UNIVERSIDAD SIGLO 21

La educación evoluciona

Legajo: VINF013412

Alumna: Valeria Bijarra

Profesora: Ana Carolina Ferreyra

Materia: Seminario

Título: TP4- Sistema Gestión de Incidentes

Fecha: 30/06/2024

Contenido

1. Proyecto Sistema Gestión de Incidentes

1.1. Título del proyecto

Gestión de incidentes y base de soluciones de la empresa “OLEOSIN”.

1.2. Introducción

La empresa “OLEOSIN” es una empresa pequeña con 50 empleados, que se dedica a la producción de aceitunas y aceite de oliva. La empresa cuenta con un sistema de administración, de registro y administración de pedidos, uno para la administración (pagos y cobranzas), recursos humanos y otro de producción.

La empresa posee un pequeño departamento de sistemas, que se encarga de la resolución de problemas de los sistemas informáticos, provisión de hardware y resolución de problemas técnicos en impresoras, monitores, cpu, teléfonos, etc. Estos problemas no tienen un registro formal, sino que los usuarios lo hacen vía mail.

La empresa presenta una deficiente gestión de incidencias, causando demoras en los tiempos de atención, debido a que en la actualidad toda esta tarea se realiza de forma manual y no se cuenta con una base de datos con información de errores históricos.

Se solicita un sistema informático que permita administrar los incidentes que generan las diferentes áreas y, además, permita registrar documentación referente a la solución de problemas.

1.3. Justificación

Debido a la deficiente gestión de incidentes, la constante rotación de personal en el área de sistemas y las diferentes metodologías que cada analista funcional/técnico utiliza para documentar las soluciones, es que se solicitó el desarrollo de un sistema que permita registrar incidentes, documentar y hacer uso de la información de resolución de problemas ya almacenada, como una base de conocimiento tanto para usuarios como para personal del departamento de sistemas.

La relevancia del proyecto se extiende a nivel organizacional ya que los directivos (dueños de la empresa) solicitan constantemente mejorar los tiempos de respuesta del área de sistemas y evitar de esta forma demoras en los principales procesos de la empresa. El sistema brindará información relevante al usuario para que pueda resolver por sí mismo su inconveniente en caso de que la solución este a su alcance y no necesite ayuda de personal de sistemas. De esta forma evitar cuellos de botella en épocas donde la cantidad de errores/fallas aumenta, por ejemplo, en los cierres de periodos contables.

1.4. Definiciones del proyecto

a) Objetivo General del Proyecto

El objetivo general del proyecto es desarrollar un sistema de gestión de Incidentes, que permita un registro formal de problemas y soluciones aplicadas, para optimizar los tiempos de resolución en la empresa OLEOSIN.

Objetivo específico 1: Formalizar registro de incidentes. Permitirá el registro de problemas habituales (incidentes). De esta forma quedará centralizada la información de problemas por área.

Objetivo específico 2: Autoservicio de Mesa de Ayuda. El usuario podrá, en el mismo momento de la carga del incidente acceder a soluciones ya registradas, sobre problemas similares. De esta forma se podrán optimizar tiempos de resolución y evitar demoras en los procesos de la empresa.

Objetivo específico 3: Base de conocimiento. Registro de documentación detallada acerca de las soluciones técnicas y funcionales aplicadas a cada incidente. Permitiendo alimentar la base de conocimiento para la posterior consulta.

Diagrama de Gantt con las tareas a realizar:

Tareas					abr-24				may-24				jun-24			
Tarea	Incio	Fin	Duraci	Priorid	01-abr	08-abr	15-abr	22-abr	29-abr	06-may	13-may	20-may	27-may	03-jun	10-jun	17-jun
Analisis	**01/04/2024**	**12/04/2024**	**10**	1												
Relevamiento de Requerimientos	01/04/2024	05/04/2024	5	1												
Comunicación con interesados	08/04/2023	08/04/2024	1	3												
Documentación sistema actual	09/04/2024	10/04/2024	2	3												
Analisis/conclusiones	11/04/2024	12/04/2024	2	3												
Diseño	**15/04/2024**	**03/05/2024**	**14**	1												
Diseño de BD	15/04/2024	18/04/2024	4	1												
Diseño de aplicación	19/04/2024	02/05/2024	9	2												
Documentación de Diseño/Especi	03/05/2024	03/05/2024	1	3												
Desarrollo	**22/04/2024**	**17/05/2024**	**20**	1												
Modulo Incidentes	22/04/2024	30/04/2024	7	1												
Modulo de base de conocimiento	02/05/2024	10/05/2024	7	1												
Acceso y seguridad	13/05/2024	14/05/2024	2	3												
Integación	15/05/2024	16/05/2024	2	3												
Puebas tempranas	17/05/2024	20/05/2024	2	2												
Pruebas	**06/05/2024**	**24/05/2024**	**15**	1												
Pruebas usuario	06/05/2024	10/05/2024	5	1												
Relevamiento de errores	13/05/2024	14/05/2024	2	2												
Corrección de errores	15/05/2024	24/05/2024	8	1												
Implementación	**27/05/2024**	**14/06/2024**	**15**	1												
Instalación	27/05/2024	28/05/2024	2	1												
Mantenimiento	27/05/2024	14/06/2024	13	2												
Evaluación	**17/06/2024**	**21/06/2024**	**5**	3												

b) Objetivo General del Sistema

El objetivo general del sistema es centralizar el registro de incidentes y soluciones. Permitirá acceder a un repositorio de información con las soluciones, este será de fácil acceso y consulta tanto para usuarios como para personal del área de sistemas. La organización de la información permitirá mejorar los tiempos de resolución.

b.1. Restricciones:

- El acceso a la información será restringido, dependerá del rol del usuario y de su área.
- El sistema debe tener una interfaz de uso intuitiva.
- El sistema debe tener un diseño e implementación sencilla.
- Lenguaje de programación Java.
- Los servidores deben ser capaces de atender consultas concurrentemente.
- El sistema se diseñará según un modelo cliente servidor.

b.2. Alcance:

- Carga / Administración de Incidentes.
- Registro de Soluciones.
- Acceso a Soluciones para usuarios y resolutores.
- Envío de notificaciones por mail a Sistemas, sobre los problemas que surjan de las áreas.
- Reportearía para toma de decisiones.
- Configuración inicial.

b.3. Fuera de Alcance:

- No contempla migración de soluciones.
- Manejo de SLA
- Solicitudes de Mejora

1.5. Elicitación

A través de este proceso de adquisición de conocimiento se aplicarán las técnicas necesarias para entender mejor el negocio donde se desarrollará el proyecto. Permitirá identificar interesados y definir los requerimientos. Algunos ejemplos de documentos usados en el proceso de Elicitación son: entrevista, respuestas de cuestionarios, actas de reunión, grabaciones de audio y video. Todos estos elementos permiten generar información no estructurada para aplicar directamente en la solución luego de sintetizar, analizar, eliminar redundancias, descubrir errores y omisiones. Pasos en la Elicitación:

1.5.1.Preparar la Elicitación:

Es necesario tener la siguiente información: Alcance de la solución, lista de roles / interesados, documentación preexistente.
Interesados:

- Usuarios supervisores de procesos troncales de la empresa que habitualmente informan incidentes
- Analistas Funcionales/Tecnicos que alimentarán la base de conocimiento
- Jefe de sistemas

Organizar y reservar los recursos necesarios para la Elicitación. Ej: Agendas, disponibilidad de salas de reunión, planificación de viajes, logística. Es clave para obtener un buen resultado en la Elicitación.
En este caso, se consideran realizar reuniones con supervisores de las áreas. En estas reuniones se relevarán datos que habitualmente informan cuando se les presenta un problema. La reunión será presencial en la empresa.

Además, se llevarán a cabo con los Analistas Funcionales y personal de Mesa de Ayuda del departamento de sistemas. En estas reuniones, se relevarán los requerimientos funcionales al momento de documentar las soluciones. Las reuniones serán via meet.

Se pactarán 1 reunión con el Jefe de sistemas para detectar indicadores que necesita medir. Esta reunión será vía meet.

También es necesario definir la técnica de Elicitación. Esta puede depender de diferentes factores: dominio del negocio, cultura empresarial, habilidades del equipo, recursos disponibles, actividad o del tipo de proceso. En la práctica se suelen utilizar más de una técnica. Para este caso se considerará entrevista con los supervisores con preguntas abiertas y técnica de observación en el área de sistemas, considerando que quien releva tiene conocimiento de las problemáticas habituales, en aplicaciones que administran incidentes y bases de conocimiento.

1.5.2. Ejecutar Elicitación

Levantar junto con los interesados sus necesidades usando las técnicas mencionadas en el primer punto.
Requisitos implícitos: profundizar conocimiento en el negocio ayudará a detectarlos
Es muy importante que no se desvíe el alcance del proyecto.

Cuestionario Usuarios/Supervisores	
Proyecto	Gestión de Incidentes
Fecha/Hora	01/04/2024
Lugar	Directorio
Entrevistador	V.Bijarra
Parte I: Identificando perfil de usuario/cliente	
Áreas	Producción/Finanzas/Comercial/RH
Roles en organización	Supervisores
Responsabilidades	Control de ejecución de procesos de la empresa
Parte II: Evaluación del problema	
Cómo reportan errores al área de IT?	Todas las áreas: Se realiza envío de mail o si es muy urgente llaman por teléfono
Cómo sabe a quién dirigir la solicitud?	Todas las áreas: Si es problema técnico a soporte de equipos tecnológicos. Si el error está en la aplicación al registrar información producción y comercial: Analista1. Si el error está en la aplicación al registrar información finanzas, Logística, Recursos Humanos: Analista2. Todas las áreas: Si el error es en la computadora o impresora, o por alta de usuario de red: IT.
Que información reporta?	Todas las áreas: Describen brevemente el problema por mail y aguardan a ser contactados. A veces envían anexo una imagen con el error.
Posee algún registro de errores comunes en su área?	Supervisor del área Producción: No, no poseemos. A veces, en las ausencias por vacaciones, se prepara un mail para que los compañeros que quedan en el puesto de trabajo sepan cómo actuar si aparece algún error. Supervisor Comercial/RH: Llevamos registro en Excel. Supervisor Financiero: Registro en cuadernos/apuntes manuales.
Qué tipo de errores son frecuentes en su área?	Supervisor Producción: Errores con la impresora o de red, lentitud. Errores de configuración de materiales o errores al registrar carga de producción. Supervisor Financiero: Errores de cálculo de impuestos, generación de pagos, error en configuración de clientes o proveedores, error en cierre contable.

	Supervisor Comercial: Errores en carga de pedidos, errores en la carda de los despachos/facturación, errores con listas de precios/actualización de listas de precios. Todas las áreas: También es habitual errores de clave en algunos colegas.
Parte III: Entendiendo el Entorno	
¿Quiénes serán los usuarios del Sistema?	Solo los encargados de cada área.
Tienen los usuarios experiencia en este tipo de aplicaciones	No.
Qué tipo de ayuda requerirá el usuario (ayuda online,…)?	Dependiendo el problema puede ser remoto o in situ.
Parte IV: Evaluando la oportunidad	
¿Quién en la organización necesita la aplicación?	Los encargados de cada área para informar los problemas que surgen.
¿Cuántos tipos de usuarios usarán la aplicación?	Solo los encargados
¿Cómo valoraría que la solución ha sido un éxito?	Si el soporte llega rápido entonces será un éxito. Si podemos acceder a recomendaciones respecto al problema que estamos informando, para acelerar los tiempos de resolución.

Firma/Aclaración/Lugar/Fecha

Cuestionario Personal de Sistemas	
Proyecto	Gestión de Incidentes
Fecha/Hora	02/04/2024
Lugar	Meet
Entrevistador	V. Bijarra
Parte I: Identificando perfil de usuario/cliente	
Nombre del entrevistado	Andrés Gutiérrez
Rol en organización	Analista I
Responsabilidades del entrevistado	Dar soporte funcional en los sistemas de producción y comercial
Parte II: Evaluación del problema	
Cómo registran las soluciones a los diferentes problemas?	En una planilla Excel

Se le otorga información relevante al usuario respecto a la solución?	No, solo en casos de que el error sea por configuración errónea, o en algunos casos solo se le indica que el problema fue resuelto.
Que información reporta/Documenta?	En un Excel el error y la solución otorgada, a veces agrego registro de pantallas.
Posee algún registro de errores comunes en su área?	Poseemos una carpeta por usuario, donde almacenamos nuestros documentos. Pero es personal, mis compañeros no pueden acceder a mi información.
Que información es necesaria contabilizar?	Cantidad de problemas por área y sistema.
¿Existe alguna categorización de problemas? Cual?	Sistema/Modulo/Parametrización/Bug.
Parte III: Entendiendo el Entorno	
¿Quiénes serán los usuarios del Sistema?	Todos en mi área.
Tienen los usuarios experiencia en este tipo de aplicaciones	Sí. Es necesario poder centralizar los errores en un sistema, y poder registrar las soluciones.
Cuáles son las expectativas de usabilidad del Producto	Que sea intuitivo para la carga de los incidentes, que se logre recolectar la mayor cantidad de información para comprender el problema del usuario. También es necesario que la carga y el acceso a las soluciones sea intuitiva.
Qué tipo de ayuda requerirá el usuario (ayuda online,…)?	El usuario va a requerir ayuda por errores, o por falta de conocimiento en los procesos del sistema. Puede que sea necesario contactarlo remotamente.
Parte IV: Evaluando la oportunidad	
¿Quién en la organización necesita la aplicación?	Usuarios de todas las áreas de la empresa y del área de sistemas.
¿Cuántos tipos de usuarios usarán la aplicación?	Serían 3 tipos de usuarios: Usuarios de las diferentes áreas de la empresa, Analistas y Técnicos de IT
¿Cómo valoraría que la solución ha sido un éxito?	Si los usuarios realmente usan el sistema es una forma de valorar y que no envíen más correos con problemas. Además, lograr que la consulta de las soluciones sea rápida y precisa para optimizar los tiempos de resolución.

