redirige hacia la página del diagramador donde se inicia una sesión para editar el diagrama. * La interfaz de usuario del Diseñador debe ser muy coherente con las demás herramientas CASE. De tal manera que sea familiar para los usuarios del sistema. * Se debe disponer de un menú principal y una barra de herramientas, donde estarán localizados los controles de edición básicos. * Los paneles mínimos deben ser: *Paleta de Componentes*, *Área de Trabajo* y *Panel de Propiedades*. Estos deben ser paneles que se puedan ocultar o colapsar, en caso de que el usuario desee utilizar toda la pantalla para el área de trabajo. * Se debe disponer de atajos de teclado para las operaciones de edición básicas, estos atajos deben tener coherencia con los comunes de todas las herramientas de edición. * Se debe disponer de un menú contextual en el área de trabajo donde se pueda acceder a las opciones básicas de manera rápida. * Los elementos mínimos soportados deben ser: Clase, Interfaz, Enumeración, Paquete y Comentario. * Para agregar elementos, el usuario puede arrastrar los elementos desde la paleta de componentes hasta el área de trabajo. Con esta operación se crea un nuevo elemento dentro del diagrama, el cual tiene un nombre predefinido. Este nombre depende del tipo de elemento. * Para editar elementos, se hace uso del panel de propiedades, el cual debe permitir editar los datos básicos como: nombre, visibilidad, entre otros. * Considerar los modificadores de acceso, tales como: *visibilidad*, *static* y *abstract*. * Si un elemento es abstracto, el nombre debe tener estilo *cursivo*. | | |

Tabla . Formato de historia de usuario H14

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H15 | **Nombre de historia:** Gestionar atributos de los elementos. | |
| **Usuario:** Diagramador, Invitado | | **Iteración Asignada:** 2 |
| **Prioridad:** Alta | | **Estimación:** 10 días |
| **Descripción:**  Esta opción permite agregar, editar o eliminar atributos de un elemento, ya sea Clase, Interfaz o Enumeración. En el caso de las enumeraciones estos son llamados Literales en vez de Atributos. | | |
| **Reglas u Observaciones:**   * Se debe disponer de un dialogo emergente que contenga el listado de atributos del elemento seleccionado y un formulario para agregar o editar los mismos. * Información básica de un atributo: Nombre, Visibilidad, Tipo, Valor inicial; además de los modificadores *static* y *final*. * Por defecto debe permitirse seleccionar los tipos básicos de Java, debido a que la herramienta está enfocada en la generación de código en este lenguaje. * Se debe permitir crear atributos de cualquier tipo creado dentro del diagrama, es decir, el tipo de dato puede ser una clase, interfaz o enumeración creada dentro del mismo diagrama. * Adicionalmente a los tipos básicos, debe permitirse crear atributos tipo colección. Incluyendo aspectos como *Generic Types*, donde se crean colecciones de objetos cuyo tipo es determinado. * Para los atributos primitivos debe usarse el respectivo objeto *Wrapper*. * En el caso especial de las enumeraciones, solamente debe ser posible editar el nombre, ya que los Literales no tienen un tipo, ni visibilidad, ni modificadores de acceso. * Al momento de guardar, se debe actualizar automáticamente el diagrama e incluir el nuevo atributo dentro del compartimento de atributos del elemento editado. * Por defecto el componente GUI de un atributo debe poseer los siguientes campos: visibilidad (carácter UML), nombre y tipo. En caso de tener un valor por defecto también debe mostrarse. * Cuando un atributo tiene el modificador *static*, el nombre del componente GUI debe tener estilo subrayado. * Los atributos no pueden moverse fuera del contenedor. | | |

Tabla . Formato de historia de usuario H15

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H16 | **Nombre de historia:** Gestionar operaciones de los elementos. | |
| **Usuario:** Diagramador, Invitado | | **Iteración Asignada:** 3 |
| **Prioridad:** Alta | | **Estimación:** 10 días |
| **Descripción:**  Esta opción permite agregar, editar o eliminar operaciones de un elemento, ya sea Clase o Interfaz. Las enumeraciones no poseen métodos, así que no se debe permitir agregarlos. | | |
| **Reglas u Observaciones:**   * Se debe disponer de un dialogo emergente que contenga el listado de operaciones de la clase o interfaz seleccionada y un formulario para agregar o editar los mismos. * Información básica de una operación: Nombre, Visibilidad, Tipo de Retorno y Parámetros; además de los modificadores *static*, *final, abstract y synchronized*. * Las reglas con respecto al tipo de retorno y los tipos de los parámetros deben ser las mismas que para los atributos. * Las enumeraciones no deben tener operaciones, por lo tanto no deber permitirse abrir el dialogo cuando esté seleccionado este tipo de elemento. * Considerar que las operaciones pueden no tener un tipo de retorno, como es el caso de los constructores. * Se debe diseñar un componente GUI para añadir parámetros dinámicamente sin la necesidad de abrir otro dialogo para tal labor, es decir, todos los componentes deben estar ubicados en el mismo dialogo de operaciones, esto representa una mejor experiencia de usuario. * Al momento de guardar, se debe actualizar automáticamente el diagrama e incluir la nueva operación dentro del compartimento de operaciones del elemento editado. * Por defecto el componente GUI de una operación debe poseer los siguientes campos: visibilidad (carácter UML), nombre, los parámetros encerrados entre paréntesis y el tipo de retorno separado por el carácter “:”, seguir la nomenclatura UML para este elemento. * Cuando una operación tiene el modificador *static*, el nombre del componente GUI debe tener estilo subrayado. | | |

Tabla . Formato de historia de usuario H16

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H17 | **Nombre de historia:** Gestionar relaciones entre elementos. | |
| **Usuario:** Diagramador, Invitado | | **Iteración Asignada:** 3 |
| **Prioridad:** Alta | | **Estimación:** 12 días |
| **Descripción:**  Esta opción permite agregar, editar o eliminar relaciones entre dos elementos. | | |
| **Reglas u Observaciones:**   * Se debe soportar los tipos de relaciones básicos de UML para diagramas de clase: *Asociación*, *Composición*, *Agregación*, *Generalización*, *Realización* y *Dependencia*. Adicionalmente se necesita un tipo de relación no UML que permita vincular un comentario o un paquete a un elemento. * La paleta de componentes debe contener los tipos de relaciones mencionadas anteriormente, de esta manera el usuario puede seleccionar el tipo deseado para luego crear la relación entre los elementos. * Las relaciones se crean mediante un botón emergente que aparece al seleccionar un elemento en la parte superior derecha, el tipo de relación es tomado de la selección en el panel de componentes; si no hay selección entonces se debe crear del tipo por defecto que es Asociación. * Se debe visualizar la navegabilidad de la relación mediante flechas en los extremos de línea que conecta dos elementos. * Las relaciones tienen por lo general dos roles: origen y destino; estos roles representan atributos que se crean a partir de la dependencia entre los dos elementos relacionados. Estos roles (atributos) deben ser visibles en el diagrama. La información básica del atributo es la misma, pero añadiendo la multiplicidad que es otro factor importante. * A diferencia de los atributos normales, los componentes GUI que representan los roles de la relación se pueden mover a voluntad del usuario, sin embargo estos deben tener una posición relativa a la relación, y se deben ajustar en caso de que esta se desplace hacia otro lugar. * Se debe disponer de un dialogo emergente que contenga la información básica de una relación. Este dialogo permitirá editar aspectos como multiplicidad, navegabilidad, nombres de rol y el nombre de la relación. Solamente se debe permitir cuando el tipo de relación es Asociación, Composición y Agregación; en los otros tipos no es necesario ya que lo único variable es el nombre y este puede ser cambiado directamente en el panel de propiedades. * Cuando se crea una relación tipo Realización, el rol de origen, que en este caso debería ser una Clase, debe implementar las operaciones establecidas por la Interfaz. Por lo tanto el sistema debe realizar esta labor automáticamente. * Las asociaciones son por defecto dirigidas, por lo tanto se debe crear el rol de destino de manera automática. Lo mismo sucede con Composición y Agregación. | | |

Tabla . Formato de historia de usuario H17

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H18 | **Nombre de historia:** Guardar cambios del diagrama. | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 2 |
| **Prioridad:** Alta | | **Estimación:** 5 días |
| **Descripción:**  Esta opción permite guardar la representación del diagrama de manera persistente. | | |
| **Reglas u Observaciones:**   * El formato en el que se guarde los diagramas debe ser flexible y representar el modelo correctamente. * Los datos deben ser transportados desde el cliente hasta el servidor de manera constante, de tal manera que se debe disminuir al máximo la cantidad de información transportada para minimizar el consumo de red. * Se debe mantener una copia en memoria del lado del servidor y solamente se transportan los cambios realizados. * Cuando se realiza algún cambio en el diagrama se debe mostrar un indicador de cambios pendientes por guardar en la interfaz gráfica del diseñador; comúnmente expresado como (\*) en la mayoría de herramientas de edición. * Una vez se guardan los cambios el indicador desaparece. | | |

Tabla . Formato de historia de usuario H18

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H19 | **Nombre de historia:** Editar diagrama concurrentemente | |
| **Usuario:** Diagramador | | **Iteración Asignada:** 5 |
| **Prioridad:** Media | | **Estimación:** 15 días |
| **Descripción:**  Esta opción permite a varios diagramadores editar de manera concurrente el mismo diagrama, siempre y cuando todos ellos tengan acceso a él. | | |
| **Reglas u Observaciones:**   * Los cambios realizados por uno u otro diagramador deben reflejarse en todos los demás. Para ello es necesario implementar un mecanismo de observadores. * Considerar que los cambios de uno no deben afectar o hacer conflicto con los demás. | | |

Tabla . Formato de historia de usuario H19

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H20 | **Nombre de historia:** Modo demostración | |
| **Usuario:** Invitado | | **Iteración Asignada:** 6 |
| **Prioridad:** Baja | | **Estimación:** 5 días |
| **Descripción:**  Un invitado de la aplicación podrá hacer uso de ella sin registro previo, con el objetivo de obtener información o verificar los beneficios que esta puede prestarle. En la página de bienvenida se encuentra un link de ingreso, este link redirige al usuario al *Diseñador* de diagramas y crea un diagrama vacío. El usuario no podrá guardar los diagramas ya que para ello necesita una cuenta de diagramador. Sin embargo puede hacer uso completo de las herramientas especiales, tales como Generar Código, Generar Imágenes y demás. | | |
| **Reglas u Observaciones:**   * La opción guardar diagrama debe estar deshabilitada. | | |

Tabla . Formato de historia de usuario H17

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H21 | **Nombre de historia:** Generar código fuente | |
| **Usuario:** Diagramador, Invitado | | **Iteración Asignada:** 4 |
| **Prioridad:** Alta | | **Estimación:** 15 días |
| **Descripción:**  Esta opción le permite al usuario generar el código fuente del diagrama que se encuentra editando mediante el Diagramador. El código fuente es generado en lenguaje Java. Los archivos fuentes pueden ser descargados uno a uno, o descargar un empaquetado que contenga todos los archivos generados. | | |
| **Reglas u Observaciones:**   * Se debe permitir descargar los archivos individuales o todo el diagrama empaquetado en un archivo ZIP. * Se debe respetar la distribución de paquetes de los elementos creados en diagramas, es decir, si se descarga todo el código fuente comprimido, este código debe estar organizado en las respectivas carpetas que representen el paquete. * Se deben seguir todas las reglas del lenguaje a la hora de generar el código, de tal manera que se garantice que el código pueda ser compilado correctamente. * Considerar todas las validaciones pertinentes, debido a que el metamodelo de UML permite conceptos que no son soportados en el lenguaje Java, por ejemplo, la múltiple herencia. | | |

Tabla . Formato de historia de usuario H21

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H22 | **Nombre de historia:** Generar imagen | |
| **Usuario:** Diagramador, Invitado | | **Iteración Asignada:** 4 |
| **Prioridad:** Alta | | **Estimación:** 10 días |
| **Descripción:**  Permite obtener una representación plana del diagrama en una imagen con formato PNG. Esta opción es muy útil para realizar documentaciones o impresiones del diagrama completo. | | |
| **Reglas u Observaciones:**   * La imagen generada debe contener todos los elementos del diagrama. | | |

Tabla . Formato de historia de usuario H22

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H23 | **Nombre de historia:** Generar constructor y métodos get/set | |
| **Usuario:** Diagramador, Invitado | | **Iteración Asignación:** 4 |
| **Prioridad:** Baja | | **Estimación:** 4 días |
| **Descripción:**  Esta opción permite generar operaciones de manera introspectiva para una clase. La idea principal es generar un constructor vacío y los métodos get/set de sus atributos. | | |
| **Reglas u Observaciones:**   * Los métodos se pueden agregar únicamente a elementos tipo Clase. | | |

Tabla . Formato de historia de usuario H23

### Arquitectura de la Aplicación

La arquitectura de la aplicación es una arquitectura basada en capas, cada capa cumple una función específica y dispone de componentes especializados para su correcto funcionamiento. La arquitectura establecida para la aplicación se asemeja a la arquitectura de la plataforma J2EE y de hecho se basa en dicha arquitectura, ya que la mayoría de herramientas usadas para el desarrollo son hechas en lenguaje Java.

La siguiente imagen describe a grandes rasgos la arquitectura usada para la aplicación.