Firma/Aclaración/Lugar/Fecha

Cuestionario Personal de Sistemas	
Proyecto	Gestión de Incidentes
Fecha/Hora	03/04/2024
Lugar	Meet

Entrevistador	V. Bijarra
Parte I: Identificando perfil de usuario/cliente	
Nombre del entrevistado	Soledad Ortiz
Rol en organización	Analista II
Responsabilidades del entrevistado	Dar soporte funcional en los sistemas de Finanzas, Logística y Recursos Humanos
Parte II: Evaluación del problema	
Cómo registran las soluciones a los diferentes problemas?	En una planilla Excel En un directorio por sistema y módulo almaceno algunos documentos Word con el detalle de la solución aplicada
Se le otorga información relevante al usuario respecto a la solución?	Sí, siempre indico lo que se realizó para resolver el problema. Sobre todo, si el usuario puede resolverlo el mismo la próxima vez que suceda.
Que información reporta/Documenta?	En un Excel el error y la solución otorgada y si es necesario más detalle organizo la información en un Word
Posee algún registro de errores comunes en su área?	Poseemos una carpeta por usuario, donde almacenamos nuestros documentos. Pero es personal, mis compañeros no pueden acceder a mi información.
Que información es necesaria contabilizar?	Cantidad de problemas resueltos por área, con distinción del tipo (parametrización, error, falta e conocimiento del usuario, etc.)
¿Existe alguna categorización de problemas? Cual?	Sistema/Modulo/Parametrización/Bug/Error de diseño/Nueva funcionalidad.
Parte III: Entendiendo el Entorno	
¿Quiénes serán los usuarios del Sistema?	Todos en mi área.
Tienen los usuarios experiencia en este tipo de aplicaciones	Sí. En otra empresa teníamos un sistema que permitía acceder a todas las soluciones otorgadas de forma rápida por palabras o categoría de problema.
Cuáles son las expectativas de usabilidad del Producto	Poder contar con un repositorio de información que sirva de consulta para mí y mi equipo.
Qué tipo de ayuda requerirá el usuario (ayuda online,…)?	El usuario va a requerir ayuda por errores, o por falta de conocimiento en los procesos del sistema. Puede que sea necesario contactarlo remotamente.
Parte IV: Evaluando la oportunidad	
¿Quién en la organización necesita la aplicación?	Usuarios de todas las áreas de la empresa y del área de sistemas.
¿Cuántos tipos de usuarios usarán la aplicación?	Serían 3 tipos de usuarios: Usuarios de las diferentes áreas de la empresa, Analistas y Técnicos de IT
¿Cómo valoraría que la solución ha sido un éxito?	Lograr reducir los tiempos de solución de problemas, haciendo uso de soluciones otorgadas en otras ocasiones.

Firma/Aclaración/Lugar/Fecha

Cuestionario Personal de IT	
Proyecto	Gestión de Incidentes
Fecha/Hora	04/04/2024
Lugar	Meet
Entrevistador	V. Bijarra
Parte I: Identificando perfil de usuario/cliente	
Nombre del entrevistado	Enrique Álvarez
Rol en organización	Técnico IT
Responsabilidades del entrevistado	Dar soporte técnico a toda la empresa
Parte II: Evaluación del problema	
Cómo registran las soluciones a los diferentes problemas?	No tengo repositorio de soluciones
Se le otorga información relevante al usuario respecto a la solución?	No, porque no entendería. En algunas ocasiones son terminologías muy técnicas.
Que información reporta/Documenta?	No tengo repositorio.
Posee algún registro de errores comunes en su área?	No.
Que información es necesaria contabilizar?	Cantidad de altas de usuarios, Cantidad de entrega de dispositivos.
¿Existe alguna categorización de problemas? Cual?	Tipo de dispositivo con el problema (telefonía, pc, notebook, celular, impresora, etc) Error, falla, red
Parte III: Entendiendo el Entorno	
¿Quiénes serán los usuarios del Sistema?	Todos en mi área.
Tienen los usuarios experiencia en este tipo de aplicaciones	Sí. En otra empresa recibía los reclamos por ticket y cuando resolvía el inconveniente registraba la solución. Esta solución.
Cuáles son las expectativas de usabilidad del Producto	Poder centralizar la información, en esta aplicación, que sea intuitiva y permita realizar trazabilidad de los incidentes.
Qué tipo de ayuda requerirá el usuario (ayuda online,…)?	En la mayoría de los casos el soporte es en el área que tiene el problema. Puede que sea necesario contactar remotamente.
Parte IV: Evaluando la oportunidad	
¿Quién en la organización necesita la aplicación?	Usuarios de todas las áreas de la empresa y del área de sistemas.
¿Cuántos tipos de usuarios usarán la aplicación?	Serían 3 tipos de usuarios: Usuarios de las diferentes áreas de la empresa, Analistas y Técnicos de IT
¿Cómo valoraría que la solución ha sido un éxito?	Al mejorar la organización de los incidentes.

Firma/Aclaración/Lugar/Fecha

Cuestionario Jefe de Sistemas/IT	
Proyecto	Gestión de Incidentes
Fecha/Hora	05/04/2024
Lugar	Meet
Entrevistador	V. Bijarra
Parte I: Identificando perfil de usuario/cliente	
Nombre del entrevistado	Marcelo Méndez
Rol en organización	Jefe
Responsabilidades del entrevistado	Supervisar las áreas de Sistemas/IT
Parte II: Evaluación del problema	
Cómo registran las soluciones a los diferentes problemas?	Tengo un directorio de problemas habituales que uso y completo con información cuando algunos de mis empleados a cargo están de vacaciones y debo dar soporte.
Se le otorga información relevante al usuario respecto a la solución?	Sí, con lenguaje que pueda ser interpretado por el usuario.
Que información reporta/Documenta?	Documentos Word con descripción del error y la solución otorgada.
Posee algún registro de errores comunes en su área?	No. Este directorio que tengo es personal, no está compartido.
Que información es necesaria contabilizar?	-Problemas recurrentes -Cantidad de tipos de problemas un rating -Cantidades de incidentes resueltos por analista -Tiempo promedio de resolución -Sistemas con mayor cantidad de errores -Cantidad de soluciones definitivas
¿Existe alguna categorización de problemas? Cual?	-De sistemas o de IT -Sistema / Modulo -Error/Falla/Configuración/Capacitación -Alta/Baja usuarios -Hardware
Parte III: Entendiendo el Entorno	
¿Quiénes serán los usuarios del Sistema?	Todos en mi área.
Tienen los usuarios experiencia en este tipo de aplicaciones	Sí. En otra empresa recibía los reclamos por ticket y las soluciones podían ser visualizadas por todo el equipo de sistemas..

Cuáles son las expectativas de usabilidad del Producto	Poder centralizar la información, en esta aplicación, que sea intuitiva y poder acceder a las soluciones de forma rápida.
Qué tipo de ayuda requerirá el usuario (ayuda online,…)?	-Remoto -Telefónico
Parte IV: Evaluando la oportunidad	
¿Quién en la organización necesita la aplicación?	Usuarios de todas las áreas de la empresa y del área de sistemas.
¿Cuántos tipos de usuarios usarán la aplicación?	Serían 3 tipos de usuarios: Usuarios de las diferentes áreas de la empresa, Analistas y Técnicos de IT
¿Cómo valoraría que la solución ha sido un éxito?	Al mejorar la consulta de las soluciones de forma rápida y precisa.

Firma/Aclaración/Lugar/Fecha

Ficha de Observación

Ficha de observación		
Elaborado por	Valeria Bijarra	
Lugar	Oficina de Sistemas	
Área	IT	
Guía de Observación		
Aspectos a observar y registrar en materia técnica-ejecución	¿Cuáles son las principales actividades que se desempeñan en el área?	-Análisis Errores/Fallas en los errores que aparecen en los sistemas de la empresa. -Soporte/Capacitación -Creación de nuevos usuarios -Diseño de reportes -Atención telefónica a usuarios -Atención remota vía -Entrega de hardware/teléfonos
	¿Cuáles son los principales actores intervinientes en el proceso y qué labor cumplen?	Los analistas realizan atendimiento de problemas en los sistemas de la empresa. El técnico de IT da soporte relacionado al hardware, red y telefonía. El jefe supervisa la tarea de sus subordinados, pero no tiene participación activa en la solución de los incidentes. Solo en caso de reemplazo por vacaciones.
	¿Cuáles son los principales problemas que	-No poseen directorio compartido donde almacenar la información de las soluciones otorgadas. - Les cuesta organizar su trabajo y priorizar.

	afectan los procesos?	- Poseen información registrada en diferentes repositorios y planillas.
	¿Qué tipo de problemas con relación a los sistemas de información se observan?	-Los usuarios no entregan toda la información sobre el problema, en el mail que envían al departamento de sistemas. En la mayoría de los casos es necesario realizar una llamada telefónica para indagar más. -Sí el analista o técnico al que le envían el mail está ausente, el problema quedará sin resolver hasta que el usuario regrese.
	¿Cuáles son los factores que más debilitan –en el manejo articulado del proceso- el resultado final?	-No contar con información centralizada. -Que los incidentes lleguen por mail dificulta la administración de los problemas y la definición de prioridades.
Aspectos a observar cuando el usuario detecta un error que no conoce	¿Sabe cómo proceder cuando aparece un error?	-Sí, los usuarios saben a quién direccionar cada problema.
	¿El usuario prefiere buscar soluciones posibles a su problema sin intervención del departamento de sistemas?	-No, en la mayoría de los casos necesitan la supervisión de personal de sistemas, aun sabiendo que la solución la puede aplicar el usuario directamente.
	¿Qué información relevante entrega el usuario al departamento de sistemas?	-Nombre de Sistema, modulo, pantalla, acción que no pudo realizar y error. En algunos casos se anexa pantalla.
	¿Acostumbra detallar? ¿Anexa información?	Pocas veces
	¿Tienen en claro como tipificar el problema?	No siempre
	¿Tienen en claro como priorizar el problema?	En la mayoría de los casos los usuarios manifiestan urgencia, pero no siempre es real o prioritario.
Aspectos referentes a la solución (para Usuario)	¿Qué información espera el usuario?	Necesita saber que su problema se resolvió.
	¿El usuario necesita conocer horas insumidas en la solución?	No
	¿Es necesario registrar documentos con	Sí siempre que el usuario interprete de que se habla, puede servir como soporte para cuando el problema se vuelva a repetir.

	mayor detalle en la solución?	
	¿Es necesario registrar si solución es definitiva o temporal e informarla al usuario?	Sí, es necesario, de esta forma el usuario puede interpretar si puede repetirse nuevamente.
Aspectos referentes a la solución (para Dto de Sistemas)	¿Qué información necesita ingresar el departamento de sistemas?	Detalla de la solución Anexar documentación de respaldo (técnica y funcional)
	¿Qué información necesita visualizar el departamento de sistemas del problema cargado por el usuario?	Problema detallado, pasó a paso para reproducir el error, usuario, área, fecha, prioridad.
	¿Es necesario registrar si la solución es definitiva o temporal e informarla al usuario?	Sí, es necesario para saber si el problema puede repetirse.
	¿Es necesario registrar horas insumidas en la solución?	Sí, el Jefe de sistemas necesita conocer las horas insumidas para cada tipo de problema.
	¿Es necesario registrar documentos con mayor detalle en la solución?	Sí, que sirvan de respaldo para resolver el mismo problema en caso de que se repita,
	Es necesario realizar mediciones de las diferentes soluciones?	Sí, es necesario para el jefe de sistemas, para saber cantidades de errores por sistema, por área, por tipo de problema que le ayuden a la toma de decisiones.
	¿Es necesario diferenciar soluciones que pueden ser interpretadas por los usuarios de las que no?	Si, para saber cuáles pueden ser resueltas por el usuario sin intervención de sistemas

1.5.3.Documentar resultados de Elicitación

Documentos como informes y actas de reunión, cuadros con anotaciones, registros de audio y video. Se registran las planillas usadas en las entrevistas y ficha de observación.

1.5.4. Confirmar resultados de Elicitación

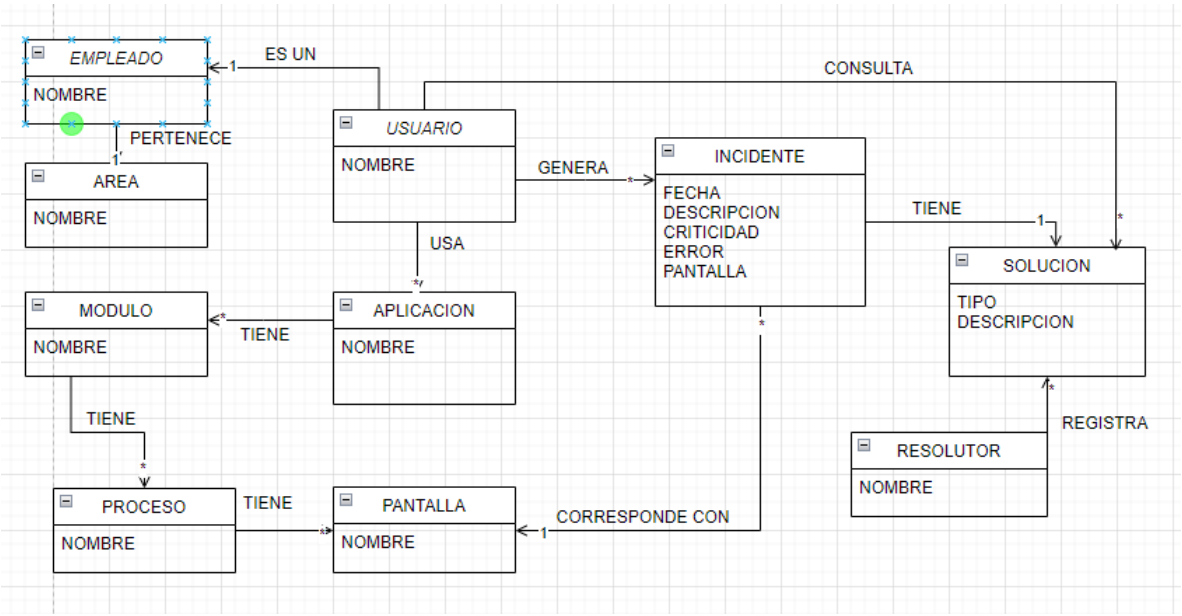
Es importante revisar las planillas de entrevista y ficha de observación con los interesados para garantizar que la comprensión del analista que realiza la Elicitación, está alineada con los deseos del usuario. Esto es muy importante, ya que los problemas de comunicación representan la mayor parte de los inconvenientes en la ingeniería de requerimientos.
Para llevar a cabo esta confirmación se realizarán una reunión con los diferentes interesados (usuarios del sistema) se revisarán los documentos con el resumen de lo relevado. Estos documentos deberán ser firmados como confirmación de conformidad.

1.6. Conocimiento del negocio

Actualmente la empresa, no cuenta con un sistema de incidentes y los usuarios comunican los problemas al área de sistemas mediante un mail. El destinatario del mail, es quien el usuario considere que puede resolver su problema.