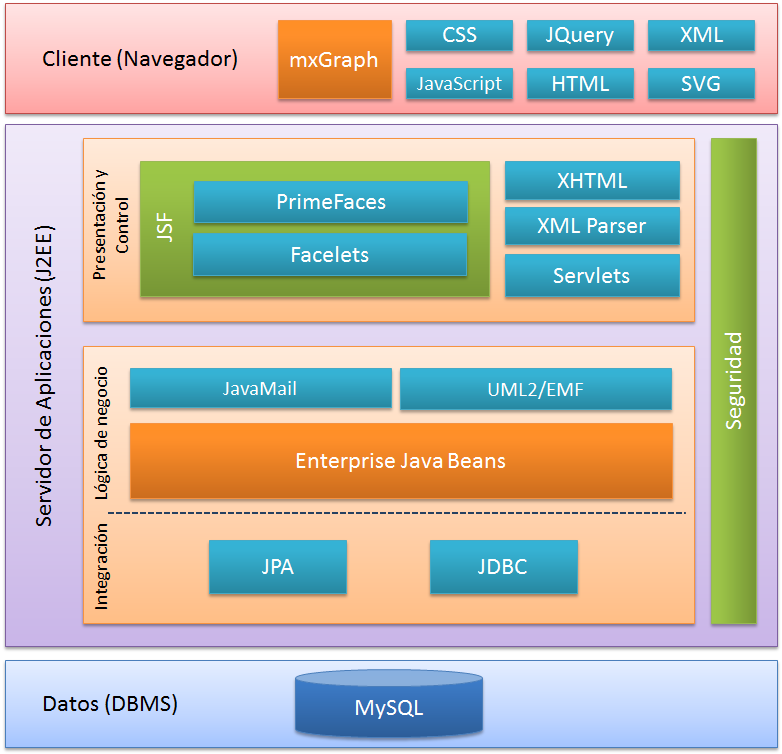


Figura . Arquitectura de la aplicación CLASS Modeler

La imagen anterior se divide en 3 grandes grupos: *Cliente*, *Servidor* y *Datos*; estos representan los niveles de la aplicación *CLASS Modeler*, cada nivel interactúa con los otros mediante interfaces definidas y cada uno tiene responsabilidades propias.

El nivel ***Cliente*** representa la interfaz mediante la cual un usuario interactúa con la aplicación. Básicamente estamos hablando de una Interfaz Gráfica de Usuario (GUI) a la cual un usuario de la aplicación accede y mediante la cual se comunica con los otros componentes. Este componente puede ser personificado por cualquier Navegador Web disponible en el mercado, siempre y cuando posea las características necesarias para ejecutar la aplicación.

El nivel ***Servidor*** representa el backend de la aplicación, desde y hacia la cual fluyen los datos manejados por la misma. Las tecnologías usadas para el desarrollo de la aplicación requieren que se disponga de un servidor especializado conocido como Servidor de Aplicaciones o Middleware, este tipo de software tiene características especiales más que el solo recibir y procesar peticiones HTTP.

Por último, el nivel ***Datos*** representa el almacenamiento de los datos de la aplicación de manera persistente, este es representado por un motor de bases de datos del mercado.

A su vez, cada componente se divide en diferentes capas, tal como se mencionó anteriormente, la arquitectura de la aplicación está basada en capas. A continuación se describen cada una de las capas.

* **Capa de Vista**: Esta capa se encuentra en el Cliente, básicamente se encarga de presentar los datos de la aplicación de una manera agradable al usuario, adicionalmente se encarga de tomar las acciones realizadas por el usuario y enviarlas al Servidor, en donde serán procesadas y se generará una respuesta en base a los datos.
* **Capa de Control y Presentación**: Esta capa se encuentra en el Servidor, se encarga de tomar los datos enviados desde la vista, procesarlos y validarlos, de tal manera que se envíen a la capa de negocio donde reposa la lógica necesaria para determinar cuál debe ser la respuesta esperada para determinada acción. Adicionalmente se encarga de tomar la respuesta de la capa de negocio, generar
* **Capa de Lógica de Negocio**: Contiene la lógica de operación de la aplicación, define los procesos y procedimientos realizados por la aplicación según las acciones del usuario.
* **Capa de Integración**: También conocida como la capa de acceso a datos, esta capa se encarga de la comunicación con el repositorio de datos de la aplicación.

En la figura vista anteriormente, adicional a la arquitectura de la aplicación, se mencionan algunas tecnologías empleadas en cada capa, en la siguiente sección se explica con más detalle cada una de estas tecnologías y la función que cumplen dentro de la aplicación.

### Definición de la Tecnología

#### JSF

El objetivo de la tecnología JavaServer Faces es desarrollar aplicaciones web de forma parecido a como se construyen aplicaciones locales con Java Swing, AWT (Abstract Window Toolkit), SWT (Standard Widget Toolkit) o cualquier otra API similar. Java Server Faces constituye un marco de trabajo (Framework) de interfaces de usuario del lado de servidor para aplicaciones web basadas en tecnología Java y en el patrón MVC (Modelo Vista Controlador).

Tradicionalmente, las aplicaciones web se han codificado mediante páginas JSP (Java Server Pages) que recibían peticiones a través de formularios y construían como respuesta páginas HTML (Hiper Text Markup Language) mediante ejecución directa o indirecta -a través de bibliotecas de etiquetas- de código Java, lo que permitía, por ejemplo, acceder a bases de datos para obtener los resultados a mostrar, realizar operaciones marginales como insertar o modificar registros en tablas relacionales, actualizar un carrito de la compra, etc.

JavaServer Faces pretende facilitar la construcción de estas aplicaciones proporcionando un entorno de trabajo (Framework) vía web que gestiona las acciones producidas por el usuario en su página HTML y las traduce a eventos que son enviados al servidor con el objetivo de regenerar la página original y reflejar los cambios pertinentes provocados por dichas acciones. En definitivas cuentas, se trata de hacer aplicaciones Java en las que el cliente no es una ventana de la clase JFrame o similar, sino una página HTML [21].

Un aspecto muy importante en JSF son los controladores; estos son conocidos como JSF Managed Beans, los cuales son un tipo especial de Bean y se caracterizan por tener un ciclo de vida específico dependiendo del ámbito otorgado por el desarrollador. Cada Bean Manejado se caracteriza por estar marcado con la anotación @ManagedBean. Los objetos de estas clases son creados y destruidos de manera automática por el contenedor cuando es necesario. Este proceso depende del ciclo de vida que se le otorgue a determinado controlador y el ciclo de vida determina la duración de un objeto en memoria, este ciclo de vida se conoce como *ámbito (Scope)* del bean. Cada bean de JSF se instancia cuando se solicita una página en la cual este es usado y permanece en memoria hasta que el ámbito lo determine. Los tipos de ámbitos enmarcados por las siguientes anotaciones: @Application, @Session, @View y @Request; mencionadas en orden de mayor a menor con respecto al tiempo de vida en memoria que tienen los beans marcados con estas anotaciones.

#### Facelets

Las interfaces de usuario son típicamente el aspecto más volátil durante el desarrollo de una aplicación web, y estas son a menudo un compuesto de código frágil que es difícil de cambiar, haciendo las interfaces de usuario costosas de implementar. Facelets fue originalmente desarrollado como una alternativa al manejo de vistas basado en JSP en JSF1.x. En JSF 2.0, Facelets reemplaza JSP como la tecnología de vista por defecto de JSF. Aparte de ser un mejor manejador de vistas, Facelets soporta un número de etiquetas para hacer plantillas y otros propósitos [21].

La mayoría de aplicaciones web siguen un patrón similar, en el cual todas las páginas tienen una distribución y estilo común. Por ejemplo, es típico para las páginas tener el mismo encabezado, pie de página y barras laterales.

Facelets permite encapsular esos aspectos comunes en una plantilla diseñada por el desarrollador, de tal manera que puede actualizar el look de la aplicación haciendo cambios únicamente en el témplate y no en todas las paginas individuales. Encapsulación es la piedra angular de la programación orientada a objetos y un principio bien conocido llamado DRY[[17]](#footnote-17).

#### PrimeFaces

Es una suite de componentes open source para Java Server Faces, conteniendo un conjunto de más de 70 componentes JSF para el desarrollo de aplicaciones RIA[[18]](#footnote-18) basadas en AJAX [22].

Las principales características de PrimeFaces son:

* Conjunto de componentes “ricos” (Editor HTML, Dialog, Auto Completado, Charts y muchos otros).
* Renderizado ligero y parcial de páginas construido en AJAX.
* Soporte nativo de AJAX Push/Comet.
* Compatible con otras librerías.
* Javascript discreto.
* Extensa documentación.

#### EJB

Los Enterprise Java Beans son una de las APIs que forman parte del estándar de construcción de aplicaciones empresariales J2EE. Su especificación detalla como los servidores de aplicaciones proveen objetos del lado del servidor, que son precisamente, los EJB.

Los EJB proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los EJB es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial, para centrarse en el desarrollo de la lógica de negocio [18].

Existen tres tipos de EJBs:

* **EJB de Entidad (Entity Beans):** Su objetivo es encapsular los objetos del lado del servidor que almacenan los datos. Los EJB de entidad presentan la característica fundamental de la persistencia. Sin embargo, en la documentación de Java para JEE 5.0, los Entity Beans desaparecen ya que son reemplazados por JPA (Java Persistence API). Dentro de esta categoría se encuentran dos tipos de persistencia: Persistencia Gestionada por el Contenedor (CMP) y Persistencia Gestionada por el Bean (BMP). El primero de ellos establece que el contenedor es el encargado de gestionar las conexiones, transacciones y operaciones de control sobre el DBMS, en el segundo, el programador debe lidiar manualmente con todos estos detalles, sin embargo tiene un control más personalizado y tiene acceso a bajo nivel del DBMS.
* **EJB de Sesión (Session Beans):** Gestionan el flujo de la información en el servidor. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor.
* **EJB dirigidos por mensajes (Message-Driven EJBs):** Son los únicos Beans con funcionamiento asíncrono. Usando el Java Messaging System (JMS), se suscriben a un tema (topic) o u una cola (queue) y se activan para recibir un mensaje dirigido a dicho tema o cola. No requieren de su instanciación por parte del cliente.

#### JPA

Java Persistence API es un Framework ligero basado en POJOs[[19]](#footnote-19) para persistencia en Java. Aunque el mapeo objeto-relacional es un componente primordial del API, también ofrece solución a retos arquitecturales de integrar persistencia en aplicaciones empresariales escalables [18]. JPA consiste en cuatro áreas:

1. El API de persistencia de Java.
2. El lenguaje de consultas (JPQL).
3. El API Java Persistence Criteria
4. Metadatos para mapeo Objeto/Relacional.

Una Entity es un objeto liviano del dominio de persistencia. Típicamente, una entity representa una tabla en una base de datos relacional, y cada instancia de una entity corresponde con una fila en esa tabla. El artefacto primario de programación de una entity es una clase entity, aunque estas también pueden usar clases auxiliares. Una clase Entity debe estar marcada por la *Anotación[[20]](#footnote-20)* @Entity de tal manera que puede ser catalogada como tal.

El estado persistente de una Entity puede ser accedido mediante las variables de instancia de la entity o las propiedades. Los campos o propiedades deben ser de algún tipo de dato del lenguaje Java: int, long, double, String, etc. Si una Entity usa propiedades persistentes, la Entity debe seguir las convenciones para métodos de los componentes JavaBeans. Las propiedades al estilo JavaBeans usan métodos get y set que permiten el acceso a los valores de la entidad y proveen el mecanismo de encapsulación. Cada propiedad de una Entity debe tener sus métodos get y set.

#### MxGraph

La librería mxGraph contiene un software de cliente, escrito en JavaScript, y una serie de backends para varios idiomas. El software de cliente es un componente gráfico con una envoltura de aplicación opcional que está integrado en una interfaz web existente. La siguiente imagen muestra la arquitectura de la librería:



Figura . Arquitectura de la librería mxGraph

El cliente requiere un servidor web para entregar los archivos necesarios en el cliente o puede ejecutarse desde el sistema de archivos local, sin un servidor web. Los motores que se utilizan tal cual, o pueden estar incrustados en una aplicación de servidor existente en uno de los idiomas compatibles.

Si existe un backend, a continuación, el cliente puede ser configurado para utilizar este backend en diversas formas, tales como:

* La creación de imágenes
* Almacenar y cargar diagramas
* Creación de una representación de objeto de un gráfico
* Compartir diagramas con otros clientes

Los escenarios anteriores pueden ser combinados de varias maneras, tales como el envío de una descripción XML de cada cambio en el backend como sucede, o guardado automático del diagrama para evitar la pérdida de datos en el cliente. El cliente también puede operar en el modo fuera de línea, en el que no requiere un motor o de un servidor web.

### Herramientas de Desarrollo

#### Glassfish

Es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Es gratuito, de código libre y se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL. La versión comercial es denominada Oracle GlassFish Enterprise Server (antes Sun GlassFish Enterprise Server).

GlassFish está basado en el código fuente donado por Sun y Oracle Corporation; este último proporcionó el módulo de persistencia TopLink. GlassFish tiene como base al servidor Sun Java System Application Server de Oracle Corporation, un derivado de Apache Tomcat.

#### MySQL

Sistema manejador de bases de datos SQL open source más popular, es desarrollado, distribuido y soportado por la corporación Oracle. MySQL es un sistema manejador de bases de datos. Para agregar, acceder y procesar datos almacenados en un computador de bases de datos, se necesita un DBMS tal como MySQL Server. Una base de datos relacional almacena datos en tablas separadas en vez de meter todos los datos en una gran sala de almacenamiento. Esto añade velocidad y flexibilidad.

MySQL soporta el lenguaje estructurado de consultas SQL, este es el lenguaje estandarizado más común usado para acceder bases de datos y es definido por el estándar ANSI/ISO SQL. Este sistema DBMS fue el seleccionado para servir de almacenamiento persistente de la aplicación CLASS Modeler, debido a que sus características se acoplan completamente a las necesidades de la aplicación y adicionando que es un software liviano y muy potente para tal labor.

#### Eclipse

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

## PLANIFICACIÓN

### Priorización de Historias de Usuario

Una vez determinadas las historias de usuario es necesario proceder a realizar una priorización y establecer cuáles deben ser implementadas en iteraciones tempranas y cuales se pueden postergar hasta el final del proyecto. Así como determinar la interdependencia que pueda existir entre estas y poder definir el orden de implementación.

La prioridad se asigna en conjunto con el cliente y se determina en base al impacto que la historia de usuario tiene sobre los objetivos del proyecto. De esta manera se establecen la siguiente clasificación de prioridades:

* **ALTA**: Significa que la historia de usuario tiene un impacto significativo en los objetivos del proyecto y debe implementarse para poder alcanzar los mismos.
* **MEDIA**: Significa que la historia de usuario tiene un impacto moderado sobre los objetivos del proyecto.
* **BAJA**: Significa que la historia de usuario es trivial y no impacta directamente los objetivos del proyecto, se puede describir como un *Nice to Have* (bueno tener) el cual puede aumentar la satisfacción del cliente.

Adicional a la prioridad asignada a cada historia de usuario, se ha considerado la interdependencia que existe entre ellas, el cual es un concepto importante al determinar el orden de implementación y la precedencia.

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | **PRIORIDAD** | **DEPENDENCIA** |
| H1. Crear cuenta de diagramador | MEDIA | - |
| H2. Activar cuenta de diagramador | MEDIA | H1 |
| H3. Reasignar contraseña | BAJA | H1 |
| H4. Iniciar y cerrar sesión | MEDIA | H1 |
| H5. Editar perfil de diagramador | BAJA | H4 |
| H6. Cambiar contraseña de cuenta | BAJA | H4 |
| H7. Crear diagrama | ALTA | H4 |
| H8. Editar diagrama | MEDIA | H7 |
| H9. Eliminar diagrama | MEDIA | H7 |
| H10. Copiar diagrama | BAJA | H7 |
| H11. Compartir diagrama | MEDIA | H7 |
| H12. Cambiar privilegio otorgado | MEDIA | H11 |
| H13. Quitar privilegio otorgado | MEDIA | H11 |
| H14. Gestionar de elementos del diagrama | ALTA | H7 |
| H15. Gestionar atributos de los elementos | ALTA | H14 |
| H16. Gestionar operaciones de los elementos | ALTA | H14 |
| H17. Gestionar relaciones entre elementos | ALTA | H14 |
| H18. Guardar cambios del diagrama | ALTA | H14, H15, H16, H17 |
| H19. Editar diagrama concurrentemente | MEDIA | H11, H14, H15,H16,H17 |
| H20. Modo Demostración | BAJA | H4, H14 |
| H21. Generar código fuente | ALTA | H14, H15, H16, H17 |
| H22. Generar imagen | ALTA | H14, H15, H16, H17 |
| H23. Generar constructor y métodos get/set | BAJA | H15, H16 |

Tabla . Priorización de historias de usuario

### Estimación de Historias de Usuario

La estimación del esfuerzo en base al tiempo que emplea el llevar a cabo cada historia de usuario. Esta tarea es realizada por el equipo de desarrollo considerando la complejidad que implica la implementación. Adicionalmente se considera el tiempo de investigación, análisis y diseño de la solución planteada.

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | **ESTIMACIÓN** | |
| H1. Crear cuenta de diagramador | 10 | días |
| H2. Activar cuenta de diagramador | 5 | días |
| H3. Reasignar contraseña | 7 | días |
| H4. Iniciar y cerrar sesión | 9 | días |
| H5. Editar perfil de diagramador | 6 | días |
| H6. Cambiar contraseña de cuenta | 5 | días |
| H7. Crear diagrama | 7 | días |
| H8. Editar diagrama | 5 | días |
| H9. Eliminar diagrama | 5 | días |
| H10. Copiar diagrama | 5 | días |
| H11. Compartir diagrama | 12 | días |
| H12. Cambiar privilegio otorgado | 5 | días |
| H13. Quitar privilegio otorgado | 5 | días |
| H14. Gestionar de elementos del diagrama | 15 | días |
| H15. Gestionar atributos de los elementos | 10 | días |
| H16. Gestionar operaciones de los elementos | 10 | días |
| H17. Gestionar relaciones entre elementos | 12 | días |
| H18. Guardar cambios del diagrama | 5 | días |
| H19. Editar diagrama concurrentemente | 15 | días |
| H20. Modo Demostración | 5 | días |
| H21. Generar código fuente | 15 | días |
| H22. Generar imagen | 10 | días |
| H23. Generar constructor y métodos get/set | 4 | días |

Tabla . Estimación de tiempo para historias de usuario

### Definición de Iteraciones

Basado en la priorización y estimación de las historias de usuarios, se definen las iteraciones del proyecto, determinando la cantidad de iteraciones (entregables del proyecto) necesarias para la completa ejecución del mismo y el cumplimiento de los objetivos, definiendo las historias de usuario cubiertas por cada una de ellas.

Basado el tamaño del proyecto se determinó que las iteraciones deben ser máximo de 3 semanas de duración, por lo tanto se trataran establecen la iteraciones basado en la estimación de tiempo realizada por cada historia de usuario. Otro factor adicional para la definición de las iteraciones es la dependencia que existe entre las historias de usuario, considerando que para poder implementar algunas de ellas es necesario que otras funcionalidades ya estén disponibles en la aplicación.

La siguiente tabla muestra el plan de iteraciones y la distribución de las historias de usuario en cada una de ellas.

|  |  |  |  |
| --- | --- | --- | --- |
| **ITERACIÓN** | **HISTORIAS DE USUARIO** | **DURACIÓN** | |
| Iteración 1 | H1, H2, H4, H7 | 31 | días |
| Iteración 2 | H14, H15, H18 | 30 | días |
| Iteración 3 | H16, H17 | 22 | días |
| Iteración 4 | H11, H12, H13, H19 | 37 | días |
| Iteración 5 | H21, H22, H23 | 29 | días |
| Iteración 6 | H8, H9, H10, H20 | 20 | días |
| Iteración 7 | H3, H5, H6 | 18 | días |

Tabla . Plan de iteraciones

### Metáfora del Sistema

Aplicación web basada en ambiente Cloud que asemeja una herramienta CASE desktop para la creación, edición y manejo de Diagramas de Clase UML. Permite a los usuarios diseñar de manera fácil diagramas de clase UML a través de una interfaz web, así como compartir sus diagramas con otros usuarios y editar concurrentemente los diagramas compartidos. Se permite agregar componentes desde una paleta al lienzo principal del diseñador, y se dispone de los elementos básicos para el diseño de diagramas de clase. Al final de la edición la aplicación permite obtener el código fuente que representa el diagrama dibujado.

## IMPLEMENTACIÓN

En primera instancia se describen los patrones de desarrollo que se utilizaron en la codificación de la herramienta, a continuación se menciona cada iteración del proyecto describiendo las historias de usuario que estas abarcan. Para cada iteración se describe los aspectos importantes que permiten su completa ejecución y los artefactos desarrollados, tales como interfaces gráficas, clases del dominio, servicios, persistencia de datos, controles y pruebas.

### Patrones de Desarrollo

#### Modelo Vista Controlador

Uno de los Frameworks más conocidos y ampliamente usado para el diseño de GUI es el Modelo-Vista-Controlador (MVC). El Framework MVC fue propuesto originalmente en la década de los 80 como una aproximación al diseño de GUI que permitió múltiples presentaciones de un objeto y estilos independientes de interacción en cada una de las presentaciones. El marco MVC soporta la presentación de los datos de diferentes formas e interacciones independientes con cada una de estas presentaciones. Cuando los datos se modifican a través de una de las presentaciones, el resultado de las presentaciones son actualizadas [6].

Este es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el modulo encargado de gestionar los eventos y comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador.

* El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación. Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada.
* El controlador: Responde a eventos (usualmente acciones de usuario) e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta de modelo. Se puede decir que el controlador hace de intermediario entre la vista y el modelo.
* La vista: Presenta el modelo (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario).

La siguiente imagen ilustra este patrón.



Figura . Patrón Modelo-Vista-Controlador

El Framework JSF de Java se basa específicamente en este patrón, siguiendo a cabalidad y forzando al desarrollador a programar en base a este concepto. De modo que las aplicaciones desarrolladas forzosamente aplican el patrón MVC.

#### Intercepting Filter

Cuando una petición ingresa a la aplicación Web, a menudo esta debe pasar varias pruebas de ingreso antes de llegar a la fase principal de procesamiento. Por ejemplo,

* ¿El cliente ha sido autenticado?
* ¿El cliente tiene una sesión valida?
* ¿La IP del cliente procede de una red verdadera o valida?
* ¿La ruta de la petición web viola alguna restricción?
* ¿Qué codificación usa el cliente para enviar los datos?
* ¿La aplicación soporta el tipo de navegador del cliente?

Algunos de estos ítems chequeados son pruebas, resultantes en una respuesta SI o NO que determina si el procesamiento continuará. Otros chequeos manipulan los datos de flujo de entrada en un formulario susceptible de ser procesado.

La solución clásica consiste en una serie de chequeos condicionales, con cualquier chequeo fallido la petición se aborta. Sentencias if/else anidadas son la estrategia estándar, pero esta solución conduce a código frágil y al estilo de programación de copiar-y-pegar, debido a que el flujo de filtrado y la acción de los filtros es compilada dentro de la aplicación [23].

La llave para resolver este problema en un manera flexible y discreta es tener un mecanismo simple para adicionar y remover componentes de procesamiento, en los cuales cada componente complete una acción específica de filtrado.

La solución es crear filtros conectables para procesar servicios comunes en una manera estándar sin requerir cambios en el código principal de procesamiento de la petición. Estos filtros intersectan las peticiones entrantes y las respuestas salientes, permitiendo pre y post procesamiento. Seriamos capaces de adicionar y remover estos filtros sin interferir los unos con los otros, sin requerir cambios en el código existente.

En java, este patrón se implementa bajo la clase Filter, la cual permite realizar el filtrado de las peticiones basada en la URL de la petición, si esta coincide con el patrón asignado al filtro entonces dicha petición es procesada por el filtro en cuestión.

Dentro de la aplicación CLASS Modeler este patrón se aplica para verificar o forzar el inicio de sesión por parte de los Diagramadores, y para brindar un grado de seguridad confiable, ya que existen páginas a las cuales solo se puede acceder si se ha autenticado dentro de la aplicación mediante el formulario de inicio de sesión. Por lo tanto si una persona solicita una página específica y no se ha autenticado, la aplicación le responde con un mensaje apropiado o lo redirección a la página principal. Más adelante en la sección de seguridad se explica este tema con mayor detalle.

#### Front Controller

El sistema requiere un punto de acceso centralizado para manipular la petición de la capa de presentación para apoyar la integración de los servicios del sistema, recuperación de contenido, gestión de la vista y navegación. Cuando el usuario accede a la vista directamente sin ir a través de un mecanismo centralizado, se pueden presentar dos problemas:

* Se requiere Cada fin de proporcionar a sus propios servicios del sistema, a menudo dando por resultado duplicar el código.
* Ver navegación se deja a los puntos de vista. Esto puede resultar en vista mezclados contenido y la vista de navegación.

Además, el control distribuido es más difícil de mantener, ya que los cambios se a menudo tienen que hacer en muchos lugares.

El controlador proporciona un punto de entrada centralizado que controla y gestiona el tratamiento de la petición Web. Por puntos y controles de decisiones centralizadas, el controlador también ayuda a reducir la cantidad de código Java, llamado scriptlets, incrustado en la JSP.

Centralizar el control en el controlador y la reducción de la lógica de negocio en la vista promueve la reutilización de código en todas las solicitudes. Es un enfoque preferible a la código alternativo-incrustación en múltiples puntos de vista, porque este enfoque puede conducir a un ambiente reutilización por copiar y pegar más propenso a errores.

Típicamente, un controlador coordina con un componente despachador. Los despachadores son responsables de control de la vista y la navegación. Por lo tanto, un despachador elige la próxima vista para el usuario y el control de vectores al recurso. Los despachadores pueden estar encapsulados dentro del controlador directamente o se puede extraer en una separada componente [23].

Este patrón es típicamente implementado en JSF, ya que existe un control central el cual es un Servlet, al cual llegan todas las peticiones de la aplicación y este se encarga de re direccionarlas al controlador especifico.

#### Session Facade

En un ambiente de aplicación J2EE multicapa, algunos problemas comunes pueden surgir:

* Fuerte acoplamiento, que produce dependencia directa entre objetos del cliente y de negocio.
* Mucha invocación de métodos entre el cliente y el servidor, produciendo problemas de rendimiento de red.
* Falta de una estrategia de acceso uniforme al cliente, exponiendo objetos de negocio que no serán usados.

Una aplicación J2EE multicapa tiene numerosos objetos del lado del Servidor que son implementados como Enterprise Beans. Adicionalmente, algunos otros objetos arbitrarios pueden proveer servicios, datos, o ambos. Estos objetos son conjuntamente llamados objetos de negocio, desde que estos encapsulan datos y lógica de negocio.

Las aplicaciones J2EE implementan objetos de negocio que proveen procesar servicios como Beans de sesión. Objetos de negocio genéricos que representan un objeto vista del almacenamiento persistente y son compartidos por múltiples usuarios son usualmente implementados como Beans de entidad.

Este patrón provee:

* Una interfaz simple para el cliente que esconde toda la complejidad de interacción entre componentes de negocio.
* Reduce el número de objetos de negocio que son expuestos al cliente a lo largo de la capa de servicios sobre la red.
* Esconde del cliente las interacciones subyacentes y las interdependencias entre componentes de negocio. Este provee una mejor manejabilidad, centralización e interacción (responsabilidad), mayor flexibilidad y mayor habilidad para hacer frente a los cambios.
* Provee una capa de servicios uniforme y genérica para separar la implementación del objeto de negocio de la abstracción del servicio de negocio.
* Evita exponer los objetos de negocio subyacentes directamente al cliente para mantener el acoplamiento entre dos capas al mínimo.

Este patrón tiene varias estrategias de implementación, a continuación se mencionan cada una de las estrategias de implementación:

* Stateless Session Facade: Un proceso de negocio que solo necesita una invocación a un método para completar el servicio es un proceso de negocio no conversacional. Este tipo de proceso son comúnmente implementados como un Session Bean Stateless, debido a que no se necesita almacenar información temporal.
* Stateful Session Facade: Un proceso de negocio que necesita múltiples invocaciones a métodos para completar el servicio es un proceso de negocio conversacional. El estado conversacional debe ser almacenado entre cada invocación del método del cliente. En este escenario, un Session Bean Stateful puede ser un enfoque más adecuado para implementar el patrón Session Facade.

La implementación de Java de este patrón se llama EJB (Enterprise Java Beans). En cuyo caso también implementa Stateless y Stateful Session Beans, sin embargo Java agrega un tipo adicional de objeto que contiene características similares pero provee una funcionalidad adicional integrando el patrón Singleton, el cual de hecho lleva el mismo nombre de Session Bean. En la siguiente sección se explica un poco más de este tema [18].

#### Singleton

El patrón de diseño Singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

El patrón Singleton se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).