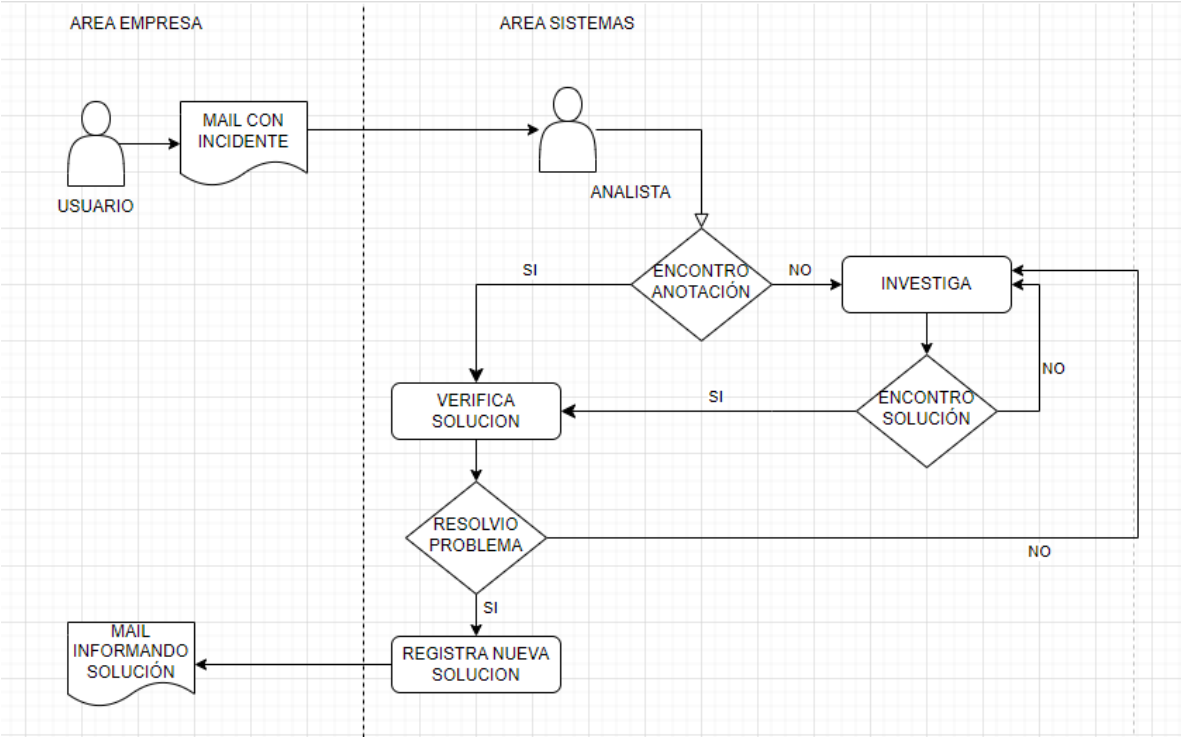
El área de sistemas tiene problemas para administrar los reclamos que reciben por mail, no pueden priorizar. Tampoco disponen de una herramienta que les permita unificar la metodología, para centralizar la información referente a las soluciones otorgadas.

1.6.1. Diagrama de Dominio



Proceso	**Informar Incidente**
Roles	Usuarios Clave
Pasos	- El usuario detecta un error - El usuario envía mail al departamento de sistemas - El usuario aguarda que le informen, que su problema fue resuelto

Proceso	**Recepción del Incidente / Tratamiento**
Roles	Analistas Funcionales / Personal de IT
Pasos	- El analista recibe el correo con el incidente - El analista busca entre sus anotaciones las soluciones posibles - Si el analista encontró solución entre sus anotaciones la aplica en entorno de pruebas previo a informarle al usuario. - Si se logró solucionar el inconveniente se notifica al usuario por mail - Si el analista no encuentra entre sus anotaciones soluciones, entonces analiza/investiga, prueba. Al encontrar nueva solución actualiza información de sus anotaciones e informa al usuario que su problema quedó resuelto.



El proceso descripto tiene las siguientes falencias:

- No hay seguimiento de incidentes.
- No se pueden emitir informes de errores, tipos de errores, áreas con mayor cantidad de incidentes, aplicaciones con mayor cantidad de errores, etc.
- Al reportar errores por mail no todos los integrantes del departamento de sistemas están al tanto de los problemas. En caso de que el analista/técnico no vea el mail este quedará sin atendimiento.
- No hay priorización de problemas
- No hay registro de soluciones aplicadas

- Los diferentes analistas tienen diferentes formas de registrar soluciones. Posiblemente exista información duplicada ocupando espacio.

1.7. Propuesta de solución

1.7.1 Propuesta Funcional:

Desarrollar una aplicación que permita la gestión de incidentes como centro de servicios de IT. El objetivo es acelerar tiempos de resolución, facilitar la tarea a los equipos de soporte y lograr mejorar la satisfacción del cliente. A partir de una buena categorización de incidentes se podrá implementar una especie de autoservicio, para que el usuario disponga de las soluciones, sin intervención de terceros.

La aplicación ofrecerá:

- ✓ **Módulo de registro de incidentes**: El usuario podrá registrar sus problemas con formularios preestablecidos dependiendo el tipo de problema/aplicación.
- ✓ **Módulo de Soluciones:** El personal de sistemas, al momento de dar tratamiento/solución al problema, podrá documentar y tipificar la solución entregada.
- ✓ **Módulo de reportes**: Para gestión del Jefe de sistemas, seguimiento de problemas habituales por aplicación/módulo y tiempos de resolución.

1.7.2. Propuesta Técnica

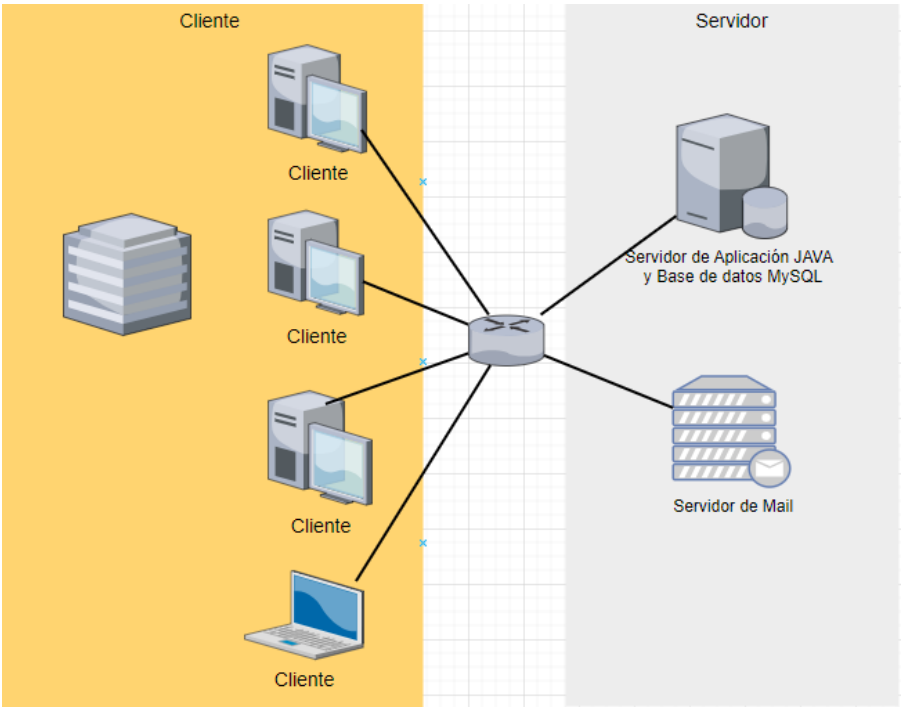
: El desarrollo del sistema de incidentes y base de conocimiento se va a realizar en Java utilizando la herramienta Eclipse, que cuenta con una gran variedad de bibliotecas y frameworks que permitirán escalamiento. La persistencia se realizará en MySQL que es una base de datos relacional abierta, flexible y de alto rendimiento para el almacenamiento de datos en tiempo real.

Se va a emplear JDBC (Java Database Connectivity) para consultar y realizar la persistencia de los datos desde Java. Esta API (interfaz de programación de aplicaciones) estándar permite a las aplicaciones Java interactuar con bases de datos.

La arquitectura propuesta sigue el modelo cliente servidor, considerando un servidor para la aplicación y Base de datos MySql y un servidor de mails.

La tecnología de comunicación a utilizar será la misma que hasta ahora, mediante un cableado Ethernet, que proporciona una infraestructura confiable y eficiente que es esencial para la comunicación en la empresa.

Diagrama de Arquitectura



1.8. Requerimientos

Después de analizar la problemática de la empresa “OLEOSIN”, considerando lo relevado en el proceso de Elicitación se identifican las siguientes necesidades:

1.8.1. Requerimientos Funcionales

Requerimiento del sistema	Descripción
RFS-LOG01	El sistema debe permitir al usuario, administrador del sistema y resolutores acceso mediante usuario y clave.
RFS-CONF01	El sistema debe permitir al administrador del sistema registrar aplicaciones.
RFS-CONF02	El sistema debe permitir al administrador del sistema registrar tipos de problemas.
RFS-CONF03	El sistema debe permitir al administrador del sistema registrar prioridades de un incidente.
RFS-CONF04	El sistema debe permitir al administrador del sistema registrar estados de incidentes.
RFS-CONF05	El sistema debe permitir al administrador del sistema registrar áreas.
RFS-CONF06	El sistema debe permitir al administrador del sistema registrar resolutores.
RFS-CONF07	El sistema debe permitir al administrador del sistema vincular usuarios con un área.
RFS-CONF08	El sistema debe permitir al administrador del sistema registrar Módulos de cada aplicación (sistemas que usan en la empresa)

RFS-CONF9	El sistema debe permitir al administrador del sistema registrar Procesos para cada módulo.
RFS-CONF10	El sistema debe permitir al administrador del sistema registrar Pantallas vinculadas para cada proceso del módulo.
RFS-CONF11	El sistema debe permitir al administrador del sistema vincular los problemas a cada Pantalla.
RFS-CONF12	El sistema debe permitir al administrador del sistema vincular resolutores con aplicaciones.
RFS-CONF13	El sistema debe permitir al administrador del sistema configurar motivos de cancelación de incidentes.
RFS-INC01	El sistema debe permitir al usuario registrar un incidente
RFS-INC02	El sistema debe establecer formulario de carga de incidente según aplicación y tipo de problema.
RFS-INC03	El sistema debe mostrar al usuario, luego de registrar el incidente, si existen o no soluciones aplicadas a incidentes similares. Solo soluciones que pueda visualizar el usuario.
RFS-SOL01	El sistema debe permitir al usuario consultar la base de soluciones. Solo soluciones que pueda visualizar el usuario.
RFS-INC04	El sistema debe permitir al usuario modificar un incidente.
RFS-INC05	El sistema debe permitir al usuario cancelar un incidente y seleccionar motivo de cancelación.
RFS-INC06	El sistema debe enviar mail a resolutores (área de sistemas) ante un nuevo incidente.
RFS-INC07	El sistema debe enviar mail a resolutores cuando el usuario cancele un incidente.
RFS-INC08	El sistema debe permitir a los usuario listar incidentes de su área.
RFS-INC09	El sistema debe permitir a los resolutores listar incidentes de todas las áreas por estado.
RFS-INC10	El resolutor puede asignarse el Incidente.
RFS-INC11	El sistema debe permitir a los resolutores solucionar un incidente.
RFS-SOL02	El sistema debe permitir a los resolutores registrar una nueva solución en la base de conocimiento.
RFS-SOL03	El sistema debe permitir a los resolutores marcar solución que pueda ser visualizada por el usuario.
RFS-SOL04	El sistema debe enviar mail al usuario informando que su incidente fue resuelto.
RFS-SOL05	El sistema debe listar a los resolutores las soluciones registradas.

1.8.2. Requerimientos No Funcionales

Requerimiento del sistema	Descripción
RNFS-APP01	El sistema debe estar desarrollado en Java.
RNFS-APP02	El sistema debe tener una disponibilidad de 5x24 con momentos de inactividad los fin de semana ya que la planta no produce.
RNFS-APP03	El sistema debe ser escalable para nuevos usuarios, aplicaciones y tipos de problemas.
RNFS-BD01	El sistema debe contar con una base de datos MySQL.
RNFS-BD02	Para la persistencia y consulta de datos en la BD se debe usar un patrón MVC (modelo-vista-controlador).
RNFS-SERV06	El sistema debe estar instalado en servidor de la empresa “OLEOSIN”.

1.8.3. Requerimientos Candidatos

Requerimiento del sistema	Descripción
RFSC-SOL01	El sistema permitirá administración de solicitudes de mejoras en los sistemas (proyectos).
RFSC-INC01	El sistema permitirá la configuración de SLA en incidentes.

1.9. Análisis

1.9.1. Diagrama de caso de uso

1.9.2. Identificación de actores

Usuario: Es quien generará nuevos Incidentes en el sistema y será informado cuando el incidente se resuelva.

Resolutor: Del área de Sistemas. Es quien trabajará en cada Incidente cargado por el usuario y le dará una solución. Registrará soluciones otorgadas.

Jefe de Sistemas/Administrador: Es quien supervisará cantidad de incidentes que llegan al área de sistemas para tomar decisiones, para optimizar los tiempos de resolución. Además, puede resolver en caso de que sea necesario (por ejemplo, en época de vacaciones de sus compañeros) incidentes y registrar soluciones en la base de conocimiento.
Realiza toda la parametrización del sistema.

1.9.3. Trazabilidad

Requerimiento	Caso de Uso	Actor	Paquete del Análisis	Comentario
RFS-INC01	CU01	Usuario	Incidente	Crear Nuevo Incidente.
RFS-INC02	CU01	Usuario	Incidente	Cargar formulario de incidente según aplicación.
RFS-INC03	CU01	Usuario	Incidente	Informar si existe solución existente.
RFS-SOL01	CU01	Usuario	Incidente	Obtener Soluciones que pueda visualizar el Usuario
RFS-INC06	CU01	Usuario	Incidente	Informar a resolutores creación de nuevo incidente.
RFS-INC05	CU02	Usuario	Incidente	Cancelar Incidente.
RFS-INC07	CU02	Usuario	Incidente	Informar a Sistemas cancelación de incidente.
RFS-INC04	CU03	Usuario	Incidente	Modificar información del incidente.
RFS-INC10	CU03	Resolutor	Incidente	Modificar responsable del incidente.
RFS-INC11	CU04	Resolutor/Jefe de Sistemas	Incidentes	Cerrar Incidente.
RFS-SOL03	CU04	Resolutor/Jefe de Sistemas	Incidentes	Indicar solución de usuario al cerrar Incidente.
RFS-SOL04	CU04	Resolutor/Jefe de Sistemas	Incidentes	Informar Cierre de Incidente.
RFS-SOL02	CU04	Resolutor/Jefe de Sistemas	Soluciones	Registrar nueva Solución.
RFS-SOL05	CU05	Resolutor/Jefe de Sistemas	Soluciones	Consultar todas las soluciones (de tipo usuarios y técnicas).
RFS-INC08	CU06	Usuario	Consulta Incidentes	Listar Incidentes del área del usuario por estado.
RFS-INC09	CU06	Resolutor/Jefe de Sistemas	Consulta Incidentes	Listar Incidentes de todas las áreas por estado.
RFS-CONF01	CU07	Jefe de Sistemas - Admin	Configuración	Registrar aplicaciones.
RFS-CONF02	CU07	Jefe de Sistemas - Admin	Configuración	Registrar tipos de problemas.
RFS-CONF03	CU07	Jefe de Sistemas - Admin	Configuración	Registrar prioridades.
RFS-CONF04	CU07	Jefe de Sistemas - Admin	Configuración	Registrar Estados.
RFS-CONF05	CU07	Jefe de Sistemas - Admin	Configuración	Registrar Areas.
RFS-CONF06	CU07	Jefe de Sistemas - Admin	Configuración	Registrar Resolutores.
RFS-CONF07	CU07	Jefe de Sistemas - Admin	Configuración	Vincular usuarios con Área.
RFS-CONF08	CU07	Jefe de Sistemas - Admin	Configuración	Registrar módulos con Aplicación.
RFS-CONF09	CU07	Jefe de Sistemas - Admin	Configuración	Registarr Procesos con cada Módulo.
RFS-CONF10	CU07	Jefe de Sistemas - Admin	Configuración	Registrar Pantallas con Procesos.