La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, sólo uno de ellos debe lograr crear el objeto. La solución clásica para este problema es utilizar exclusión mutua en el método de creación de la clase que implementa el patrón.

Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único (como puede ser el ratón o un archivo abierto en modo exclusivo) o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación.

El patrón Singleton provee una única instancia global gracias a que:

* La propia clase es responsable de crear la única instancia.
* Permite el acceso global a dicha instancia mediante un método de clase.

Declara el constructor de clase como privado para que no sea instanciable directamente.

#### Data Access Object

Muchas aplicaciones J2EE del mundo real necesitan usar datos persistentes en algún punto. Para muchas aplicaciones, el almacenamiento persistente es implementado con diferentes mecanismos, y hay muchas diferencias marcadas en las APIs usadas para acceder estos diferentes mecanismos de almacenamiento persistente. Otras aplicaciones pueden necesitar acceder datos que residen en sistemas separados. Por ejemplo, los datos pueden residir en sistemas MainFrame, repositorios Lightweight Directory Access Protocol (LDAP) y algunos otros. Otro ejemplo es donde los datos son proporcionados por servicios a través de sistemas externos tales como sistemas de integración B2B, servicios de tarjetas de crédito y así sucesivamente.

Tipicamente, las aplicaciones usan componentes distribuidos y compartidos tales como Entity Beans para representar datos persistentes. Una aplicación es considerada para emplear un Bean manejado de persistencia (BMP) para sus Entity Beans cuando estos accedan explícitamente el almacenamiento persistente. Una aplicación con requerimientos simples puede prescindir del uso de Entity Bean y en vez de ellos usar Session Beans o Servlets para acceder directamente al almacenamiento persistente para obtener o modificar datos. O, la aplicación puede usar Entity Beans con persistencia auto manejada por el contenedor (Servidor de Aplicaciones usualmente), y entonces dejar al contenedor manejar la transaccionalidad y los detalles de persistencia [23].

El patrón DAO implementa el mecanismo de acceso requerido para trabajar con la fuente de datos (Data Source). La fuente de datos puede ser un almacenamiento persistente como un DBMS, un servicio externo como intercambios B2B, un repositorio como una base de datos LDAP o un servicio de negocio accedido via CORBA o Sockets de bajo nivel.

### Estructura del Proyecto

Como se mencionó anteriormente el ambiente de desarrollo utilizado fue el IDE Eclipse, esta aplicación permite la configuración y desarrollo de todos los componentes que utiliza la aplicación, tanto el código fuente en Java como el desarrollo de los componentes Web del cliente, paginas HTML, rutinas de JavaScript y hojas de estilo en cascada CSS.

Dentro del IDE se creó un Proyecto de Aplicación Java Enterprise, el cual engloba y define el contexto de la aplicación en general, dentro de este proyecto existen algunos archivos de configuración que permiten definir diferentes aspectos de la aplicación, tal como el nombre, la descripción y los módulos que harán parte de ella. Una aplicación Enterprise se puede componer de varios módulos y estos módulos pueden ser de diferentes tipos. Por ejemplo, un módulo de la aplicación puede ser un Proyecto EJB que engloba componentes y servicios EJB, otro modulo puede ser una aplicación Web que publica una interfaz de usuario para acceder a los servicios de la aplicación y un último modulo puede ser una aplicación Desktop hecha en Swing o JavaFX la cual accede a los servicios prestados por el proyecto EJB.

La siguiente imagen describe la estructuración del código del proyecto y sus módulos.



Figura . Estructura y distribución del código de la aplicación CLASS Modeler

CLASS Modeler se compone de 3 módulos base, un módulo EJB, un módulo Web y un módulo de Tests; cada uno de ellos aloja los componentes creados para cada capa de la arquitectura del proyecto.

De esta manera, las capas de Vista, Presentación y Control se encuentran alojadas en el proyecto *CLASSModelerWeb*, y las capas de Lógica de Negocio e Integración están alojadas en el proyecto *CLASSModelerEJB*.

El proyecto *CLASSModelerTest* aloja las clases creadas como pruebas unitarias para los servicios desplegados por el proyecto EJB, estas clases fueron aisladas en un proyecto propio debido a que estas no hacen parte de los componentes necesarios para ejecutar de la aplicación y no deben ser incluidos en el código binario a desplegar en el servidor. Los archivos binarios resultado de compilar y empaquetar el proyecto se generan desde el proyecto principal *CLASSModeler****,*** estos van auto contenidos en un archivo .EAR que se crea mediante el IDE.

### Interfaz Gráfica de Usuario

Al tratarse de una aplicación web es común que la interfaz gráfica de usuario se diseñe en lenguaje HTML, sin embargo, y debido al uso de algunos Frameworks como JSF, Facelets y PrimeFaces la interfaz de usuario se realizó de una manera un tanto diferente.

Básicamente las páginas de la aplicación se crearon en base a plantillas, estas plantillas son diseñadas mediante la librería Facelets, la cual a su vez es parte del Framework JSF 2.0. Estas plantillas permiten que todas las páginas de la aplicación sigan un mismo patrón de diseño sin la necesidad de reescribir todas las secciones de la página cada vez que se crea una nueva, esto reduce la cantidad de código duplicado, posibilita la reutilización y ayuda a que el mantenimiento de la aplicación sea mucho más fácil.

Facelets permite la inserción de código HTML de una página a otra, definiendo secciones para tal propósito, esto significa que cada página contiene el código especifico correspondiente a su contenido y el contenido de las demás secciones simplemente se “heredan” de la plantilla base. Para la aplicación se crearon en total 2 plantillas: common.xhtml y portal.xhtml.



Figura . Plantillas principales para la interfaz gráfica de la aplicación

Estas dos plantillas son la base para todas las paginas HTML de la aplicación, de igual manera es posible que una plantilla se pueda crear en base a otra; en este caso la plantilla *portal.xhtml* está basada en la plantilla *common.xhtml* y a su vez sirve como plantilla de todas las demás páginas de la aplicación, a excepción de la página creada para el Diseñador, la cual está basada directamente desde *common.xhtml*.

A continuación veremos la distribución de algunas otras páginas y algunas imágenes tomadas de la aplicación real.



Figura . Jerarquía de las páginas basadas en plantillas

¿Cómo funciona el proceso?, como ya se mencionó anteriormente, las páginas web no son programadas en lenguaje HTML, en su lugar se usan las etiquetas especiales del Framework Facelets. Estas etiquetas son procesadas por el controlador JSF y son convertidas a su equivalente en código HTML nativo al momento de visualizar una página, esto se hace mediante un mecanismo de Renderizacion que permite transformar las etiquetas especiales e insertar el código proveniente de las plantillas base de la página que está siendo procesada.

Adicionalmente a este mecanismo basado en plantillas, la personalización y embellecimiento de las páginas está a cargo del lenguaje CSS, mediante hojas de estilo en cascada se personaliza las vistas, agregando tipos de fuente, tamaño y color de letras y entre otras.

Los elementos que normalmente se usan en formularios web tales como Cajas de Texto, Combos, Tablas y otros componentes son personalizado mediante el uso del Framework PrimeFaces, el cual como ya se mencionó anteriormente es una librería de componentes gráficos que permiten diseñar páginas web basadas en JSF de manera fácil.

### Iteración 1

Las historias de usuario abarcadas por esta iteración son H1, H2 y H5. Estas historias de usuario comprenden el inicio del proyecto y permiten, además de las características propias, la correcta implementación de las historias de usuario de iteraciones subsiguientes, por lo tanto el análisis realizado para esta iteración es de vital importancia y debe considerar aspectos que si bien no serán implementados aún deben engranarse con las funcionalidades posteriores. A continuación se describen las actividades a realizar, el diseño de los componentes necesarios para cumplir con cada historia de usuario y las pruebas realizadas para verificar cada funcionalidad.

#### Actividades a Realizar

Dentro del desarrollo de esta iteración es necesario realizar una serie de actividades que permiten la ejecución y el cumplimiento de los objetivos para la misma; todas estas actividades están enfocadas finalmente en la implementación de las historias de usuario para esta iteración, sin embargo y como se mencionó anteriormente también servirán de base para posteriores iteraciones.

* Investigar sobre patrones de diseño J2EE para el desarrollo de aplicaciones en Java.
* Investigar las APIs de Java para envío de correos electrónicos.
* Investigar las APIs de Java para encriptación de textos.
* Modelar la estructura de clases necesarias para implementar cada historia de usuario, considerando lógica del negocio y relaciones entre los componentes.
* Modelar los datos que deben ser persistentes y diseñar la estructura de base de datos.
* Diseñar la interfaz gráfica principal de la aplicación.
* Diseñar el formulario de creación de usuarios.
* Implementar un mecanismo para enviar correos electrónicos desde la aplicación.
* Implementar un mecanismo para encriptar textos desde la aplicación.
* Implementar el mecanismo de guardado para nuevos usuarios.
* Verificar los resultados mediante los casos de prueba.

#### Diagrama de Clases

En esta iteración tenemos algunos conceptos importantes que descartar, y en base a los cuales se ha construido el siguiente diagrama de clases. Algunos de estos conceptos son: Cuenta de Usuario, Diagramador, Invitado, Código de Seguridad, Email.

El siguiente diagrama de clase incluye las clases del dominio para esta iteración, básicamente las clases se distribuyen en 2 paquetes, *classmodeler.domain.user* y *classmodeler.domain.security*:



Diagrama . Diagrama de clases del dominio de la iteración 1

Como se muestra en el diagrama, un diagramador (*Diagrammer*) es un usuario de la aplicación, el cual puede acceder mediante registro previo y de manera autorizada; sin embargo, considerando los actores de la aplicación y la historia de usuario H17, es posible ingresar en modo invitado (*Guest*) y acceder a las funcionalidades del software. Es por ello que se realizó una abstracción del concepto usuario y se definió la interfaz *User*, de tal manera que esta encapsulara el comportamiento de cualquier usuario de la aplicación sin importar si fuese registrado o no. Esto será útil en la iteración 2.

Las clases *Diagrammer* y *SecurityCode* representan entidades del dominio que son mapeadas mediante anotaciones Java con las tablas de base de datos. Algunos tipos que representan grupos de conceptos finitos fueron modelados como enumeraciones, por ejemplo, el estado de la cuenta del diagramador (*EDiagrammerAccountStatus*) o el tipo de código de seguridad (*ESecurityCode*). Los números entre paréntesis representan los valores almacenados en base de datos para el literal del enumerado.

La enumeración EDiagrammerAccountStatus representa los diferentes estados de una cuenta de diagramador, los cuales incluyen:

* INACTIVATED (0): Este es el estado que adquiere la cuenta de diagramador una vez se es creada por medio del formulario.
* ACTIVATED (1): Este estado se otorga cuando el diagramador activa su cuenta mediante la historia de usuario H2.
* DEACTIVATED (2): Representa que la cuenta de diagramador ha sido desactivada y no se puede ingresar al sistema a través de ella.

La enumeración ESecurityCode define los tipos de código de seguridad que se generan para las acciones del diagramador:

* ACTIVATE\_ACCOUNT (0): El tipo de código de seguridad generado para activar la cuenta de diagramador.
* RESET\_PASSWORD (1): El tipo de código de seguridad generado para solicitar una reasignación de contraseña de la cuenta de diagramador.

Por convención a las enumeraciones se les colocó la letra E al inicio del nombre.

#### Modelo de Datos

El modelo de datos comprende el almacenamiento de la información persistente de esta iteración, básicamente se necesitan algunos artefactos que permiten realizar esta labor, a continuación se muestra el modelo de datos de esta iteración.

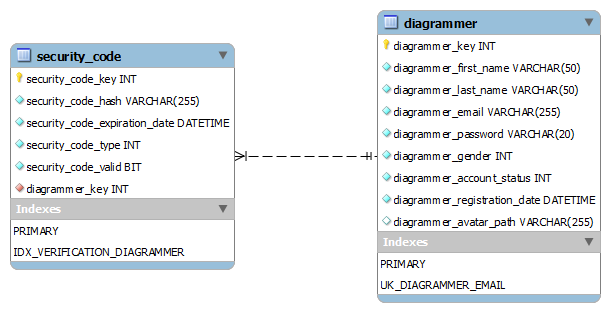


Diagrama . Modelo de datos para la iteración 1

La tabla *diagrammer* almacena la información básica y necesaria de una persona que se registra dentro de la aplicación, esta tabla se encuentra indexada por una llave primaria auto numérico. El email del diagramador está controlado mediante una llave única que restringe la posibilidad de ingresar dos veces la misma dirección para dos diagramadores distintos.

La tabla *security\_code* almacena la información del código de verificación, utilizado para confirmar la dirección de correo del diagramador, así como para reasignar la contraseña de la cuenta. Este incluye el código hash, la fecha de expiración y el tipo de verificación, esta tabla almacenará varios tipos de verificación para diferentes opciones de la aplicación.

La persistencia de los datos se realiza mediante el uso del Framework JPA, este permite interactuar con una abstracción de la base de datos representada por clases entidad (Entity Beans). Hablando específicamente de las tablas mencionadas anteriormente, estas son mapeadas y asociadas directamente con las entidades Diagrammer y SecurityCode (ver diagrama de clases anterior), de esta manera se pueden insertar, actualizar, eliminar u obtener registros de la base de datos como si se tratasen de objetos. Este mapeo se realiza mediante anotaciones agregadas al código de las clases, cada clase que contenga la anotación @Entity es considerada por el Framework y asociada con una tabla de la base de datos. Las filas o registros de la tabla son convertidos a objetos de la clase entidad, permitiendo manipular la base de datos bajo una simulación del paradigma orientado a objetos. De esta forma se encapsula completamente el acceso a los datos y permite al programador aislarse de aspectos como sentencias SQL y demás consideraciones de las bases de datos relacionales.

La conexión a la fuente de datos se realizó mediante la configuración de un recurso JDBC, el cual conecta a la base de datos relacional. El recurso JDBC es provisto por el servidor Glassfish, quien se encarga de mantener las conexiones entrantes a la base de datos, reutilizar las conexiones usadas y liberar el recurso una vez ya no se necesite, este mecanismo es llamado *Pool de Conexiones[[21]](#footnote-21)*. Dicho pool de conexiones se configura de manera manual mediante la consola de administración del servidor de aplicaciones (Ver documento *CLASS Modeler – Installation Guide.pdf*) y se identifica mediante un nombre (para nuestro caso se llama *jdbc/CLASSModeler*). Este nombre es el identificador mediante el cual el Framework JPA accede al recurso publicado por el servidor Glassfish y establece la conexiona a la base de datos. Este recurso JDBC puede ser accedido únicamente por las aplicaciones que se encuentren alojadas dentro del servidor, es decir, ninguna aplicación externa podrá acceder a los datos, lo cual significa cierto grado de seguridad.

Para establecer dicha conexión el Framework JPA dispone de un archivo de configuración, llamado *persistence.xml*, este archivo permite configurar la fuente de datos y algunas otras opciones para el funcionamiento del Framework. A continuación se muestra el contenido del archivo utilizado para la aplicación.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<persistence version=*"2.0"*

xmlns=*"http://java.sun.com/xml/ns/persistence"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_2\_0.xsd"*>

<persistence-unit name=*"CLASSModelerPU"*>

<jta-data-source>jdbc/CLASSModelerDS</jta-data-source>

<shared-cache-mode>NONE</shared-cache-mode>

<properties>

<property name=*"eclipselink.logging.level"* value=*"FINE"*/>

</properties>

</persistence-unit>

</persistence>

Mediante el uso de este archivo, el Framework es capaz de buscar la fuente de datos a la cual se va a conectar, es por ello que el nombre del recurso provisto por Glassfish debe ser el mismo que el consignado en el archivo. Al momento de desplegar la aplicación, se inicializa y configura el Framework JPA, probando la conexión al recurso definido. Los datos básicos de conexión, tales como URL, nombre de la base de datos, usuario y contraseña se encuentran en el recurso JDBC y no en el archivo XML.

Algo importante para mencionar del archivo es el nombre de la unidad de persistencia (en este caso llamado *CLASSModelerPU*), este nombre se ubica en la etiqueta <persistence-unit name=*"CLASSModelerPU"*> y representa la identificación del recurso interno para la aplicación. Mediante este nombre el contenedor es capaz de realizar la inyección del objeto *EntityManager* en clases EJB o cualquier otra que lo requiera. Esto se explica un poco más a fondo en las siguientes secciones.

#### Interfaz grafica

Los elementos de interfaz gráfica son básicamente paginas XHTML, estas páginas siguen el mismo concepto descrito en la sección 7.3.3 y son implementadas en base a plantillas. A continuación se muestran algunas imágenes tomadas de las interfaces graficas desarrolladas para esta iteración.

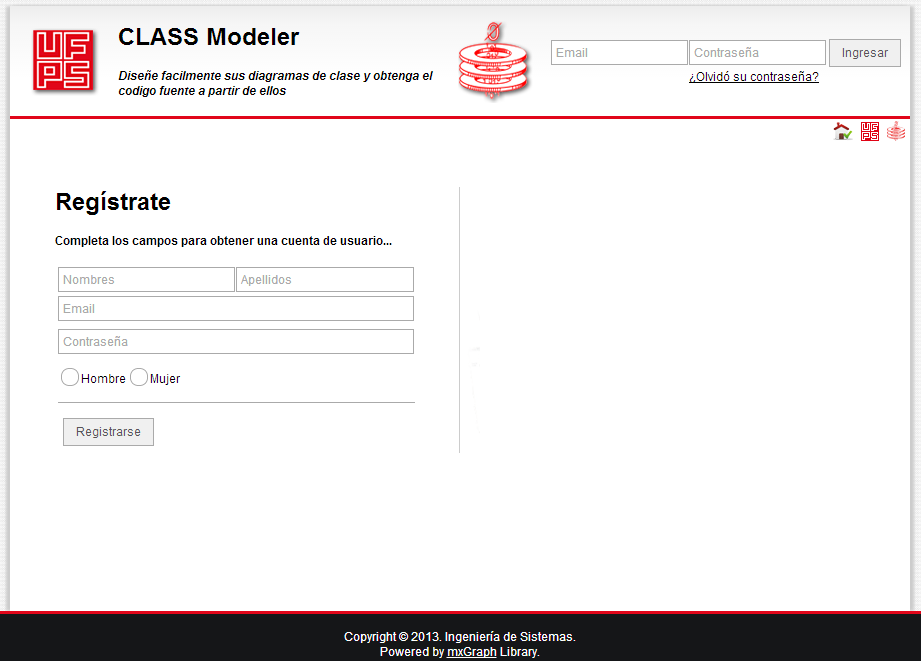


Figura . Interfaz gráfica principal de la aplicación CLASS Modeler

La página principal *index.html* se basa en la plantilla *portal.xhtml* (ver la Figura 48), y sobrescribe la parte central de dicha plantilla. Esta contiene 2 secciones, la sección izquierda *signUp.xhtml* contiene el formulario de registro para nuevos diagramadores, el cual fue diseñado para cumplir con la historia de usuario H1, la sección derecha aún se encuentra vacía y está destinada para una historia de usuario próxima.

La parte superior de la página contiene el encabezado de la aplicación, el cual es común para todas las demás páginas, y además contiene en la sección derecha, que contiene el formulario de inicio de sesión. De momento este formulario no realiza ninguna opción y será trabajado en una iteración posterior.

El formulario de creación de cuentas de diagramador *signUp.xhtml* se puede ver a continuación.

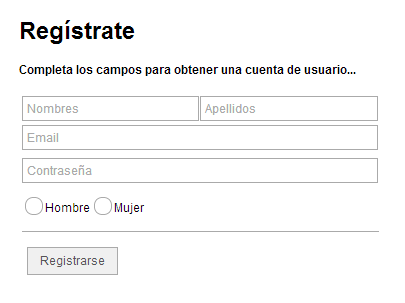


Figura . Interfaz gráfica para la creación de cuenta de diagramador

Por medio del formulario llamado *forgotPassword.xhtml* mostrado a continuación, el diagramador puede solicitar la reasignación de la contraseña para su cuenta, este formulario es mostrado cuando el diagramador pulsa sobre el link *“¿Olvidó su contraseña?”* que se sitúa en la parte superior de la página, debajo del formulario de inicio de sesión.

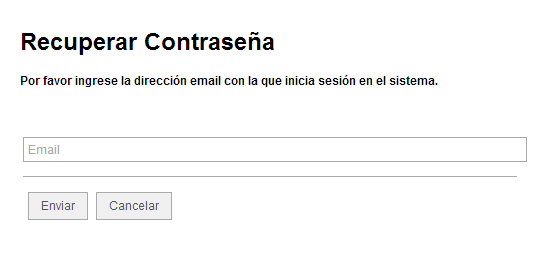


Figura . Interfaz gráfica para solicitud de reasignación de contraseña

Una vez solicitada la reasignación, el link de confirmación enviado al correo del diagramador conduce al formulario llamado *resetPassword.xhtml*, el cual le permite cambiar su contraseña. A continuación se muestra dicho formulario.

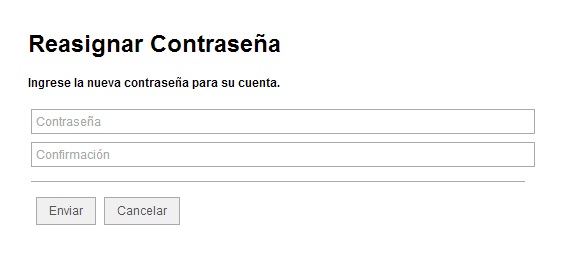


Figura . Interfaz gráfica para reasignar la contraseña

#### Controles

Los controladores desarrollados para esta iteración son los siguientes:

* SignUpControllerBean: JSF bean que controla la interacción del usuario con el formulario para crear una cuenta de diagramador. Este controlador tiene el ámbito @View, lo cual significa que mantiene su información mientras el usuario se encuentre visualizando dicho formulario.
* ActivateAccountControllerBean: JSF bean de ámbito @Request, el cual controla la activación de la cuenta de diagramador.
* ForgotPasswordControllerBean: JSF bean de ámbito @View que controla el formulario para solicitar la reasignación de la contraseña para la cuenta de diagramador.
* ResetPasswordControllerBean: JSF bean de ámbito @View que controla el formulario para reasignar la contraseña de la cuenta de diagramador.

#### Configuración de correo electrónico

El envio de correos electrónicos desde la aplicación se realiza mediante el uso del API JavaMail, básicamente consiste en la autenticación en una cuenta de correo existente desde donde se envían los correos electrónicos de la aplicación hacia las direcciones de correo de los diagramadores.

La autenticación se realiza con algunos parámetros que pueden ser configurados en el archivo *smtp\_email\_config.properties* que se encuentra en el proyecto *CLASSModelerEJB*. Este archivo es cargado cada vez que se envía un correo electronico desde la aplicación y se leen las propiedades para la autenticación, tales como el servidor de correo host, el puerto, el protocolo, el usuario y la contraseña. El proceso de envio de correos se encuentra centralizado en el servicio EmailService, desde donde se envían todos los correos electrónicos de la aplicación, es allí donde se cargan los datos del archivo de configuración.

A continuación se muestra el contenido del archivo usado en las pruebas hechas en fase de desarrollo.

mail.smtp.host=smtp.gmail.com

mail.smtp.socketFactory.port=465

mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory

mail.smtp.auth=true

mail.smtp.port=465

mail.transport.protocol=smtps

mail.user=gabriel.leonardo.diaz@xxxxxxx.com

mail.password=xxxxxxxx

Una vez la aplicación se encuentra en fase de producción, estas credenciales deben ser modificadas y direccionadas hacia la dirección de correo utilizada para la aplicación.

#### Servicios

Para esta iteración se definieron varios servicios que componen la capa de lógica de negocio de las historias de usuario abarcadas (ver sección 7.1.4 Arquitectura de la Aplicación). Estos servicios proveen métodos básicos para cada funcionalidad, tales como insertar diagramador, activar la cuenta, enviar el email de activación de la cuenta.

Estos servicios están definidos por las interfaces:

* UserService
* SecurityService
* EmailService

El siguiente diagrama de clases muestra la relación de los servicios para esta iteración, esta relación se mantiene y reutiliza para varias de las otras funcionalidades e iteraciones.



Diagrama . Diagrama de clases de los servicios para la iteración 1

Las interfaces son marcadas con la anotación @Local, que indica que la implementación de la clase es un recurso en el mismo contenedor.

Los servicios son implementados como clases Enterprise Java Beans Stateless, es decir, clases sin estado, debido a que estos sus métodos solo contienen lógica de negocio y no necesitan guardar un valores temporales a lo largo de la ejecución de sus métodos. Algunos servicios descritos en las siguientes iteraciones necesitan mantener estado e inclusive necesitan compartir información a través de diferentes sesiones y peticiones de usuarios.

Cada clase implementación de la interfaz definida para un servicio, es marcada con la anotación @Stateless, esto le indica al contenedor (Glassfish) que esta clase representa un componente Enterprise que debe ser publicado y reutilizado como un recurso de otras clases.

El acceso a los datos desde los servicios es realizado a través del Java Persistence API (JPA), esta API provee los mecanismos para obtener, insertar, actualizar y borrar registros de la base de datos mediante una interfaz central llamada *EntityManager*. Esta interfaz dispone de métodos tanto para operaciones básicas (CRUD), como para operaciones complejas como ordenamientos y agrupaciones. El objeto EntityManager se agrega al servicio mediante *Inyección de Dependencias[[22]](#footnote-22)*, y es realizado de manera automática por el contenedor. Lo único que se necesita es un atributo en la clase del servicio en el cual se almacena el objeto. Dicho atributo es marcado con una anotación especial llamada @PersistenceContext. El nombre de la unidad de persistencia debe ser exactamente el mismo que se asignó en el archivo *persistence.xml*.



Diagrama . Diagrama de clases de implementacion para servicos de la iteración 1

Una vez mencionados los servicios y su estructura, pasamos a describir la interacción básica de éxito para cada historia de usuario abarcada en esta iteración, los 4 siguientes diagramas de secuencia muestran el flujo de mensajes y los componentes que intervienen en cada una de ellas. Cada diagrama describe únicamente el flujo principal de la historia de usuario, sin embargo en cada una existen flujos alternativos que conducen a resultados diferentes. En cada diagrama se muestran los servicios utilizados para llevar a cabo la historia de usuario, sin embargo estos son representados a nivel de interfaces y no de la implementación concreta en EJB, esto es debido a que se quiere resaltar el flujo del proceso global sin mencionar los detalles de implementación.



Diagrama . Diagrama de secuencia para creación de cuenta de diagramador H1

**Descripción del Proceso**:

1. Una persona en calidad de visitante accede a la página de bienvenida de la aplicación, y diligencia los datos de registro que se encuentran allí.
2. Pulsa sobre el botón “Registrarse”.
3. Internamente se envía una petición HTTP (Request) con los datos ingresados por el visitante, esta petición es asíncrona por lo tanto la página no se recarga.
4. Los datos son recibidos por el controlador Servlet de JSF (Faces Servlet), el cual se encarga de ejecutar todo el ciclo de vida de las peticiones JSF.
5. En primera instancia se ejecutan los validadores, los cuales determinan si los datos ingresados son correctos. Estos validadores comprueban si los campos marcados como obligatorios tienen valor, y si la cantidad de caracteres es la correcta.
6. Para este formulario se desarrollaron 2 validadores JSF personalizados (*SignUpEmailJSFValidator* y *PasswordJSFValidator*), estos validadores comprueban que la dirección email tenga el formato correcto y que la contraseña tenga la cantidad de caracteres mínima obligatoria, respectivamente.
7. En caso de que durante los validadores se encuentre algún problema, se notifica a la vista y se muestran mensajes de alerta.
8. Una vez ha finalizado la fase de validación, se procesan los datos y se envían desde el Servlet controlador de JSF al controlador específico del formulario, llamado *SignUpControllerBean*.
9. A esta clase se le pasan los datos ingresados por el usuario mediante los métodos set de cada atributo, estos atributos concuerdan con los del bean *Diagrammer*, el cual será inicializado con estos los valores.
10. Luego el Servlet de JSF invoca el método en el controlador *SignUpControllerBean.processAJAX(),* el cual se encarga mediante el uso del servicio (*UserService*) de crear la cuenta de diagramador. Este controlador tiene el ámbito (*View*) lo que significa que el objeto es instanciado automáticamente por el contendor una vez se renderiza la página y permanece vivo mientras el usuario se encuentre visualizando la misma página, el método mencionado anteriormente es enlazado con el evento clic del botón en el formulario. El servicio es creado mediante inyección de dependencias por el contenedor de EJBs. Este concepto permite la instanciación automática de los objetos cada vez que son requeridos, y son destruidos una vez el objeto ya no es necesario. Además que permiten realizar la inyección de la implementación adecuada para la interfaz definida.
11. La implementación del servicio (*UserServiceBean*) contiene toda la lógica necesaria para crear la cuenta de diagramador, realizando las validaciones necesarias, por ejemplo que la dirección email sea única. Luego de hacer estas validaciones, se persiste en base de datos el nuevo diagramador, se crea el código de verificación y se envía el correo de confirmación a la dirección ingresada.
12. Finalmente se le notifica al visitante que la cuenta ha sido creada y que debe revisar su dirección de correo para activar la cuenta.