RFS-CONF11	CU07	Jefe de Sistemas - Admin	Configuración	Registrar Problemas con Pantallas.
RFS-CONF12	CU07	Jefe de Sistemas - Admin	Configuración	Vincular resolutores con Aplicaciones.
RFS-CONF13	CU07	Jefe de Sistemas - Admin	Configuración	Regsitrar Motivos de cancelación.

1.9.4. Descripción de casos de uso

Caso de Uso	CU01-Crear nuevo Incidente	
Actores	Usuario	
Referencia	RFS-INC01, RFS-INC02, RFS-INC03, RFS-SOL01, RFS-INC06	
Descripción	Permite al usuario agregar un nuevo incidente.	
Precondición	-El usuario ingresó al sistema con usuario y clave. -El usuario tiene permisos para crear nuevos incidentes. -El sistema debe tener cargadas listas de aplicaciones, módulos, procesos, pantallas, tipo de problema, prioridades, estados, áreas y usuarios vinculados a un área.	
Flujo Principal	1	El usuario selecciona la opción “Nuevo Incidente” en la interfaz de usuario.
	2	El sistema presenta el formulario vacío. La información que se debe proporcionar es: • Aplicación (Lista) • Módulo (Lista) • Proceso(Lista) • Pantalla(Lista) • Tipo de problema (Lista) • Código de error • Prioridad (Lista) • Descripción corta • Descripción Detallada • Anexo El usuario confirma la entrada de datos
	3	El sistema valida la información ingresada por el usuario. (S1)
	4	El sistema asigna número de Incidente (correlativo) y es mostrado por pantalla.
	5	El sistema completa fecha de registro, usuario y estado “Nuevo” en los campos correspondientes.
	6	El sistema agrega el incidente a la base de datos.
	7	El sistema muestra un mensaje de confirmación, indicando que el incidente fue registrado con éxito.
	8	El sistema verifica si existen soluciones disponibles. Si encuentra coincidencias muestra mensaje al usuario.
	9	Muestra soluciones encontradas en la base de soluciones. (S2)
	10	El sistema emite mensaje al área de sistemas informando creación de un nuevo incidente.
Postcondición	Se agregó un nuevo incidente con la información proporcionada por el usuario.	
Flujo Alternativo	S1	Validación de Datos con error: 1- El sistema muestra mensaje con error.

		2- No permite continuar hasta que la información es corregida. Se regresa al punto de carga de formulario.
	S2	Si el usuario encuentra solución a su problema acciona el caso de uso: “Cancelar Incidente”.
Excepciones		No contempla

Caso de Uso	CU02-Cancelar Incidente	
Actores	Usuario	
Referencia	RFS-INC05, RFS-INC07	
Descripción	Permite al usuario buscar un incidente y cancelarlo.	
Precondición	El usuario ha iniciado sesión en el sistema. El usuario tiene acceso a buscar y cancelar los incidentes del área. Deben existir Incidentes.	
Flujo Principal	1	El usuario accede por la opción “Buscar Incidentes” (CU06-Listar Incidentes) y visualiza los incidentes Abiertos de su Área.
	2	El sistema solicita que ingrese el número de incidente.
	3	El sistema encontró el Incidente (E1)
	4	El sistema despliega Acción a Realizar: [1]Editar Incidente, [2] Cancelar Incidente.
	5	El usuario selecciona opción [3]Cancelar Incidente.
	6	El sistema despliega formulario con la información del incidente.
	7	El sistema muestra opciones: [1] Confirma Cancelación, [2] Cancelar Operación.
	8	El usuario confirma Cancelación con opción [1]. (S1)
	9	El sistema solicita motivo de cancelación al usuario.
	10	El sistema emite mensaje de Incidente CANCELADO por pantalla.
	11	El sistema modifica el estado del incidente en la base de datos a CANCELADO. (S2)
	12	El sistema emite mensaje al usuario: “Incidente Cancelado”.
	13	El sistema emite mail al área de sistemas informando cancelación del incidente.
Postcondición	Se ha modificado el estado del Incidente en la base de datos. El sistema regresa al menú principal del sistema.	
Flujo Alternativo	S1	Si el usuario selecciona opción [2] Cancelar operación, el sistema despliega menú principal.
	S2	El sistema despliega menú principal.
Excepciones	E1	El sistema no encontró el Incidente y le informa al usuario para que ingrese nuevamente el número de Incidente.

Caso de Uso	CU03-Modificar Incidente
Actores	Usuario/Resolutor
Referencia	RFS-INC04- RFS-INC10
Descripción	Permite al usuario modificar un incidente cargado en su área. Permite al resolutor asumir el incidente.
Precondición	El usuario ha iniciado sesión en el sistema. El usuario tiene acceso a buscar y modificar los incidentes de su área. El resolutor tiene acceso a buscar y modificar incidentes de todas las áreas. Deben existir Incidentes.

Flujo Principal	1	El usuario accede por la opción “Buscar Incidentes” CU06- Listar Incidentes y visualiza los incidentes Abiertos de su Área. Si el usuario es del tipo Resolutor puede visualizar todos los incidentes abiertos de todas las áreas.
	2	El sistema solicita que ingrese el número de incidente.
	3	El sistema encontró el Incidente (E1)
	4	El sistema despliega Acción a Realizar: [1]Editar Incidente, [2]Cancelar Incidente.
	5	El usuario selecciona opción [1]Editar Incidente.
	6	El sistema busca la información en la base de datos y despliega formulario con la información del incidente.
	7	El sistema habilita los campos a modificar según el rol: Si es usuario puede modificar todos los datos a excepción del número de incidente, responsable de resolución y fecha de creación. Si es resolutor puede modificar solo información referente al responsable de resolución, y además información como proceso /prioridad/tipo de problema.
	8	El usuario confirma modificación. (S1)
	9	El sistema modifica la información del incidente en la base de datos
	10	El sistema emite mensaje de Incidente MODIFICADO por pantalla.
Post Condición	Se ha modificado información del Incidente en la base de datos. El sistema regresa al menú principal del sistema.	
Flujo Alternativo	S1	Si el usuario no confirma la modificación, regresa al punto 6 (visualizar la información del incidente)
Excepciones	E1	El sistema no encontró el Incidente y le informa al usuario para que ingrese nuevamente el número de Incidente.

Caso de Uso	CU04-Resolver Incidente	
Actores	Resolutor / Jefe de Sistemas-Administrador	
Referencia	RFS-INC11, RFS-SOL03, RFS-SOL04, RFS-SOL02	
Descripción	Permite dar al resolutor solución un Incidente.	
Precondición	El resolutor ha iniciado sesión en sistema. El resolutor tiene permisos para buscar y resolver Incidentes. Deben estar configurados los resolutores por aplicación y los tipos de soluciones (técnica o de configuración de usuario).	
Flujo Principal	1	El resolutor accede por la opción de búsqueda de Incidentes (CU06- Listar Incidentes) en estado abierto, donde aparece como responsable de resolución.
	2	El sistema despliega lista de incidentes.
	3	El sistema solicita se ingrese el número de Incidente para visualizar detalles.
	4	El sistema encontró el Incidente (E1)
	5	El sistema despliega el formulario con la información del incidente registrada en la base de datos.

	6	El sistema pregunta que acción desea realizar [1] Editar Incidente [2] Cancelar Incidente [3]Resolver Incidente.
	7	El usuario selecciona la opción [3]Resolver Incidente.(S1)(S2)
	8	El sistema pregunta si desea continuar [1] Confirmar, [2] Cancelar.
	9	El usuario decide Confirmar. (S3)
	10	El sistema emite mensaje: desea desplegar soluciones existentes [1] Si, [2] No
	11	El Resolutor presiona [2] No. (S4)
	12	El sistema habilita cargar detalle de solución. • Descripción corta • Descripción detallada • Tipo de Problema • Nro de Error El sistema autocompleta la información en base a los atributos del Incidente: • Aplicación • Módulo • Proceso • Pantalla
	13	El sistema autocompleta los datos: • Nro de Solución (Asigna número correlativo) • Fecha de Solución • Resolutor (Resolutor que inició sesión y está realizando el registro de la nueva solución)
	14	El sistema emite mensaje al Resolutor “Desea cargar anexo” [1] Si, [2] No.
	15	El resolutor selecciona [1]Si. (S5)
	16	El Resolutor carga Anexo (E2)
	17	El sistema muestra mensaje de Confirmación de Resolución de Incidente. [1] Confirmar Solución, [2] Cancelar.
	18	El usuario presiona [1] Confirmar Solución (S6)
	19	El sistema registra la nueva solución y el anexo en la base de datos.
	20	El sistema cambia el estado del Incidente a CERRADO en la base de datos.
	21	El sistema muestra mensaje de confirmación de Cierre del Incidente y Registro de la Solución.
	22	El sistema regresa al punto 2 (listado de incidentes).
Postcondición	Se ha registrado información de solución y cierre del Incidente en la base de datos. El sistema regresa al listado de incidentes.	
Flujo Alternativo	S1	Si el usuario selecciona [1]Editar Incidente se habilita modificación de campos responsable de resolución, aplicación, modulo, proceso, pantalla, tipo de error. CU03 (modificar Incidentes)

	S2	Si el usuario selecciona [2] Cancelar Incidente, el sistema activa caso de uso CU02 (Cancelar Incidentes)
	S3	Si el usuario selecciona [2] Cancelar, el sistema vuelve al punto 5 (despliega formulario con información del incidente).
	S4	El resolutor selecciona solución existente. El sistema despliega Caso de uso CU04 "Consultar Soluciones" punto 3. Se autocompletan los parámetros con la información del incidente. El usuario selecciona una Solución existente, esta queda vinculada al Incidente.
	S5	Si el usuario selecciona [2] No, no carga anexo. Entonces continua con punto 17 (mensaje de confirmación)
	S6	Si el usuario selecciona [2] Cancelar, el sistema retorna al punto 5 (despliega formulario con información del incidente).
Excepciones	E1	El sistema no encontró el Incidente y le informa al usuario para que ingrese nuevamente el número de Incidente.
	E2	Da error la carga del anexo, el sistema le advierte al usuario. El sistema cancela la operación y habilita cargar otro anexo.

Caso de Uso	CU05-Consultar Soluciones	
Actores	Usuario, Resolutor, Jefe de Sistemas - Administrador	
Referencia	RFS-SOL05	
Descripción	Permite listar soluciones a resolutores.	
Precondición	El resolutor inicia sesión y tiene acceso a consultar todas las soluciones. Deben existir soluciones registradas.	
Flujo Principal	1	El resolutor accede por la opción de menú "Buscar Soluciones"
	2	El sistema despliega parámetros para la búsqueda: • Aplicación (Lista) • Módulo (Lista) • Proceso(Lista) • Pantalla(Lista)
	3	El usuario completa los parámetros y confirma la búsqueda.
	4	El sistema realiza búsqueda en la base de datos, según parámetros ingresados. Si el usuario es Resolutor visualiza todas las soluciones. Si el usuario no es Resolutor, solo visualiza soluciones de tipo usuario de su área.
	5	El sistema encontró soluciones. (E1)
	6	El sistema despliega la información de las soluciones en forma de listado: • Nro de Solución • Descripción corta • Aplicación • Módulo • Proceso • Fecha Solución • Resolutor
	7	El usuario selecciona una solución.
	8	El sistema despliega formulario con toda la información de la solución:

		• Nro de Solución • Descripción corta • Descripción detallada • Aplicación • Módulo • Proceso • Pantalla • Tipo de Problema • Nro de Error • Fecha Solución • Resolutor • Incidentes relacionados
Postcondición		
Flujo Alternativo		
Excepciones	E1	No encontró soluciones. El sistema despliega mensaje: "No se encontraron soluciones con los parámetros indicados."

Caso de Uso	CU06- Listar Incidentes	
Actores	Usuario / Resolutor / Jefe de Sistemas-Administrador	
Referencia	RFS-INC08, RFS-INC09	
Descripción	Permite Listar Incidentes a usuarios, resolutores y Jefe de sistemas-Administrador	
Precondición	El usuario inicia sesión y debe tener acceso a la consulta de todos los Incidentes de su área. El resolutor incicia sesión y debe tener acceso a la consulta de todos los incidentes. Deben existir Incidentes.	
Flujo Principal	1	El usuario / resolutor accede por la opción de búsqueda de Incidentes.
	2	El sistema permite buscar por aplicación, por módulo, por proceso y por pantalla.
	3	El sistema despliega lista de incidentes dependiendo de los parámetros seleccionados, y del rol del usuario: si es Resolutor visualiza todos los Incidentes y si es Usuario solo visualiza Incidentes de su área. La información que el sistema despliega es: • Nro de Incidente • Descripción Corta • Aplicación • Módulo • Proceso • Prioridad • Usuario • Área • Responsable de Resolución • Estado • Fecha de Creación • Fecha de resolución

	4	El sistema solicita al usuario que ingrese el número de Incidente para visualizar detalles.
	5	El sistema encontró el Incidente (E1)
	6	El sistema despliega el formulario con la información del incidente registrada en la base de datos. • Número de Incidente • Fecha de creación • Usuario • Área • Aplicación • Módulo • Proceso • Pantalla • Tipo de problema • Código de error • Prioridad • Descripción corta • Descripción Detallada • Estado • Solución • Resolutor • Fecha de Cierre
Postcondición		
Flujo Alternativo		
Excepciones	E1	El sistema no encontró el Incidente y le informa al usuario para que ingrese nuevamente el número de Incidente.

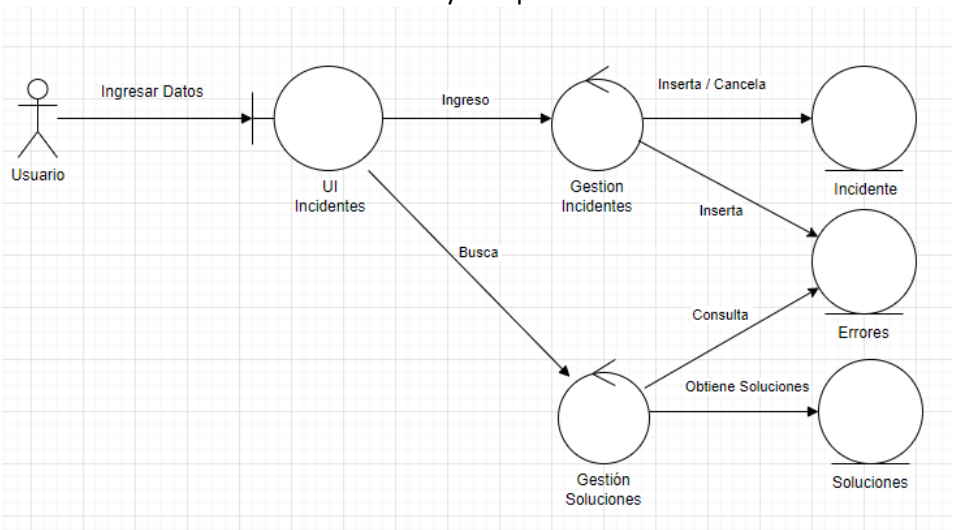
2. Diseño y Análisis Sistema Gestión de Incidentes

2.1. Etapa de análisis

2.1.1. Diagrama de clases de análisis

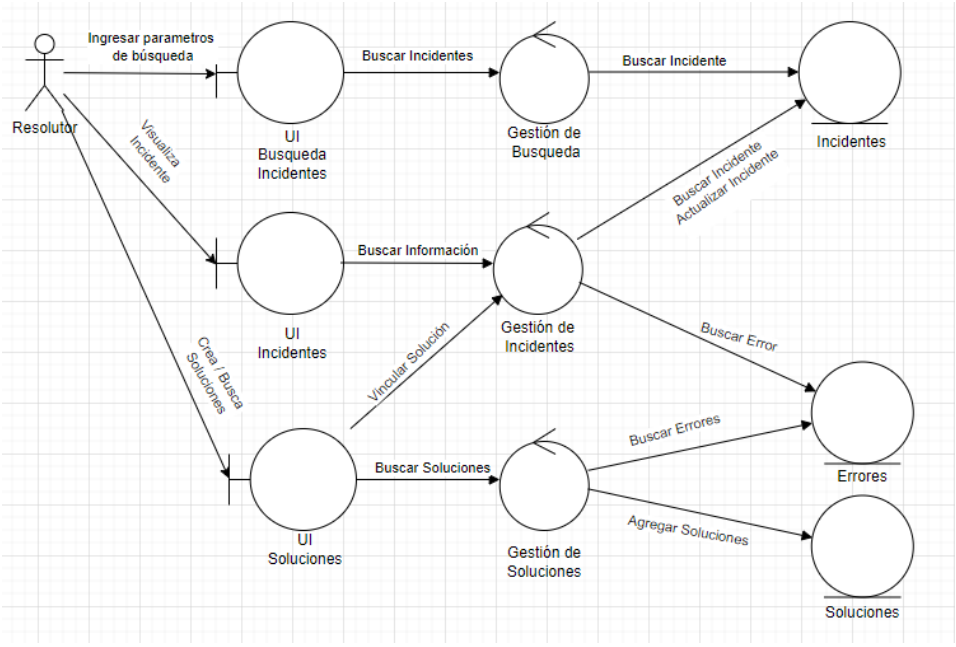
En este se pueden visualizar el actor, clases de interfaz, clases de control y las entidades. No tiene referencia de secuencia ni tiempo.

Para el CU01- Crear Nuevo Incidente y Búsqueda de Soluciones:



En el diagrama se pueden visualizar como la interfaz de Incidentes se relaciona con la con las clases de control Gestión Incidentes, para el registro en las entidades de Incidentes y Errores. Y además, con Gestión soluciones que permite a partir de la información registrada en el incidente buscar soluciones para mostrarle al usuario.

Para el CU04- Resolver Incidente:

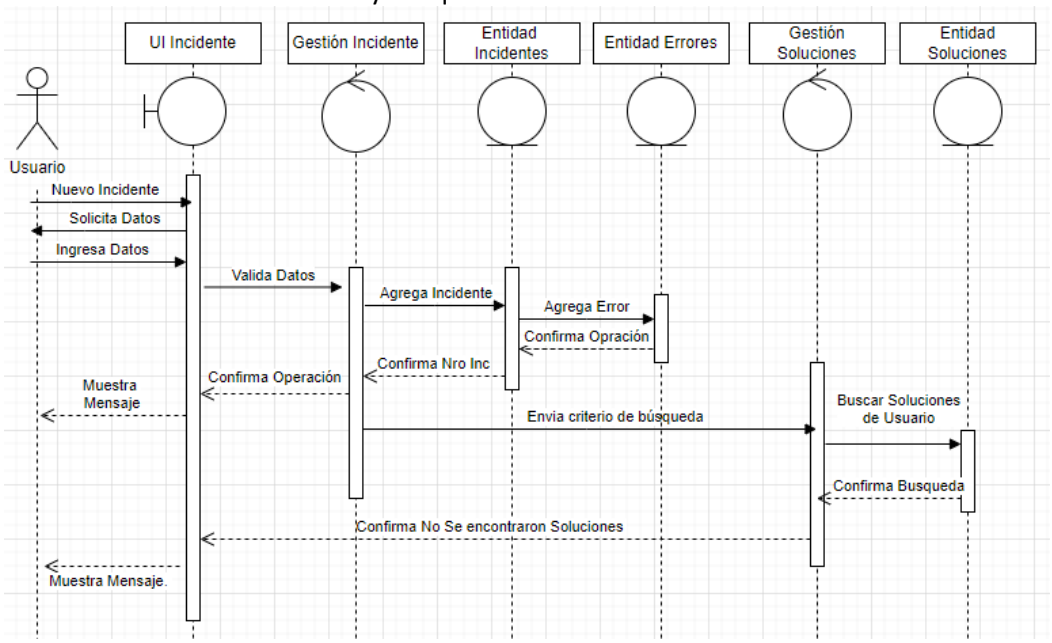


El caso de uso Resolver Incidentes hace uso de tres interfaces y tres clases de control. Gestión de Búsqueda de Incidentes para buscar en la entidad Incidentes los incidentes en estado diferente a Cerrado, Gestión de soluciones para la búsqueda de soluciones a

partir de los errores registrados o registro de una nueva solución en la entidad Soluciones, y la clase Gestión de Incidentes que permite obtener información del incidente y de los errores, como así también actualización del estado del Incidente.

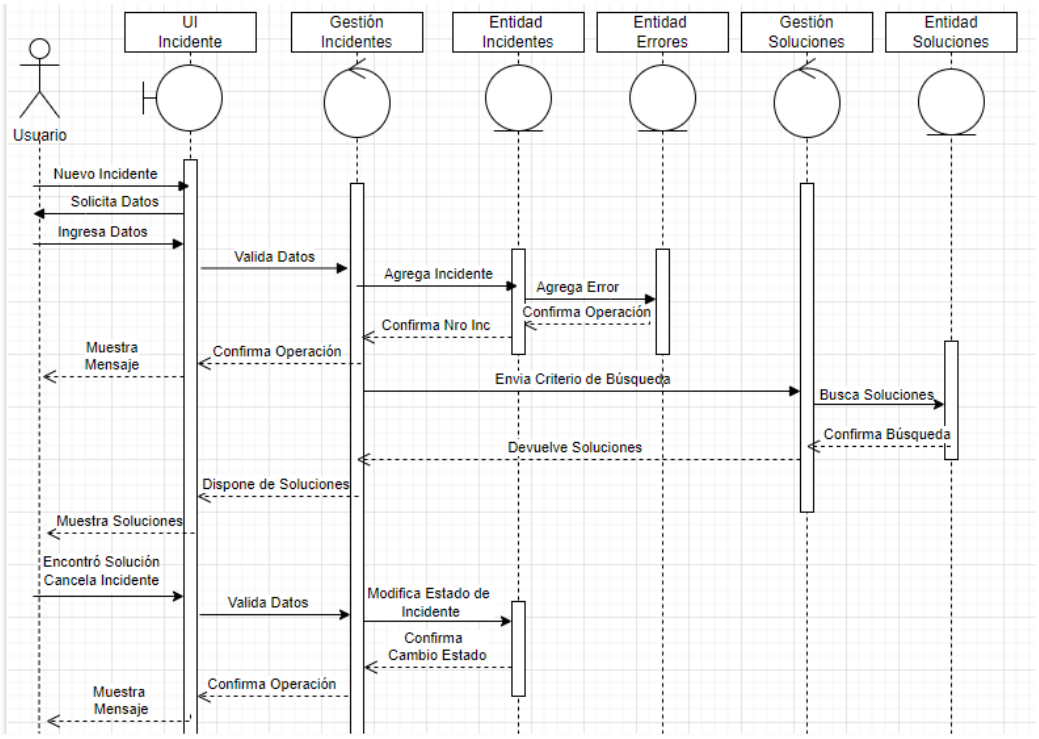
2.1.2. Diagrama de Secuencia

Es un tipo de diagrama de iteración que describe cómo y en qué orden un grupo de objetos funcionan en conjunto. A continuación, sigue el diagrama de secuencia del CU01- Crear Nuevo Incidente y Búsqueda de Soluciones en su curso normal:



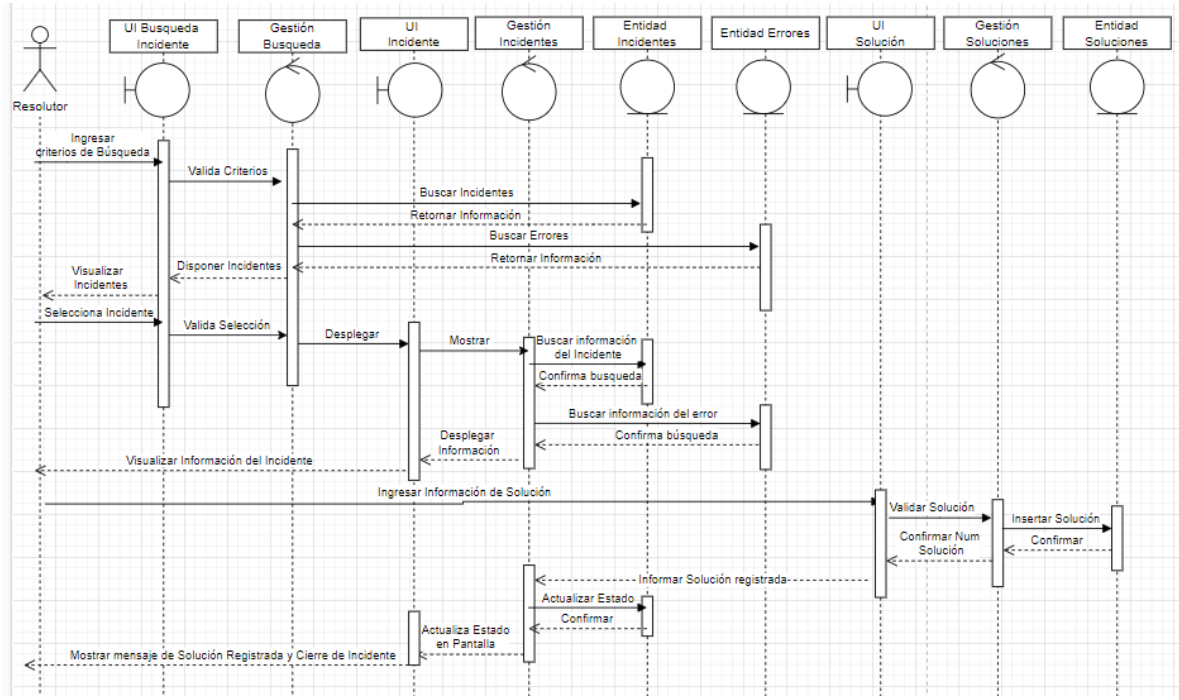
En el diagrama se visualiza la secuencia de mensajes que se realizan desde el ingreso de la información de un nuevo incidente a través de la interfaz Incidentes. En primera instancia la clase de control Gestión de Incidente valida la información ingresada por el usuario para su posterior inserción en las entidades Incidentes y Errores. Una vez que la operación es realizada la clase de control Gestión de Incidentes confirma al usuario con mensaje. En el mismo momento Gestión de Incidentes envía parámetros de búsqueda de soluciones similares a la clase Gestión de Soluciones, para que esta busque en la entidad soluciones. Para el curso normal no se encontraron soluciones coincidentes, por lo que se muestra mensaje al usuario.

CU01- Crear Nuevo Incidente y Búsqueda de Soluciones en su curso alternativo:



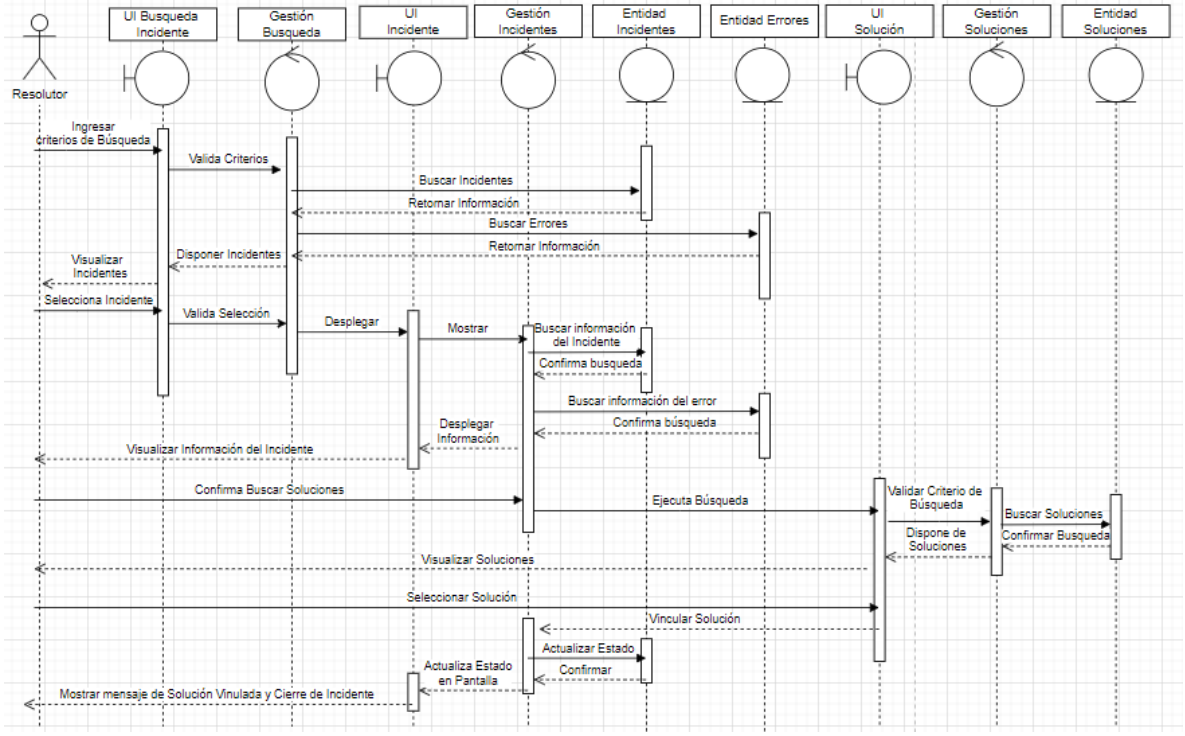
Para el curso alternativo, la diferencia radica en que la clase de control Gestión de Soluciones encontró solución de usuario coincidente con la información del Incidente ingresado, en la entidad Soluciones. Las soluciones son mostradas al usuario.
El usuario decide cancelar el incidente, a través de la clase Gestión de Incidentes se modifica el estado del Incidente a Cancelado, en la entidad Incidentes.