Diagrama . Diagrama de secuencia para activación de cuenta de diagramador H2

**Descripción del proceso**:

1. En la historia de usuario anterior se envió el correo de activación de la cuenta a la dirección email ingresada por el diagramador.
2. El diagramador ingresa a su dirección de correo y pulsa sobre el link de activación, el cual contiene el código de activación y la dirección email para confirmar los datos.
3. Cuando se pulsa el link, se envía una petición HTTP al controlador de JSF, el cual pasa el flujo de la petición al controlador llamado *ActivateAccountControllerBean*, el cual toma los parámetros de la petición y valida los datos. En caso de no encontrar la cuenta de diagramador o de no encontrar una dirección valida se cancela el proceso y se le notifica a la persona con un mensaje adecuado.
4. La verificación de los datos es realizada por el servicio (*UserService*), más específicamente el método *UserService.activateDiagrammerAccount()*, el cual es invocado desde el controlador mencionado anteriormente.
5. Adicional a las verificaciones básicas se comprueba si el código de activación no ha expirado, en caso de haberlo hecho, se genera un nuevo código y se envía nuevamente el email de activación mencionado en la historia anterior.
6. El código de activación es único y está unido a la dirección email del diagramador, por lo tanto si el código no concuerda con el email, este se considera inválido y no se continúa el proceso.
7. Una vez se pasan todas las validaciones se procede a cambiar el estado de la cuenta a ACTIVATED y se actualiza el registro en base de datos.
8. Por último se notifica al diagramador que la cuenta ha sido activada correctamente, esta notificación se realiza re-direccionando al usuario a la página *activateAccount.xhtml* en donde se muestra un mensaje con el resultado del proceso.



Diagrama . Diagrama de secuencia para solicitud de nueva contraseña H3



Diagrama . Diagrama de secuencia para reasignación de contraseña H3

**Descripción del Proceso:**

El proceso de reasignación de contraseña de divide en 2 partes, inicialmente la solicitud de reasignación de contraseña y posterior el cambio de contraseña por parte del diagramador.

* Solicitud de reasignación de contraseña:

1. El diagramador inicia el proceso al hacer clic en el link (*Olvidó la contraseña?)* ubicado en la página principal de la aplicación.
2. Este link redirige a la página *forgotPassword.xhtml*, dentro de la cual se solicita únicamente la dirección email usada por el diagramador.
3. Una vez se llena el campo y al presionar clic sobre el botón de aceptar, se envía el formulario con los datos. Esta petición es recibida y procesada por el bean JSF *ForgotPasswordControllerBean*, el cual se encarga únicamente de verificar el formato del email ingresado y de pasar los datos al servicio *UserService.resetPassword()*.
4. Internamente el servicio implementado (*UserServiceBean*) valida la existencia de la cuenta de usuario y verifica que la cuenta se encuentre ACTIVADA. En caso de que esta verificación no sea válida, entonces se arroja una excepción personalizada y se aborta el proceso, notificando al usuario sobre el problema encontrado.
5. Una vez ejecutadas las validaciones, se genera el código de verificación y se envía un email a la dirección ingresada. Esto se realiza para tener constancia que el propietario de la cuenta es quien está intentando recuperar o cambiar la contraseña.

* Reasignación de contraseña:

1. Como se mencionó anteriormente, el correo de verificación que se envía al email del diagramador contiene la información necesaria para realizar la reasignación. El diagramador solo tiene que hacer clic en la URL y automáticamente se lanza esta segunda parte el proceso.
2. El link envía una petición que es procesada por el bean controlador *ResetPasswordControllerBean*, el cual válida la información (email y código de verificación) y redirige el proceso a la vista que contiene el formulario de reasignación (*resetPassword.xhtml*).
3. En la vista mencionada el diagramador debe ingresar la nueva contraseña y la confirmación de la misma. Al hacer clic sobre el botón confirmar, se envía la petición y se procesa por el controlador *ResetPasswordControllerBean* el cual valida que los dos valores sean idénticos y que correspondan con una dirección email válida.
4. Luego los datos se pasan al servicio (*UserServiceBean*) el cual se encarga de modificar los datos de la cuenta.

#### Pruebas de unidad

Las pruebas de unidad están enfocadas a verificar el correcto funcionamiento de componentes específicos de la aplicación, en este caso estas pruebas van dirigidas directamente a probar los servicios provistos por la aplicación. Por lo tanto para esta iteración se encargaran de probar los servicios para crear diagramadores, enviar correos electrónicos y generar códigos de verificación.

A continuación se listan en la siguiente tabla, las pruebas unitarias realizadas a los servicios que intervienen en esta iteración.

|  |  |
| --- | --- |
| **Componente** | **Pruebas de unidad** |
| UserServiceTest | testExistsUser (); |
| testInsertDiagrammer (); |
| testInsertDiagrammer\_failAlreadyExisting(); |
| testActivateDiagrammerAccount(); |
| testActivateDiagrammerAccount\_failNonExisting(); |
| testActivateDiagrammerAccount\_invalidAccountStatus(); |
| testActivateDiagrammerAccount\_invalidSecurityCode(); |
| testActivateDiagrammerAccount\_expiredSecurityCode(); |
| testRequestResetPassword(); |
| testResetPassword(); |
| testGetDiagrammerByEmail(); |
| SecurityServiceTest | testInsertSecurityCode(); |
| testGetSecurityCode(); |
| EmailServiceTest | testSendEmail(); |
| testSendAccountActivationEmail(); |
| testSendResetPasswordEmail(); |

Tabla . Listado de pruebas de unidad para la iteración 1

El proceso de pruebas unitarias se realizó mediante el uso de un Framework para pruebas automáticas en Java. TestNG permite la realización de pruebas automáticas desde las pruebas unitarias hasta las pruebas de integración[[23]](#footnote-23).

En TestNG como en algunos otros Frameworks para testing, basta con marcar las clases con la anotación @Test para que puedan ser reconocidas y ejecutadas por el Framework.

A continuación se muestra una fracción de código utilizado para una de las pruebas unitarias que se realizaron, este se encuentra en la clase *UserServiceTest*.

/\*\*

\* Unit test to verify the service method {@link UserService#insertDiagrammer(Diagrammer)}.

\*

\* **@throws** SendEmailException

\* **@throws** InvalidUserAccountException

\* **@author** Gabriel Leonardo Diaz, 11.09.2013.

\*/

**public** **void** testInsertDiagrammer () **throws** InvalidUserAccountException,

SendEmailException {

String email = "leonar248@gmail.com";

Diagrammer diagrammer = **new** Diagrammer();

diagrammer.setFirstName("Nombre Diagramador");

diagrammer.setLastName("Apellido Diagramador");

diagrammer.setEmail(email);

diagrammer.setPassword("12345");

diagrammer.setGender(EGender.*MALE*);

userService.insertDiagrammer(diagrammer);

Diagrammer afterInsert = userService.getDiagrammerByEmail(email);

**try** {

**assert** (afterInsert != **null**) :

"Error: The diagrammer was not created";

**assert** (afterInsert.getAccountStatus()==EDiagrammerAccountStatus.*INACTIVATED*):

"Error: The account status must be INACTIVATED after inserting";

**assert** (email.equals(afterInsert.getEmail())) :

"Error: The email does not match with the saved value.";

}

**finally** {

userService.deleteDiagrammer(diagrammer.getKey());

}

}

Una vez se invoca el servicio con la información necesaria, se verifican los resultados del mismo mediante sentencias *assert.* Este es el mismo mecanismo usado para todas las pruebas unitarias realizadas en cada iteración. Por último se borra la información creada por la prueba unitaria de tal manera que el ambiente permanezca en el mismo estado con el cual se inició.

Las siguientes imágenes muestra el resultado de las pruebas unitarias hechas para cada servicio.

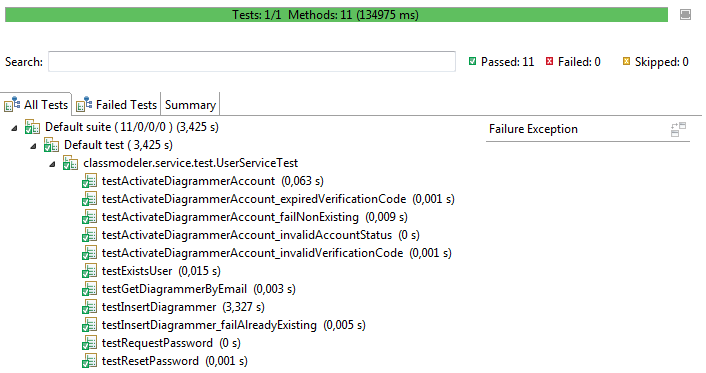


Figura . Resultados de las pruebas unitarias del servicio *UserServiceTest*

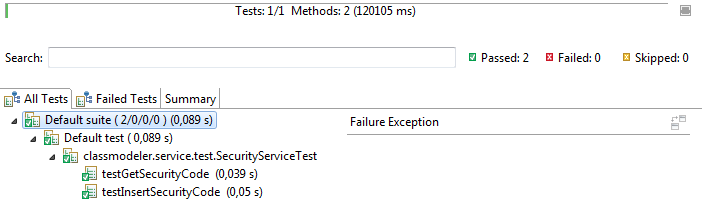


Figura . Resultados de las pruebas unitarias del servicio *SecurityServiceTest*

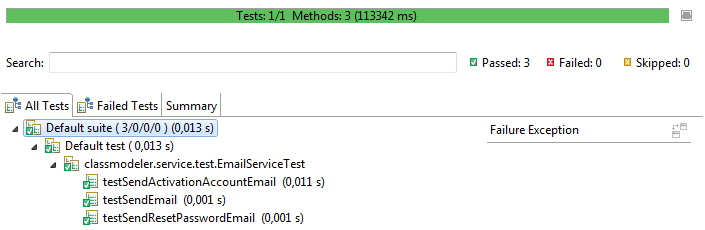


Figura . Resultados de las pruebas unitarias del servicio *EmailServiceTest*

### Iteración 2

Esta iteración incluye únicamente 2 historias de usuarios, H6 y H9. Estas historias abarcan las funcionalidades de Iniciar Sesión y Crear Diagrama respectivamente, lo que significa que una vez terminada esta iteración, un diagramador podrá iniciar sesión en el sistema y podrá crear un nuevo diagrama para el diagramador. A continuación se describe paso a paso las actividades realizadas y los componentes desarrollados para cumplir con los requerimientos.

#### Actividades a Realizar

* Investigar el mecanismo para mantener sesiones de usuario en una aplicación Java Web JSF.
* Investigar sobre aspectos de seguridad en el desarrollo de aplicaciones Web Java, tales como evitar ingresos no autorizados o forzar el inicio de sesión para acceder a determinada página.
* Diseñar la interfaz gráfica para el formulario de inicio de sesión.
* Modelar e Implementar los servicios necesarios para el inicio de sesión.
* Implementar el mecanismo de inicio de sesión, considerando los diferentes tipos de usuarios que pueden existir en la aplicación.
* Implementar los mecanismos de seguridad para proteger la aplicación de accesos no autorizados.
* Diseñar una interfaz gráfica para listar los diagramas creados por un diagramador.
* Implementar la búsqueda de los diagramas que pueden ser accedidos por el diagramador.
* Diseñar la interfaz gráfica para crear un diagrama.
* Modelar e Implementar los servicios necesarios para crear diagramas.
* Implementar el mecanismo de guardado de los nuevos diagramas.

#### Diagrama de Clases

El diagrama de clases para esta iteración agrega algunos conceptos nuevos a la iteración anterior, más específicamente el concepto Diagrama. Este objeto del dominio encapsula la información básica de un diagrama creado por un usuario. La siguiente imagen muestra las clases del domino de la iteración 2.



Diagrama . Diagrama de clases de la iteración 2

Un diagrama puede incluir una variedad de elementos UML, tales como Clases, Interfaces, Enumeraciones y varios tipos de relaciones. Todos los elementos del diagrama son convertidos y representados mediante el estándar XMI, dicha representación es almacenada como texto en el atributo llamado *XMI* de la clase *Diagram*. Esto permite cierto nivel de compatibilidad y portabilidad de los diagramas creado por la aplicación, además de liberar la aplicación de la complejidad del meta modelo UML al almacenar no los elementos individuales sino una representación global del diagrama que puede ser convertida de vuelta en un gráfico con la notación pertinente.

Adicionalmente un diagrama contiene algunos datos de auditoria, tales como el creador del diagrama y el último diagramador que realizó una modificación al mismo. El atributo *createdBy* determina el propietario del diagrama y el atributo *modifiedBy* determina el ultimo diagramador que realizó una modificación. Estos campos de auditoria permiten conocer de manera sencilla, información relevante con respecto al manejo que han tenido los diagramas (usado más adelante cuando se permita compartir diagramas entre diferentes usuarios).

#### Modelo de Datos

En relación con el nuevo objeto de dominio añadido en esta iteración, se incluyó una nueva tabla que almacena la información de los diagramas. La siguiente imagen ilustra el modelo de datos para esta iteración.