A continuación, sigue el diagrama de secuencia del CU04- Resolver Incidentes en su curso normal:



El diagrama muestra como a partir de la búsqueda del incidente el resolutor realiza el ingreso de una nueva solución.
A partir de la interfaz Búsqueda de Incidentes se hace uso de la clase Gestión de búsqueda de Incidentes, y se realiza la búsqueda en las entidades Incidentes y errores, retornando información de los Incidentes encontrados.
Luego el resolutor selecciona el incidente y a través de la interfaz Incidentes y la clase de control Gestión de Incidentes se obtiene la información del incidente para que el resolutor pueda visualizarla.
Finalmente, a través de la interfaz Soluciones y la clase de control Gestión de soluciones el resolutor puede crear una nueva solución en la entidad soluciones y retornar a Gestión de Incidentes para actualizar estado del Incidente a Cerrado. Se muestra mensaje al usuario confirmando operación.

CU04- Resolver Incidente en su curso alternativo:

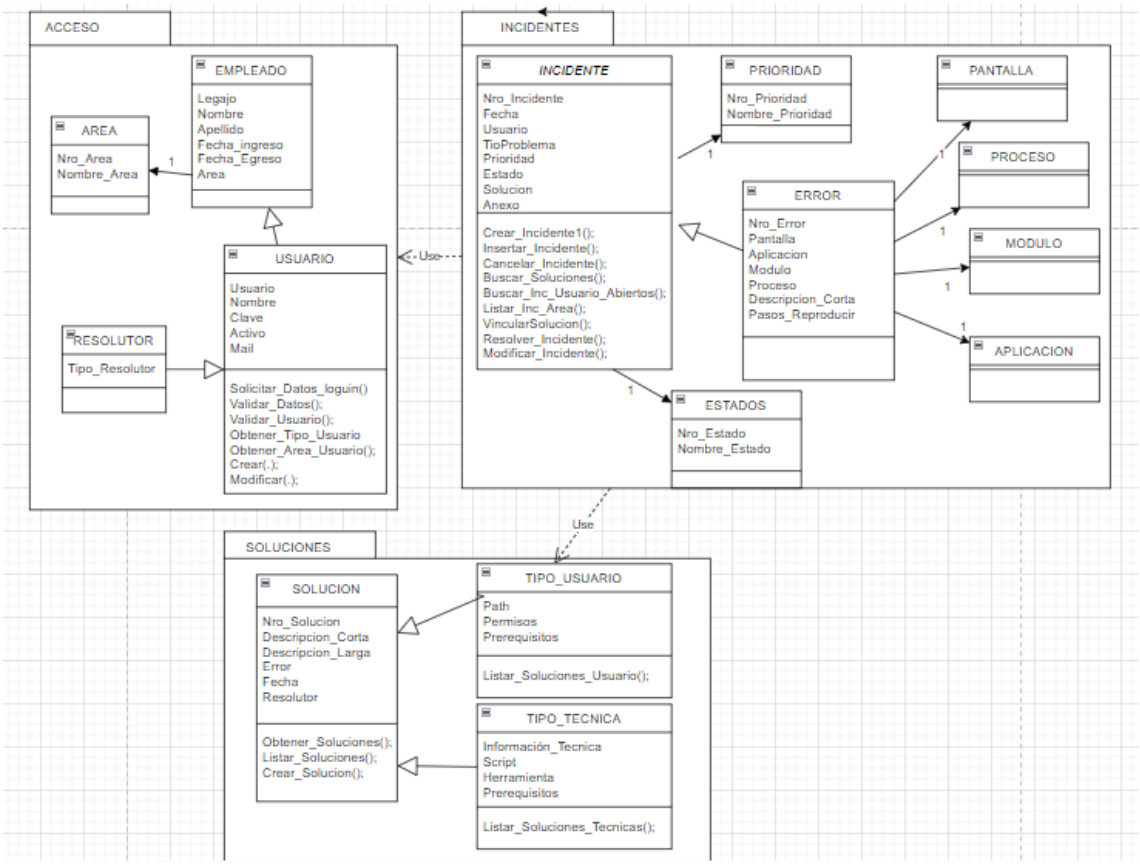


Para el curso alternativo, la diferencia está en que el Resolutor decide buscar soluciones y al encontrar una solución a través de la clase Gestión de Soluciones la misma es mostrada en la interfaz Soluciones y vinculada al Incidente a través de la clase Gestión de Incidentes, que además actualiza el estado en la entidad Incidentes.

2.1.3. Paquetes del Análisis

Permite optimizar y facilitar el manejo de los diferentes modelos de un sistema. Para el caso en análisis se consideran 3 paquetes:

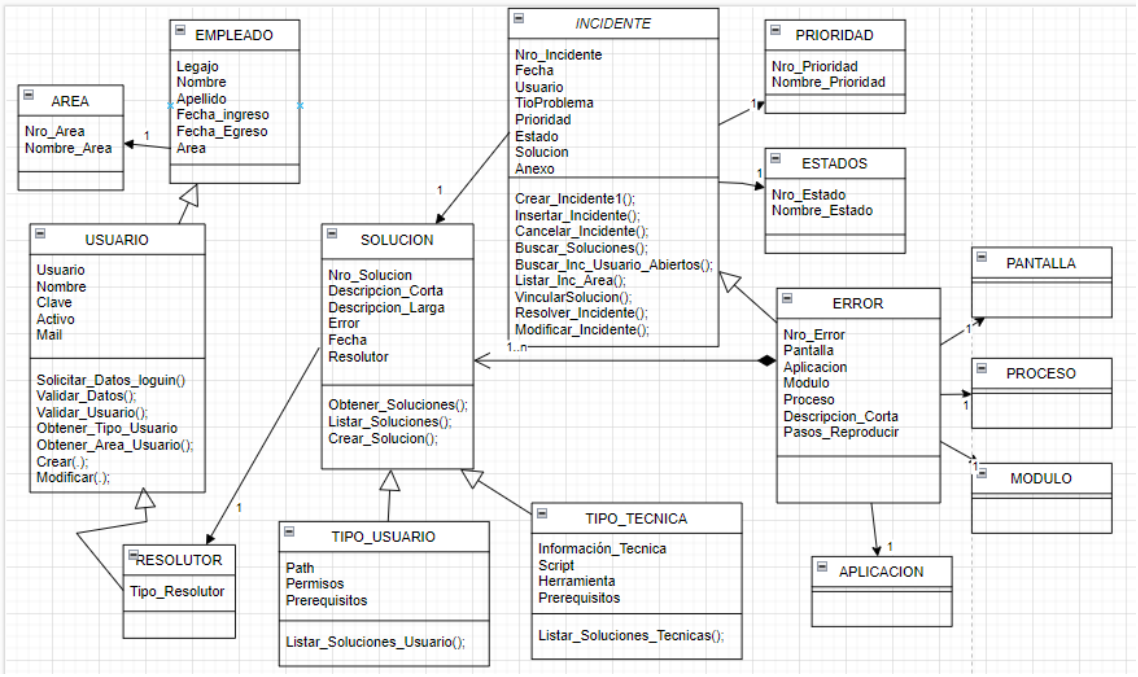
- Acceso: Agrupa las clases referentes a los empleados y usuarios del sistema.
- Incidentes: Contiene clases que permiten la creación de Incidentes.
- Soluciones: Contiene las clases referentes a las Soluciones.



2.2. Etapa de diseño

2.2.1. Diagrama de Clases

Permiten trazar claramente la estructura de un sistema modelando clases, atributos, operaciones y relaciones entre objetos. Para el sistema Gestión de Incidentes se puede considerar el siguiente diagrama de clases:



2.3. Etapa de implementación

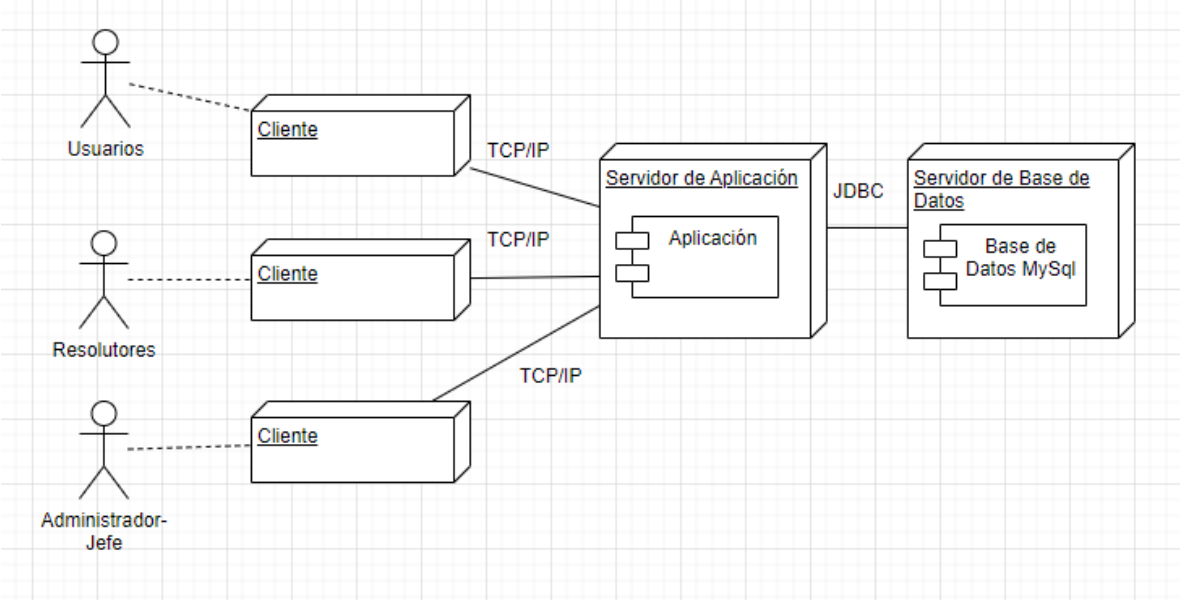
2.3.1. Diagrama de Despliegue

A través de un diagrama de despliegue se intenta representar la distribución de los componentes en el sistema en nodos físicos (servidores, dispositivos y computadoras) y cómo interactúan entre sí.

Es necesario considerar los requerimientos relevados hasta el momento.

Requerimiento del sistema	Descripción
RNFS-APP01	El sistema debe estar desarrollado en Java.
RNFS-APP02	El sistema debe tener una disponibilidad de 5x24 con momentos de inactividad los fin de semana ya que la planta no produce.
RNFS-APP03	El sistema debe ser escalable para nuevos usuarios, aplicaciones y tipos de problemas.
RNFS-BD01	El sistema debe contar con una base de datos MySQL.
RNFS-BD02	Para la persistencia y consulta de datos en la BD se debe usar un patrón MVC (modelo-vista-controlador).
RNFS-SERV06	El sistema debe estar instalado en servidor de la empresa “OLEOSIN”.

A partir de los Requerimientos No Funcionales, se presenta el Diagrama de despliegue correspondiente:



2.4. Etapa de pruebas

2.4.1. Procedimiento de pruebas

Se considera necesario seguir el siguiente procedimiento para asegurar la máxima organización en planificación, diseño, ejecución y evaluación de pruebas de sistemas para que el software sea aprobado en el final del ciclo de desarrollo.

Las fases en las que se realizarán las pruebas son:

1. Planificación de pruebas: Identificar los requerimientos, desarrollar la estrategia de pruebas, identificar los recursos necesarios para realizar las pruebas, generar plan de pruebas.
2. Diseño de pruebas: Desarrollo de pruebas, definir y describir los casos de pruebas.
3. Implementación de pruebas: Definir y disponer de entorno de pruebas. Definir cantidad de aprobadores por tipo de prueba.

4. Ejecución de las pruebas: Ejecutar los casos de prueba. Evaluar proceso de prueba, verificar los resultados y completar tabla de evaluación de pruebas funcionales y pruebas no funcionales. Completar tabla tratamiento de defectos.
5. Evaluación de las pruebas: Evaluar la cobertura de los casos de prueba, analizar los defectos, determinar si se alcanzaron los criterios de las pruebas. Crear los informes de evaluación.
   A considerar criterios de aprobación y rechazo.
   - Graves: Información crítica presentada erróneamente, información mal registrada en la base de datos, incumplimiento de requerimientos funcionales principales.
   - Medios: Errores de presentación de datos, incumplimiento de objetivos en funciones secundarias.
   - Leves: Errores en presentación de datos secundarios, comportamiento incorrecto, dificultad para operar dentro del sistema, etc.
   - Aprobación: Se aprobará con un 100% de pruebas ejecutadas, y con un 90% de aceptación. En el 10% restante, pueden existir errores medios o bajos, pero no Graves.
   - Rechazo: El sistema no cumple con el nivel exigido.

2.4.2. Plan de Pruebas

El plan de pruebas, describe las estrategias de prueba para cada caso de uso desarrollado, el orden de prueba y los criterios de aceptación. A continuación, se presenta el plan de pruebas básico propuesto para los casos de uso: CU01 (Crear Incidente) y CU04 (Resolver Incidente).

Caso de Uso	Código de Prueba	Tipo de Prueba	Técnica	Observaciones
CU01	CPC01	Componente	Cobertura	Aplicación de caja blanca al método CrearIncidente(…) de la clase Incidentes
CU01	CPC02	Componente	Cobertura	Aplicación de caja blanca al método BuscarSoluciones(..) de usuario de la clase Soluciones.
CU01	CPS03	Sistema	Pruebas Funcionales	Verificación de requerimientos funcionales para el caso de uso “Crear Incidente”.
CU04	CPC04	Componente	Cobertura	Aplicación de caja blanca al método CerrarIncidente(..) de la clase Incidentes
CU04	CPC05	Componente	Cobertura	Aplicación de caja blanca al método BuscarSoluciones(..) de la clase Soluciones
CU04	CPS06	Sistema	Pruebas Funcionales	Aplicación de caja blanca al método VincularSoluciones(..) de la clase Incidentes.
CU04	CPS07	Sistema	Pruebas Funcionales	Verificación de requerimientos funcionales para el caso de uso “Cerrar Incidente”.
…..	……	……	……	…..

2.4.3. Caso de pruebas

Los casos de pruebas incluyen una descripción detallada de la funcionalidad, acciones, condiciones y entradas necesarias para evaluar al usar el sistema. Puede basarse de los requisitos o en los casos de uso.
Se presenta la siguiente tabla para detallar el CPS03 (caso de prueba de sistema 03), que hace referencia al CU01-Crear Incidente y CPS07 (caso de prueba de sistema 07) que hace referencia al CU04-Resolver Incidente.

Caso de Prueba General - **CPS03**			
Caso de uso: **CU01- Crear Incidente**			
Caso de prueba	**Secuencia**	**Resultado Esperado**	**Objetivo**
CPS03.1	1. El usuario ingresa por la opción de menú: Crear Incidentes. 2. Cargar descripción: "*Problema al registrar factura. Se generó error 1002 por inconsistencia tipo de dato al registrar registro.*" Y descripción para reproducir: "*Luego de seleccionar el proveedor OLEOS DEL SUR, al dar tab, aparece error 1002.*". 3.Seleccionar Aplicación: "Sys_Admin", Modulo: "Administración" , Proceso: "*Ingreso Facturas*" , Pantalla "*Factura*". Prioridad "2" Media 5. Confirmar Operación. 6. Registrar Información en la base de datos. 7. Emitir mail al área de sistemas. 8. El sistema busca Soluciones de usuario y muestra por pantalla que no encontró soluciones.	El incidente es guardado en la base de datos, se asigna número. El sistema emite mensaje de confirmación Exitosa. El sistema no encuentra soluciones coincidentes para la aplicación, módulo, proceso, pantalla ingresada.	Registro exitoso de Ingreso de Incidente, sin soluciones coincidentes.
CPS03.2	1. El usuario ingresa por la opción de menú: Crear Incidentes. 2. Cargar descripción: "*Error al registrar nuevo Transporte.*" Y descripción para reproducir: "Luego de colocar información de razón social, chasis y choferes aparece error 1000-No Data Found. Ddddddddddddddddddddd Ddddddddddddddddddddddddd Ddddddddddddddddddddddddd ddddddddddddddddddddddddd ddddddddddddddddddddddddd ddddddddddddddddddddd ". 3. Seleccionar Aplicación: "Sales System", Modulo: "Pedidos" , Proceso: "Despachar" , Pantalla "Transporte". Prioridad = [5] 4. Confirmar operación	Advierte "Información extensa para el campo Pasos reproducción". Advierte campo "prioridad" con información incorrecta.	Advertir problemas con la información que se ingresó y/o faltante de información.

CPS03.3	1.El usuario ingresa por la opción de menú: Crear Incidentes. 2. Cargar descripción: "*Error al registrar pago.*" Y descripción para reproducir: "Luego de colocar información del banco aparece error 1000-No Data Found. ". 3. Selecciona Tipo de problema: "Sistemas". 4.Seleccionar Aplicación: "Sys_Admin", Modulo: "Administración" , Proceso: "Ingreso Pagos" , Pantalla "Pagos". Prioridad: [2]Media 5. Confirmar Operación. 6. Registrar Información en la base de datos. 7. El sistema busca Soluciones de usuario y muestra por pantalla.	Se encontraron soluciones en la base de datos, se despliega listado de soluciones de usuario coincidentes con los parámetros usados. Ejemplo: Solución nro 1, Descripción: "Se agregó código de banco. " Descripción larga: " Se accede por la configuración de bancos y se agrega código de banco válido. "	Registrar Incidente y mostrar soluciones.
CPS03.4	1.El usuario ingresa por la opción de menú: Crear Incidentes. 2. Cargar descripción: "*Error al registrar nuevo Transporte.*" Y descripción para reproducir: "Luego de colocar información de razón social, chasis y choferes aparece error 1000-No Data Found. ". 3.Selecciona Tipo de problema: "Sistemas". 4.Seleccionar Aplicación: "Sales System", Modulo: ""	El sistema no permite registrar el incidente, por que no se indicó el Modulo.	Controlar Selección de Módulo.
CPS03.5	1.El usuario ingresa por la opción de menú: Crear Incidentes. 2. Cargar descripción: "*Error al registrar nuevo Transporte.*" Y descripción para reproducir: "Luego de colocar información de razón social, chasis y choferes aparece error 1000-No Data Found. ". 3.Selecciona Tipo de problema: "Sistemas". 4.Seleccionar Aplicación: "Sales System", Modulo: "Pedidos" Proceso: ""	El sistema no permite registrar el incidente, por que no se indicó el Proceso.	Controlar Selección de Proceso.

CPS03.6	1.El usuario ingresa por la opción de menú: Crear Incidentes. 2. Cargar descripción: "*Error al registrar nuevo Transporte.*" Y descripción para reproducir: "Luego de colocar información de razón social, chasis y choferes aparece error 1000-No Data Found. ". 3.Selecciona Tipo de problema: "Sistemas". 4.Seleccionar Aplicación: "Sales System", Modulo: "Pedidos" Proceso: "Despachar" Pantalla: ""	El sistema no permite registrar el incidente, por que no se indicó Pantalla.	Controlar Selección de datos de ingreso.
CPS03.7	1. El usuario ingresa por la opción de menú: Crear Incidentes. 2. Cargar descripción: "*Error al registrar nuevo Transporte.*" Y descripción para reproducir: "Luego de colocar información de razón social, chasis y choferes aparece error 1000-No Data Found. ". 3.Seleccionar Aplicación: "Sales System", Modulo: "Pedidos" , Proceso: "Despachar" , Pantalla "Transporte". 4. Selecciona Tipo de problema: "Sistemas". 5. Cancela Operación.	No se registra Incidente en la base de datos.	Cancelar carga de incidente.

Caso de Prueba General - **CPS07**			
Caso de uso: **CU04- Resolver Incidente**			
Caso de prueba	**Secuencia**	**Resultado Esperado**	**Objetivo**
CPS07.1	1. El resolutor ingresa por la opción de menú: Buscar Incidentes. 2. Se despliegan los incidentes abiertos. 3. Se selecciona el Incidente con siguientes características información de Aplicación: "Sys_Admin", Modulo: "Administración", Proceso: "Ingreso Facturas", Pantalla: "Factura" 4. Se despliega formulario con la información registrada en la base de datos del Incidente.	Encontrar Incidente, Crear solución Vincular Solución con Incidente. Modificar estado de Incidente a Resuelto. Completar resolutor con usuario que generó la solución.	Crear nueva solución y resolver Incidente.

	5. Se selecciona opción Resolver Incidente. 6. El sistema consulta si desea desplegar soluciones existentes. 7. El resolutor decide registrar nueva solución. 8. El sistema autocompleta información de Aplicación: "Sys_Admin", Modulo: " Administración", Proceso: " Ingreso Facturas", Pantalla: "Factura" según Incidente por Cerrar. 9. El sistema habilita campos para registrar la solución-- Descripción Corta: "Verificación de tipo de dato CUIT en el cliente", Descripción detallada= "Revisión de información de CUIT en ABM de clientes.", Nro de Error: "1002", Tipo de Problema: Sistemas, Solución de Usuario: [1] Verdadero. 10. Se anexa Archivo con más información sobre la solución. 11. Se confirma Operación. 12. Se muestra mensaje de confirmación. 13. Se registra solución en la base de datos. 14. Se modifica estado de Incidente a Resuelto. 15. Se vincula solución a Incidente. 16. Se informa al usuario por mail que el Incidente fue resuelto.		
CPS07.2	1. El resolutor ingresa por la opción de menú: Buscar Incidentes. 2. Se despliegan los incidentes abiertos. 3. Se selecciona el Incidente Nro 4. 4. Se despliega formulario con la información registrada en la base de datos del Incidente. 5. Se selecciona opción Resolver Incidente. 6. El sistema consulta si desea desplegar soluciones existentes. 7. El resolutor decide Buscar soluciones existentes. 8. El sistema despliega soluciones que coincidan con las características del Incidente (Aplicación: "Sys_Admin", Modulo: "Pagos", Proceso: "Ingreso	Encontrar Incidente, Encontrar solución en la base de datos. Vincular Solución con Incidente. Modificar estado de Incidente a Resuelto.	Vincular solución y resolver Incidente.

	de Pagos", Pantalla: "Pago", Código de error: 1010) 9. El resolutor selecciona solución número 1. 10. Se confirma Operación. 11. Se muestra mensaje de confirmación. 12. Se modifica estado de Incidente a Resuelto. 16. Se vincula solución a Incidente. 17. Se informa al usuario por mail que el Incidente fue resuelto.		
CPS07.3	1. El resolutor ingresa por la opción de menú: Buscar Incidentes. 2. Se despliegan los incidentes abiertos. 3. Se selecciona Incidente con las siguientes características Aplicación: "Sys_Admin", Modulo: "Administración", Proceso: "Ingreso Facturas", Pantalla: "Factura" 4. Se despliega formulario con la información registrada en la base de datos del Incidente. 5. Se selecciona opción Resolver Incidente. 6. El sistema consulta si desea desplegar soluciones existentes. 7. El resolutor decide registrar nueva solución. 8. El sistema autocompleta información de Aplicación, Modulo, Proceso, Pantalla que corresponden al Incidente por Cerrar. 9. El sistema habilita campos para registrar la solución-- Descripción Corta: "Verificación de tipo de dato CUIT en el cliente en ABM Clientes desde Menú Clientes, en el módulo de Administración. ", Descripción detallada, tipo de problema = "Revisión de información de CUIT en ABM de clientes. sssssssssssssssss sssssssssssssssssssssssssssssssssssss sssssssssssssssssssssssssssssssssssss sssssssssssssssssssssssssssssssssssss sssssssssssssssssssssssssssssssssssss sssssssssssssssssssssssssssssss ",	Advertir al resolutor ante errores u omisión en la carga de datos obligatorios.	Advertir errores en la cargar de soluciones nuevas.

	10. El resolutor omite cargar datos obligatorios como Nro de Error y Tipo de Problema. 10. Se anexa Archivo con más información sobre la solución. 11. Se confirma Operación. 12. Se muestra mensaje de error en el registro de la solución. 13. El sistema regresa al formulario de carga de solución.		
CPS07.4	1. El resolutor ingresa por la opción de menú: Buscar Incidentes. 2. Se despliegan los incidentes abiertos. 3. Se selecciona el Incidente Nro 4. 4. Se despliega formulario con la información registrada en la base de datos del Incidente. 5. Se selecciona opción Resolver Incidente. 6. El sistema consulta si desea desplegar soluciones existentes. 7. El resolutor decide Buscar soluciones existentes. 8. El sistema despliega soluciones que coincidan con las características del Incidente (Aplicación: “Sys_Admin”, Modulo: “Pagos”, Proceso: “Ingreso de Pagos”, Pantalla: “Pago”, Código de error: 1010) 9. El resolutor selecciona solución número 1. 10. El usuario Cancela la operación.	El incidente no cambia su estado. No se vincula solución.	Cancelar la resolución de un incidente.
CPS07.5	1. El resolutor ingresa por la opción de menú: Buscar Incidentes. 2. Se despliegan los incidentes abiertos. 3. Se selecciona el Incidente con siguientes características información de Aplicación: “Sys_Admin”, Modulo: “Administración”, Proceso: “Ingreso Facturas”, Pantalla: ”Factura” 4. Se despliega formulario con la información registrada en la base de datos del Incidente. 5. Se selecciona opción Resolver Incidente. 6. El sistema consulta si desea desplegar soluciones existentes. 7. El resolutor decide registrar nueva solución.	La solución no queda registrada. El incidente no cambia su estado.	Cancelar carga de solución y resolución de incidente.

	8. El sistema autocompleta información de Aplicación: “Sys_Admin”, Modulo: “ Administración”, Proceso: “ Ingreso Facturas”, Pantalla: ”Factura” según Incidente por Cerrar. 9. El sistema habilita campos para registrar la solución-- Descripción Corta: “Verificación de tipo de dato CUIT en el cliente”, Descripción detallada, tipo de problema = “Revisión de información de CUIT en ABM de clientes.”, Nro de Error: “1002” 10. Se anexa Archivo con más información sobre la solución. 11. El usuario cancela la operación.		
CPS07.6	1. El resolutor ingresa por la opción de menú: Buscar Incidentes. 2. Se despliegan los incidentes abiertos. 3. Se selecciona el Incidente con siguientes características información de Aplicación: “Sys_Admin”, Modulo: “Administración”, Proceso: “Ingreso Facturas”, Pantalla: ”Factura” 4. Se despliega formulario con la información registrada en la base de datos del Incidente. 5. Se selecciona opción Resolver Incidente. 6. El sistema consulta si desea desplegar soluciones existentes. 7. El resolutor decide registrar nueva solución. 8. El sistema autocompleta información de Aplicación: “Sys_Admin”, Modulo: “ Administración”, Proceso: “ Ingreso Facturas”, Pantalla: ”Factura” según Incidente por Cerrar. 9. El sistema habilita campos para registrar la solución-- Descripción Corta: “Verificación de tipo de dato CUIT en el cliente”, Descripción detallada= “Revisión de información de CUIT en ABM de clientes.”, Nro de Error: “1002”, Tipo de Problema: Sistemas, Solución de Usuario: ‘XXX’.	No permitir continuar con la carga de la solución.	Advertir sobre valor ingresado en campo Tipo Solución de Usuario inválido. Solo se aceptan [1] Verdadero, [2] Falso

	El sistema debe advertir error de ingreso de Tipo de Solución.		

2.4.4. Tablero de evaluación

Por cada caso de prueba se realiza una valoración de funcionalidad, usabilidad y apariencia en la solución otorgada. El porcentaje de aprobación determinará si el caso de prueba cumplió o no con la necesidad del cliente.

Caso de prueba	Descripción	Usuario	Fecha	Observaciones	% Acept	Resultado
CPS03.1	Usabilidad Pantalla	VBIJARRA	12/05/24	Lentitud al pasar de campo a campo.	90%	Aceptado
CPS03.1	Funcionalidad	VBIJARRA	12/05/24	Sin Observaciones	90%	Aceptado
CPS03.1	Apariencia	VBIJARRA	12/05/24	Sin Observaciones	90%	Aceptado
CPS03.2	Usabilidad Pantalla	VBIJARRA	12/05/24			
CPS03.2	Funcionalidad	VBIJARRA	12/05/24			
CPS03.2	Apariencia	VBIJARRA	12/05/24			
CPS03.3	Usabilidad Pantalla	VBIJARRA	12/05/24			
CPS03.3	Funcionalidad	VBIJARRA	12/05/24			
CPS03.3	Apariencia	VBIJARRA	12/05/24			
…	…	…	…	…	…	..

2.4.5. Tratamiento de defectos

Se considera la siguiente tabla, con ejemplos de tratamiento de defectos, para la etapa de pruebas del proyecto.