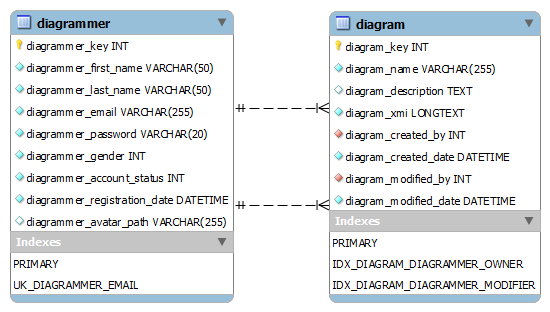


Diagrama . Modelo de datos de la iteración 2

La tabla *diagram* almacena la información de los diagramas creados por cada diagramador. Lo más representativo es la columna llamada *diagram\_xmi* la cual es de tipo LONGTEXT, esto debido a que la información almacenada allí puede ser de gran longitud, debido a que es un texto en formato XML. Las demás columnas almacenan información representativa del diagrama tal como el nombre y la descripción, además de los campos de auditoria que son llaves foráneas de la tabla *diagrammer*.

#### Interfaz grafica

La interfaz gráfica de esta iteración incluye el formulario para inicio de sesión que fue tomado desde la iteración anterior, además de la página que sirve de tablero de control para los diagramas que pertenecen el diagramador y por último el formulario para creación de diagramas.

A continuación se muestran algunas imágenes de las interfaces graficas desarrolladas para esta iteración.

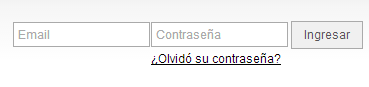


Figura . Formulario de inicio de sesión

Este formulario se encuentra en la página *logIn.xhtml*, la cual fue incrustada en la página principal *index.xhtml* desarrollada en la iteración anterior.

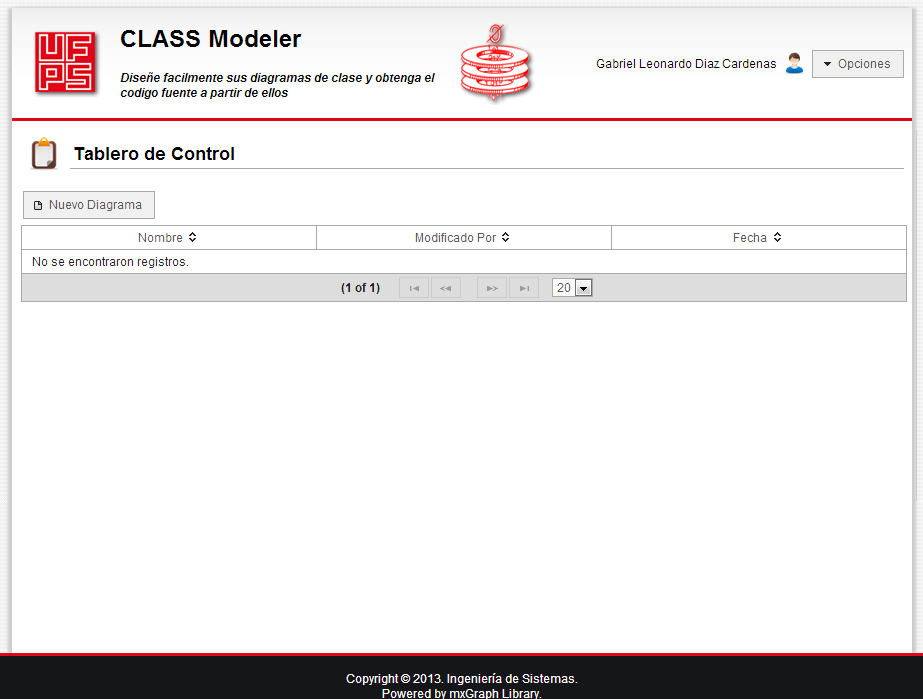


Figura . Tablero de control para listar los diagramas del usuario

El tablero de control *dashboard.xhtml* se basa al igual que la página principal en la plantilla *portal.xhtml* (ver la Figura 48), de esta “hereda” el encabezado, el pié de página y una sección central en blanco. La sección central se reemplaza con la tabla que contiene los diagramas creados por el diagramador y del encabezado se reemplaza el formulario de inicio de sesión, por el panel *userSection.xhtml* que contiene el nombre del diagramador, el avatar o imagen representativa y un menú desplegable que contendrá opciones para manipular la cuenta.

El tablero de control contiene una tabla en donde se listan todos los diagramas creados por el diagramador, esta tabla es un componente pertenece a la librería llamada PrimeFaces. Esta tabla es creada con una lista de objetos de tipo *Diagram*, y cada fila representa un objeto dentro de la lista. El componente prefabricado responde a la etiqueta <p:dataTable/> y permite añadir características, tales como filtros, paginación, selección, ordenamiento y algunos otros. La tabla se alimenta con la lista de diagramas que se encuentran en el controlador *DashboardControllerBean*. Las columnas de la tabla son fijas y se definen estáticas en el código.

A continuación se muestra una porción del código utilizado para crear este componente:

<p:dataTable id=*"diagramsTable"*

value=*"#{dashBoardController.diagrams}"*

var=*"diagram"*

emptyMessage=*"#{msgs.EMPTY\_TABLE\_MESSAGE}"*

resizableColumns=*"true"*

rowKey=*"#{diagram.key}"*

selectionMode=*"single"*

selection=*"#{dashBoardController.diagram}"*

paginator=*"true"*

paginatorPosition=*"bottom"*

rowsPerPageTemplate=*"10, 20, 30"*

rows=*"20"*>

<p:column headerText=*"#{msgs.COLUMN\_DIAGRAM\_NAME\_LABEL}"* sortBy=*"#{diagram.name}"*>

<h:commandLink title=*"#{msgs.BUTTON\_OPEN\_DESIGNER\_DIAGRAM\_LABEL}" /*>

</p:column>

<p:column headerText=*"#{msgs.COLUMN\_DIAGRAM\_MODIFIER\_LABEL}"*>

<h:outputText value=*"#{formatController.getFormattedUserName(diagram.modifiedBy)}"*>

</p:column>

<p:column headerText=*"#{msgs.COLUMN\_DIAGRAM\_MODIFIED\_DATE\_LABEL}"*>

<h:outputText value=*"#{formatController.getFormattedDate(diagram.modifiedDate)}"* />

</p:column>

</p:dataTable>

En el controlador es necesario disponer de un atributo de tipo lista con los métodos de acceso para que pueda ser invocado por el Framework en el momento de renderizar al página *dashboard.xhtml*.

Estos diagramas son cargados mediante la invocación a un servicio desarrollado para esta labor específica, el cual permite obtener los diagramas cuyo propietario es el diagramador enviado como parámetro.

* *DiagrammService.getDiagramsByDiagrammer(Diagrammer)*;

El controlador envía como parámetro el diagramador que se encuentra en sesión.

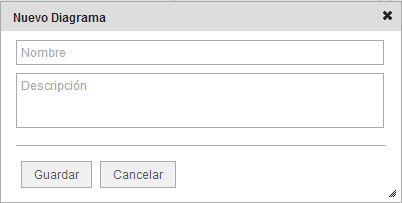


Figura . Formulario de creación de diagramas.

El formulario de creación de diagrama solicita la información básica del diagrama, nombre y descripción, este formulario se muestra en una ventana emergente al pulsar sobre el botón *Nuevo Diagrama* que se encuentra en la parte superior de la tabla. El nombre del diagrama es un campo obligatorio, sin embargo la descripción es opcional y el diagramador decide si la llena o no.

#### Seguridad

La seguridad es proporcionada por el mecanismo de sesiones en el servidor, cada sesión tiene un identificador único que se almacena en una cookie del cliente llamada *jsessionid*, esta cookie se envía en cada petición realizada y es usada para identificar la sesión del servidor creada para el usuario autorizado. Todo este mecanismo es trasparente para el desarrollador y el encargado del mantenimiento es el mismo Framework JSF.

Las sesiones son controladas por el bean manejado de JSF llamado *SessionControllerBean*, este bean tiene el ámbito @Session, por lo tanto permanece en memoria hasta que es destruido mediante la invocación del proceso de cierre de sesión o cuando se agota el tiempo máximo de inactividad en el sistema.

Esta clase almacena la información del usuario que ingresa al sistema, además sirve de suministro para los demás controladores de la aplicación que necesitan la información del usuario en sesión. La siguiente imagen muestra la estructura de la clase, los atributos y métodos que esta posee.

Otro aspecto importante de seguridad es la protección contra accesos no autorizados a páginas que requieren un inicio de sesión previo, tal como la página *dashboard.xhtml* que representa el tablero de control para los diagramas de un usuario.

Esta característica es manejada mediante un filtro web, el cual cumple la función de intersectar las peticiones enviadas desde el cliente y determinar si se trata de una petición autorizada. Este filtro es implementado por la clase *SessionFilter* que utiliza el API Java Servlet y se encuentra marcada con la anotación @WebFilter. La anotación permite añadir meta datos al filtro, tales como el nombre y los patrones de URL que este debe intersectar. Básicamente el filtro se ejecuta cada vez que se solicita la página que concuerda con los patrones definidos.

El siguiente diagrama representa el proceso de ejecución del componente desarrollado.



Diagrama . Proceso de filtro de peticiones hacia las páginas de la aplicación

Este es el proceso básico que se ejecuta cuando se solicita una página de la aplicación, si la página concuerda con el patrón “/pages/\*” y las validaciones no son satisfactorias, entonces se redirige al visitante a la página inicial *index.xhtml*. Los patrones básicamente se basan en las rutas relativas en la estructura de carpetas del proyecto web. Es decir, cualquier página que se encuentre sobre la carpeta “pages” requiere una autorización previa, por lo tanto para acceder a cualquiera de ellas se debe haber iniciado sesión.

#### Servicios

Los servicios definidos para esta iteración básicamente comprenden el manejo del inicio de sesión y los métodos que permiten manejar diagramas. Esta iteración agrega dos servicios adicionales llamados *SessionService* y *DiagramService*.



Diagrama . Diagrama de clases de los servicios para la iteración 2

A continuación se describe mediante diagramas de secuencia, el proceso básico de éxito de las dos historias de usuario de esta iteración.



Diagrama . Diagrama de secuencia de la historia de usuario H6

**Descripción del proceso:**

1. Un diagramador accede a la página principal de la aplicación con el objetivo de ingresar y utilizar sus servicios.
2. En el formulario de inicio de sesión *logIn.xhtml*, se solicita el email y la contraseña de la cuenta.
3. El diagramador ingresa la información y pulsa sobre el botón *Ingresar*, tras esta acción se envía una petición al servidor que es capturada por el Servlet de JSF y transfiere el flujo al método *process()* de clase controladora *LogInControllerBean*, esta clase procesa la petición y valida que los datos sean correcto, es decir, que el email y la contraseña no sean vacíos. En caso de no pasar la validación, se cancela la operación y se notifica a la vista.
4. Una vez se realiza la validación y el resultado es satisfactorio se transfiere el flujo a la clase principal para el manejo de sesiones, llamada *SessionControllerBean* la cual almacena los objetos pertinentes de sesión. Esta clase hace uso del servicio *SessionService*, quien tiene la lógica necesaria para determinar si las credenciales usadas para ingresar (es decir, el email y la contraseña) pertenecen a una cuenta de diagramador valida.
5. El servicio invocado es *SessionService.logIn(String, String)*, el cual busca la cuenta de diagramador asociada al email recibido y verifica en primera instancia que la cuenta exista, si no existen entonces se arroja una excepción. Como segundo paso verifica que la cuenta se encuentre activada. De no ser así, se arroja una excepción de tipo *InvalidDiagrammerAccountException*, la cual aborta el proceso de inicio de sesión; esta excepción es capturada en el controlador *LogInControllerBean* quien a su vez notifica a la vista para informarle al diagramador que su cuenta no ha sido activada.
6. Si la cuenta se encuentra en estado ACTIVATED, se procede a validar la contraseña, la cual es comparada con la que se encuentra almacenada en base de datos. Si la comparación determina que la contraseña es inválida entonces se aborta el proceso con otro tipo de excepción.
7. Por ultimo habiendo pasado todas las validaciones se retorna la cuenta de usuario que pertenece a dicha persona y se devuelve el flujo al controlador *SessionControllerBean*, el cual procede a almacenar la cuenta en memoria y a mantenerla allí mientras la sesión se encuentre activa.
8. Se redirige hacia la vista *dashboard.xhtml* que muestra el tablero de control con los diagramas pertenecientes al diagramador que inició sesión.



Figura . Diagrama de secuencia de la historia de usuario H9

**Descripción del proceso**

1. El proceso inicia cuando el diagramador pulsa sobre el botón *Nuevo Diagrama*. Inmediatamente se abre un panel de tipo modal que contiene el formulario de creación de diagramas.
2. El formulario presenta dos cajas de texto, una para el nombre del diagrama y la otra para la descripción del mismo.
3. El diagramador llena los campos y pulsa sobre el botón *Guardar*.
4. Los datos son enviados al servidor en una petición HTTP y son procesados por el controlador *DiagramControllerBean*.
5. Luego de validar que los datos sean correctos, es decir, que el nombre del diagrama no este vacio, el controlador invoca al servicio *DiagramService.insertDiagramm().*
6. Este servicio recibe un objeto de tipo *Diagram*, el cual debe contener como minimo el nombre, y el diagramador que es propietario del diagrama. Estos datos son configurados por el controlador un momento antes de invocar el servicio. El propietario es el diagramador en sesión, por lo tanto este objeto se toma del controlador de la sesión *SessionControllerBean*.
7. Internamente el servicio insertDiagram(), se encarga de poner los campos adicionales como la representación XMI para un diagrama vacio, la fecha de creación y modificación.
8. La información se persiste en base de datos y se retorna el nuevo objeto con todos los campos asignados.
9. El controlador DiagramControllerBean, recibe el objeto y lo añade al listado de diagramas que se muestran en la tabla

#### Pruebas de unidad

Las pruebas unitarias de los servicios de esta iteración se distribuyen de la siguiente manera.

|  |  |
| --- | --- |
| **Componente** | **Pruebas de unidad** |
| SessionService | testLogIn(); |
| testLogIn\_failNonExistingAccount(); |
| testLogIn\_failNonActivatedAccount(); |
| DiagramService | testInsertDiagram(); |
| testInsertDiagram\_failNonExistingAccount(); |

Tabla . Listado de pruebas de unidad para la iteración 2

**Resultados de las pruebas de unidad**

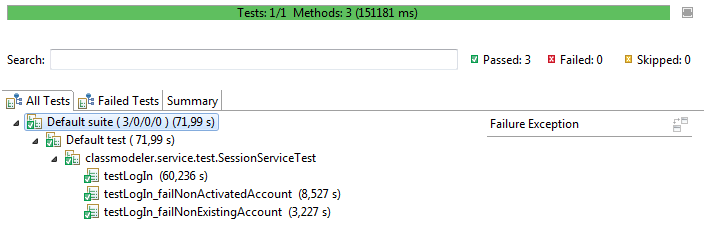


Figura . Resultados de las pruebas de unidad del servicio SessionService

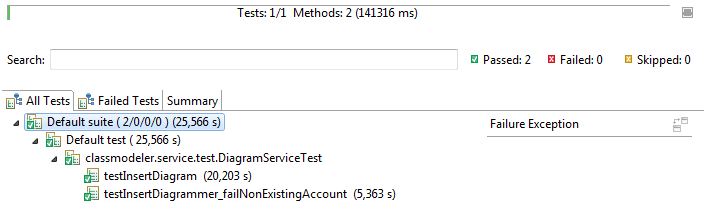


Figura . Resultados de las pruebas unitarias del servicio DiagramService

## PRODUCCIÓN

La etapa de Producción del proyecto consiste en el despliegue de la aplicación en un ambiente real, de tal manera que pueda ser usado directamente por el cliente. El despliegue de la aplicación implica en copiar el código fuente compilado, en el servidor de aplicaciones. Este código fuente puede estar empaquetado en un archivo .EAR[[24]](#footnote-24), o también se puede hacer mediante la copia manual de todos los archivos en una carpeta del servidor.

Básicamente este proceso se debe hacer una vez el servidor de aplicaciones Glassfish este en ejecución, inclusive el mismo servidor provee algunos mecanismos para el despliegue de la aplicación.

La segunda tarea importante para el despliegue de la aplicación es la instalación del script de base de datos, el cual contiene la estructura de tablas usadas para almacenar la información persistente de la aplicación. Esta tarea incluye configurar el Data Source para MySQL en el servidor de aplicaciones Glassfish.

Por último es necesario realizar algunas configuraciones antes de desplegar la aplicación, tal como configurar la cuenta de correo que servirá de remitente al momento de enviar correos desde la aplicación.

En los anexos del proyecto se encuentra el documento Guía de Instalación llamado *CLASS Modeler – Installation Guide.pdf*, dicho documento sirve de guía para realizar las configuraciones necesarias y desplegar correctamente la aplicación.

# IMPLEMENTACIÓN

# PRUEBAS

* Pruebas de Unidad y de Integración
* Pruebas de caja negra
* Pruebas de carga
* Encuesta
* Resultados

# DOCUMENTACIÓN

La documentación generada esta en:

* JavaDocs
* Manual de usuario
* Video Tutoriales

# CONCLUSIONES

* Se logró construir una herramienta que facilita y ayuda al aprendizaje
* Posibilidad de modelar diagramas de clase
* Conocimiento aportado y adquirido
* Capacidad de convertir metamodelo UML a Java.
* Integración de tecnologías.

# RECOMENDACIONES

* Adopción de la herramienta
* Realizar Documentación, JavaDocs
* Exportar e Importar XMI
* Generar Árbol del Proyecto
* Generar código en otros lenguajes
* Control de Versiones
* Edición de múltiples diagramas al tiempo.

# BIBLIOGRAFIA

1. **Amaya Torrado, Yegny Karina y Herrera Angarita, Lady Torcoroma.** *Identificación de las causas que generan problemas en el aprendizaje de fundamentos de programación de computadores en las facultades de ingenieira de sistemas de las universidades de la ciudad de Cúcuta.* Cúcuta N. de S. : Universidad Francisco de Paula Santander, Biblioteca Eduardo Cote Lamus, 2003. TIS 371.334A489i.

2. **GenMyModel.** GenMyModel official page. [En línea] GenMyModel Team, 2013. http://www.genmymodel.com/.

3. **Benson, David y Alder, Gaudenz.** Libreria JavaScript HTML 5 para diagramas. [En línea] JGraph Company, 26 de 12 de 2012. [Citado el: 23 de 01 de 2013.] http://www.jgraph.com/mxgraph.html.

4. **Cinergix.** Creately Pagina Web Oficial. [En línea] Cinergix Pty. Ltd., 2008. [Citado el: 26 de 12 de 2012.] http://creately.com/.

5. **Durman, David.** JoinJS Libreria HTML 5 JavaScript. [En línea] 2009. [Citado el: 26 de 12 de 2012.] http://www.jointjs.com/.

6. **Sommerville, Ian.** *Ingenieria de Software.* Madrid : Pearson Education S.A., 2005. ISBN 8478290741.

7. **Rumbaugh, James, Booch, Grady y Jacobson, Ivar.** *El lenguaje unificado de modelado, Manual de Referencia.* Madrid : Pearson Education S.A., 2000. ISBN 8478290370.

8. **Joyanes Aguilar, Jose Luis y Fernandez Azuela, Matilde.** *Java 2: Manual de Programación.* s.l. : McGraw-Hill, 2001. 8448131932.

9. **Pressman, Roger.** *Ingenieria de Software un enfoque práctico.* Mexico : McGraw Hill, 2005. ISBN 9701054733.

10. **Creative Commons.** Atribución No Comercial Compartir igual a 2.5 (Colombia). [En línea] http://creativecommons.org/licenses/by-nc-sa/2.5/co/legalcode.

11. **Sun Microsystems.** Licencia de la tecnologia Java SE. [En línea] http://www.bhweb.es/cdlibre/pdf/jvm.pdf.

12. **JGraph Limited.** Contrado de Licencia de Usuario Final mxGraph. [En línea] 2012. http://www.jgraph.com/mxGraphLicense.pdf.

13. **Free Software Fundation Inc.** GNU General Public Licence. [En línea] 23 de Junio de 2007. http://www.gnu.org/licenses/gpl.txt.

14. **Programa de Ingenieria de Sistemas.** Pensum académico vigente. [En línea] 2012. http://ingsistemas.ufps.edu.co/images/Descargas/pensum1155.pdf.

15. **Sparks, Geoffrey.** Enterprise Architect User Guide. [En línea] 3 de 2012. http://www.sparxsystems.com.au/bin/EAUserGuide.pdf.

16. **Oracle.** Página Web Oficial Netbeans UML. [En línea] Oracle, 27 de 04 de 2007. [Citado el: 15 de 10 de 2013.] http://plugins.netbeans.org/plugin/1801/netbeans-uml.

17. **Gamma, Erich, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley, 1994. 0-201-63361-2.

18. **Keith, Mike y Schincariol, Merrick.** *Pro JPA 2 Mastering the Java Persistence API.* New York : Apress, 2009. ISBN 9781430219569.

19. **StarUML.** StarUML 5.0 Guía de Usuario (Versión en Ingles). [En línea] http://staruml.sourceforge.net/docs/user-guide(en)/toc.html.

20. **Canós, Jose H., Letelier, Patricio y Penadés, Carmen Maria.** *Metodologías Ágiles para el desarrollo de software.* Valencia : Universidad Politécnica de Valencia.

21. **Geary, David y Horstmann, Cay.** *Core Java Server Faces.* Boston : Prentince Hall, 2010. ISBN 9780137012893.

22. **Civici, Catagay.** PrimeFaces Documentation. *PrimeFaces Guia de Usuario.* [En línea] 24 de 02 de 2013. [Citado el: 08 de 03 de 2013.] http://www.primefaces.org/documentation.html.

23. **Aluk, Deepak, Cupri, Jhon y Malks, Dan.** *Core J2EE Patterns Best Practices and Design Strategies.* California : Prentice Hall Professional, 2003. ISBN 0131422464.

24. **Dominguez, José Alberto.** UML: Lenguaje Unificado de Modelado. [En línea] 14 de Marzo de 2009. [Citado el: 8 de Octubre de 2011.] http://www.que-informatica.com/index.php/tag/uml.

25. **Ramirez, Alejandro, y otros.** ArgoUML User Manual: A tutorial and reference description. [En línea] 2011. http://argouml-downloads.tigris.org/nonav/argouml-0.34/manual-0.34.pdf.

26. **Group, OMG Object Management.** OMG Unified Modeling Language Superstructure. [En línea] 6 de Agosto de 2011. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/. formal/2011-08-06.

1. CASE (Computer Aided Software Engineering, Ingeniería de software asistida por computadora), son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo los costos del mismo en términos de tiempo, esfuerzo y dinero. [↑](#footnote-ref-1)
2. Página web oficial de Enterprise Architect, <http://www.sparxsystems.com/> [↑](#footnote-ref-2)
3. Tomado del portal web del programa Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, en la sección Presentación del Programa – Objeto de Estudio.

   Disponible en : <http://ingsistemas.ufps.edu.co/index.php/informacion-programa/objeto-de-estudio> [↑](#footnote-ref-3)
4. Cloud Computing, definición, características, tipos y ventajas, información disponible en la sección 2.2.2 del documento. [↑](#footnote-ref-4)
5. Google Drive es parte de la nube de servicios proporcionada por la empresa Google para sus usuarios, a través de esta nube los usuarios pueden disponer de un kit de herramientas entre los que se incluye un procesador de texto, hojas de cálculo, diseñador de presentaciones, entre otras. [↑](#footnote-ref-5)
6. Compañía Startup, hace referencia en el mundo empresarial a un nuevo negocio o ideas de negocio que están en construcción, es decir, son empresas emergentes apoyadas en la tecnología. [↑](#footnote-ref-6)
7. Backend se le llama a la parte de la aplicación que se ejecuta en el servidor, la cual está encargada de procesar los datos enviados desde el cliente (Frontend) y generar una respuesta. [↑](#footnote-ref-7)
8. API (“Application Programming Interface”, en español “Interfaz de programación de aplicaciones”), se refiere a un conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software. [↑](#footnote-ref-8)
9. GWT es un Framework para la construcción de aplicaciones web en lenguaje Java, el cual permite trabajan con la tecnología AJAX (*Asynchronous Javascript and XML*) de forma trasparente para el desarrollador, evitándole tener que lidiar con los aspectos complejos y tediosos de la tecnología; además de ser compatible con la mayoría de navegadores web, lo cual es verdaderamente significativo ya que el código escrito funciona igual para todos. [↑](#footnote-ref-9)
10. Más información en: <http://www.oracle.com/technetwork/java/javaee/documentation/index.html> [↑](#footnote-ref-10)
11. Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. [↑](#footnote-ref-11)
12. Release es una versión de lanzamiento del software, es decir, que el software se hace público. En ocasiones una versión del software puede ser una Release Candidate o candidata a lanzamiento; es decir, es una versión previa al lanzamiento definitivo de dicho software. [↑](#footnote-ref-12)
13. Universidad Francisco de Paula Santander. Estatuto estudiantil. Acuerdo 065, agosto 26, 1996.

    Disponible en : <http://www.ufps.edu.co/ufpsnuevo/archivos/reglamentacion/acuerdo065.pdf> [↑](#footnote-ref-13)
14. El Object Management Group u OMG (o por su nombre en español, Grupo de Gestión de Objetos) es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA. Es una organización sin ánimo de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas. El grupo está formado por diversas compañías y organizaciones con distintos privilegios dentro de la misma. [↑](#footnote-ref-14)
15. Github es una plataforma de desarrollo colaborativo que permite alojar proyectos de software utilizando el sistema de control de versiones Git. El código se almacena de forma pública, aunque también es posible obtener acceso de manera privada. [↑](#footnote-ref-15)
16. Un Clasificador representa una clase del metamodelo UML que tiene la capacidad de contener: Atributos, Operaciones y Dependencias con otros clasificadores. De esta clase heredan varias de los elementos de diagramas de clase, tales como, Clase, Interface y Tipo de Dato (que a su vez es la clase padre de una Enumeración). [↑](#footnote-ref-16)
17. DRY (Don’t Repeat Yourselft). Es un principio en el desarrollo de software enfocado en reducir la repetición de información de todo tipo, especialmente usado en arquitecturas multicapa. [↑](#footnote-ref-17)
18. RIA (Rich Internet Applications). Son aplicaciones web que tienen la mayoría de las características de las aplicaciones de escritorio tradicionales. Estas aplicaciones usan un navegador web estándar para ejecutarse y por medio de complementos o mediante una máquina virtual se agregan características adicionales. [↑](#footnote-ref-18)
19. POJO (acrónimo de Plan Old Java Object) es una sigla utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un Framework en especial. Este acrónimo surge como una reacción en el mundo Java a los Frameworks cada vez más complejos, y que requieren un complicado andamiaje que esconde el problema que realmente se está modelando. [↑](#footnote-ref-19)
20. Una Anotación Java es una forma de añadir metadatos al código fuente Java que están disponibles para la aplicación en tiempo de ejecución. Las anotaciones pueden añadirse a clases, métodos, campos, parámetros, variables locales y paquetes. Las anotaciones Java son completamente accesibles al programador mientras que el software se ejecuta usando *Reflexión*. Las anotaciones son comúnmente precedidas del carácter @. [↑](#footnote-ref-20)
21. Se denomina Connection Pool (agrupamiento de conexiones) al manejo de una colección de conexiones abiertas a una base de datos de manera que puedan ser reutilizadas al realizar múltiples consultas o actualizaciones (tomado de <http://es.wikipedia.org/wiki/Connection_pool>). [↑](#footnote-ref-21)
22. Inyección de Dependencias (en inglés Dependency Injection, DI) es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. [↑](#footnote-ref-22)
23. Para más información sobre TestNG puede consultar la página principal de la herramienta: http://testng.org/doc/index.html [↑](#footnote-ref-23)
24. EAR (Enterprise Archive) es un formato de archivo utilizado por Java EE para el empaquetado de uno o más módulos en un solo archivo para que el despliegue de los diferentes módulos en un servidor de aplicaciones que sucede al mismo tiempo y de manera coherente. También contiene los archivos XML llamados descriptores de despliegue que describen cómo implementar los módulos. [↑](#footnote-ref-24)