N° de Defecto	Caso de Prueba	Tipo de Prueba	Descripción	Importancia	Estado	Responsable	Fecha Solución
DF.INC1	CPS03.1	P. Func.	Luego de completar todos los datos para el registro de incidentes, al dar confirmar para guardar, no se dispara envío de mail a sistemas.	Media	Pendiente		
DF.INC2	CPS03.3	P. Func.	Luego de registrar el incidente nro 100, aparece error al obtener información de la solución.	Alta	Pendiente		
DF.SOL3							
DF.SOL4…							

Referencias: DF:Defecto – INC:Modulo Incidente – SOL:Modulo Soluciones

2.5. Interfaz

Se describe prototipo de interfaz a utilizar en el sistema Gestión de Incidentes.

En primera instancia el sistema ofrece un menú con las diferentes operaciones que se pueden realizar:

```
_____________________________________________________
GESTI@N DE INCIDENTES - OLEOSIN
_____________________________________________________
MENU PRINCIPAL
_____________________________________________________
[1] Crear Incidente
[2] Modificar Incidente
[3] Cancelar Incidente
[4] Resolver Incidente
[5] Buscar Incidentes
[6] Buscar Soluciones
[0] Salir
_____________________________________________________
Seleccione una opci@n.
```

El usuario debe seleccionar una opción según la tarea que see realizar:

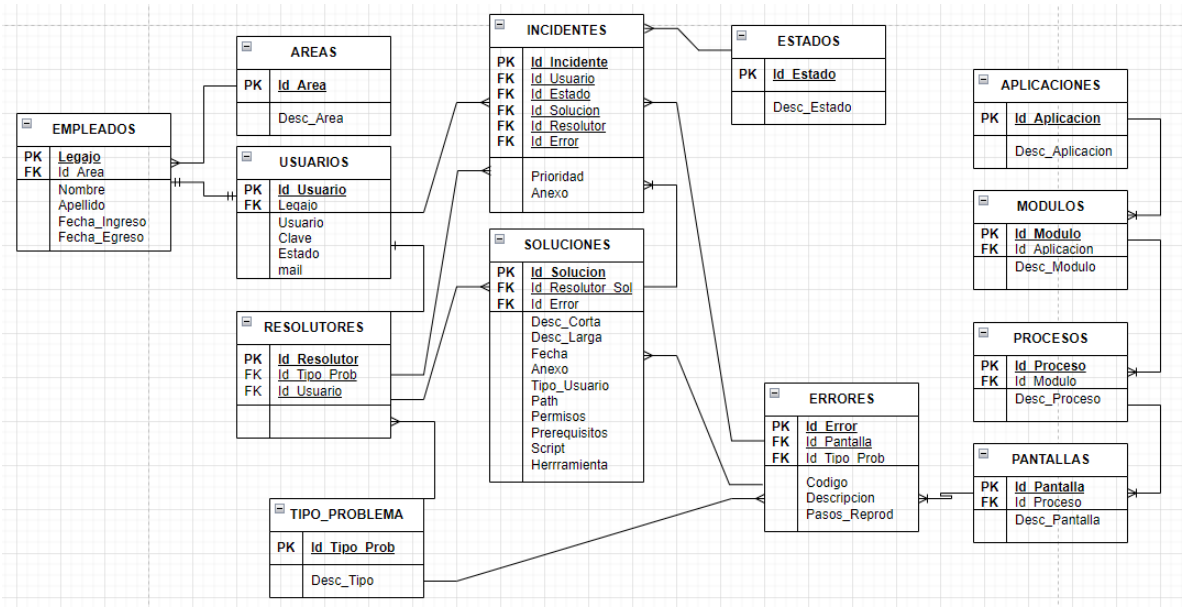
Se simula selección de la primera opción de menú [1]- Crear Incidente (CU01-Crear Incidente). El usuario completa la información que el sistema solicita. En caso de encontrar error, el sistema muestra mensaje de error.

```
_____________________________
INGRESAR INCIDENTE
_____________________________
Ingrese Descripci@n del problema:
No se puede registrar producción L1 18/05/2024.
Ingrese pasos para reproducir el error:
Al ingresar al sistema de producción L1, pedido 1245 el sistema emite error por falta de información de material.
Ingrese C@digo de error:
1001
Seleccione Tipo Problema:
[1] Sistema [2] IT
1
Seleccione aplicaci@n:
[1] SysAdmin [2] Producci@n [3] Sales System [4] HR System [5] Hardware [6] Telefonia
2
Seleccione Modulo:
[3] Producci@n [4] Empaquetado
3
Seleccione Proceso:
[5] Ingreso Producci@n
5
Seleccione Pantalla:
[9] Producci@n L1 [10] Producci@n L2 [11] Producci@n L3
9
Desea anexar DOcumento?
[1] Si [2] No
2
Desea confirmar Incidente?
[1] Si [2] No
1
No se encontraron Soluciones coincidentes.
El incidente nro 1 fu@ regstrado con @xito.
```

2.6. Definición de base de datos para el sistema

La base de datos donde se van a crear las tablas del sistema se denomina “GESTION_INC”, se considera el siguiente prototipo de tablas de base de datos. Estas tablas contendrán información de los atributos principales de los incidentes, errores y soluciones disponibles.

2.6.1. Diagrama entidad-relación de la base de datos.



2.6.2. Creación de las tablas

A continuación, se muestra la como se crean las principales entidades de la aplicación Gestión de Incidentes en la base de datos MySql. Se aclara que previamente se realizó creación y carga de tablas paramétricas (a detallar al final de este apartado):

Tabla <<**Errores**>> contiene Identificador de Error, pantalla, descripción, campo para información de pasos para reproducir el error y tipo de problema.

```
CREATE TABLE inc.errores  ( Id_Error INT NOT NULL,
Id_Pantalla INT NOT NULL,
Codigo VARCHAR(10) NOT NULL,
Descripcion VARCHAR(250) NOT NULL,
Pasos_Reprod VARCHAR(250) NOT NULL,
Id_Tipo_Prob INT NOT NULL,
PRIMARY KEY (Id_Error),
INDEX Id_Pantalla_FK_idx (Id_Pantalla ASC),
INDEX Id_TipoProb_idx (Id_Tipo_Prob ASC),
CONSTRAINT Id_Pantalla_FK    FOREIGN KEY (Id_Pantalla)    REFERENCES inc.pantallas (Id_Pantalla),
CONSTRAINT Id_Tipo_Prob_FK    FOREIGN KEY (Id_Tipo_Prob)    REFERENCES inc.tipo_problema (Id_Tipo_Prob));
```

Tabla <<**Soluciones**>> Identificador de Solución, descripción, resolutor, fecha, anexo, Identificador de error, Tipo_Usuario que determina si la solución puede ser visualizada por los usuarios junto con Desc_Patch y Prerequisitos. Mientras que los campos Permisos, Script y Herramienta se completarán para las soluciones de Tipo Técnicas (es decir las que tengan Tipo_Usuario = N).
La tabla se vincula con las tablas Resolutores y Errores.

```
CREATE TABLE inc.soluciones ( Id_Solucion INT NOT NULL,
Desc_Corta  VARCHAR(30) NOT NULL,
Desc_Larga VARCHAR(250) NOT NULL,
Id_Resolutor INT NOT NULL,
Fecha Date NOT NULL,
Anexo  LONGBLOB,
Id_Error INT NOT NULL,
Tipo_Usuario boolean NOT NULL,
Desc_Path varchar(100),
Permisos varchar(250),
Prerequisitos varchar(250),
Script varchar(250),
Herramienta varchar(100),
PRIMARY KEY (Id_Solucion),
INDEX Id_Resolutor_FK_idx (Id_Resolutor ASC), INDEX Id_Error_FK_idx (Id_Error ASC),
CONSTRAINT Id_Resolutor_FK    FOREIGN KEY (Id_Resolutor)    REFERENCES inc.resolutores (Id_Resolutor),
CONSTRAINT Id_Error_FK    FOREIGN KEY (Id_Error)    REFERENCES inc.errores (Id_Error));
```

Tabla <<**Incidentes**>> que contiene Identificador de Incidentes, fecha, usuario que genera el incidente, resolutor, estado, prioridad, descripción, Identificador de error, solución.

```
CREATE TABLE inc.incidentes ( Id_Incidente INT NOT NULL,
Id_Usuario INT NOT NULL,
Id_Resolutor INT,
Fecha Date NOT NULL,
Anexo  LONGBLOB,
Id_Error INT NOT NULL,
Prioridad INT NOT NULL,
Id_Estado INT NOT NULL,
Id_Solucion INT,
PRIMARY KEY (Id_Incidente),
INDEX Id_Res_FK (Id_Resolutor ASC),
INDEX Id_Err_FK_idx (Id_Error ASC),
INDEX Id_Sol_FK_idx (Id_Solucion ASC),
INDEX Id_Est_FK_idx (Id_Estado ASC),
INDEX Id_Usu_FK_idx (Id_Usuario ASC),
CONSTRAINT Id_Res_FK    FOREIGN KEY (Id_Resolutor)    REFERENCES inc.resolutores (Id_Resolutor)   ,
CONSTRAINT Id_Err_FK    FOREIGN KEY (Id_Error)    REFERENCES inc.errores (Id_Error)  ,
CONSTRAINT Id_Sol_FK    FOREIGN KEY (Id_Solucion)    REFERENCES inc.soluciones (Id_Solucion) ,
CONSTRAINT Id_Est_FK    FOREIGN KEY (Id_Estado)    REFERENCES inc.estados (Id_Estado)    ,
CONSTRAINT Id_Usu_FK    FOREIGN KEY (Id_Usuario)    REFERENCES inc.usuarios (Id_Usuario));
```

Tablas parametricas:

<<**Aplicaciones**>> Contiene las Aplicaciones que utilizan los usuarios en la empresa, por ejemplo: 1- Sys_Admin que es el sistema de Administración.

<<**Modulos**>> Contiene los Módulos asociados a las aplicaciones usadas en la empresa, por ejemplo para la aplicación [1]-Sys_Admin los módulos [1]-Administración y [2]-Pagos.

<<**Procesos**>> Vinculados a la tabla de módulos, por ejemplo: para el módulo [1]-Administración, los procesos son: [1]-Ingreso Factura y [2]-Proveedores.

<<**Pantallas**>> Contiene información referente a las pantallas de cada Proceso. Por ejemplo para el proceso [1]- Ingreso Factura las pantallas [1]-Factura y [2]- Líneas de Facturas.

<<**Estados**>> que pueden tomar los incidentes: [1]-Abierto, [2]-Cancelado, [3]-Asignado, [4]-Cerrado.

<<**TipoProblema**>> Contiene los tipos de problemas [1]-Sistemas, [2]-IT

<<**Areas**>> Áreas de la empresa, a la que se vincula a cada Empleado. Por ejemplo: [1]-Administración, [2]-Producción, [3]-Envasado, [4]-Recursos Humanos, [5]-Sistemas, [6]-Ventas, [7]-Dirección y [8]-Compras.

<<**Empleados**>> Contiene información del empleado: Nombre, Apellido, fecha de ingreso, fecha de egreso, área.

<<**Usuarios**>> Contiene información de usuarios como mail, legajo vinculado porque todo usuario es un empleado de la empresa, clave de login, mail.

<<**Resolutores**>> Contiene listado de resolutores por tipo de problema. Se vincula a la tabla de usuarios.

2.6.3. Inserción, consulta y borrado de registros

```
INSERT INTO inc.errores (Id_Error, Id_Pantalla, codigo, Descripcion, Pasos_Reprod, Id_Tipo_Prob)
values (1, 1, '1002',
'Problema al registar factura. Se generó error 1002 por inconsistencia tipo de dato al registrar registro.',
'Luego de seleccionar el proveedor OLEOS DEL SUR, al dar tab, aparece error 1002.', 1),
(2, 22, '1000', 'Error al registrar nuevo Transporte.',
'Luego de colocar información de razón social, chasis y choferes aparece error 1000-No Data Found.', 1),
(3, 5, '1010', 'Error al registrar pago.',
'Luego de seleccionar el banco CITI cta 9987, aparece error 1010, Código de Banco es NULL.', 1)
;
commit;


INSERT INTO inc.incidentes
(Id_Incidente,  Id_Usuario,  Id_Resolutor,  Fecha ,  Anexo, Id_Error,  Prioridad,Id_Estado, Id_Solucion)
VALUES  (1, 1, NULL, '2024-05-10', NULL, 1, 2, 1,  NULL),
        (2, 2, NULL, '2024-05-10', NULL, 2, 2, 1,  NULL),
        (3, 1, NULL, '2024-05-11', NULL, 3, 2, 1,  NULL);
commit;


INSERT INTO inc.soluciones
(Id_Solucion, Desc_Corta, Desc_Larga,Id_Resolutor,Fecha,Id_Error, Tipo_Usuario,
Desc_Path, Permisos, Prerequisitos, Script, Herramienta)
VALUES ('1', 'Se agregó código de banco. ',
'Se accede por la configuración de bancos y se agrega código de banco válido.', '1', '2024-05-11', '3', 1,
'Configuración > Entidades > Bancos', 'Acceso a Modulo de Configuración', null, null, null);

commit;
```

Vincular solución con Incidente, cerrar Incidente.

```
update inc.incidentes
set Id_Solucion = 1, Id_Estado = 4 /*Cerrado*/, Id_Resolutor = 1 /*GLAZARINI*/
where Id_Incidente = 3;
commit;
```

2.6.4. Presentación de las consultas SQL

A continuación, se dispone de una consulta sql con la totalidad de tablas vinculadas.

```
select i.Id_Incidente "Nro_Incidente", i.Fecha "Fecha_Incidente", i.prioridad "Prioridad", err.Descripcion "Error",
Est.Desc_Estado "Estado", Sol.Desc_Larga "Solucion",
case when sol.Tipo_Usuario is null then  ""     when sol.Tipo_Usuario is true then  "De_Usuario"  else 'Tecnica' end "Tipo_Solucion",
ap.desc_aplicacion "Aplicación", mo.Desc_Modulo "Modulo", pro.Desc_Proceso "Proceso", pan.Desc_Pantalla "Pantalla",
usu.Usuario "Usuario", ar.Desc_Area "Area"
from   inc.incidentes i
inner join inc.errores err on i.Id_Error= err.Id_Error
inner join inc.estados est on i.Id_Estado = est.Id_Estado
left join  inc.Soluciones sol on sol.Id_Solucion = i.Id_Solucion
inner join inc.pantallas pan on err.Id_Pantalla = pan.Id_Pantalla
inner join inc.procesos pro on pan.Id_Proceso = pro.Id_Proceso
inner join inc.modulos mo on pro.Id_Modulo = mo.Id_Modulo
inner join inc.aplicaciones ap on mo.Id_Aplicacion = ap.Id_Aplicacion
left join inc.resolutores res on res.Id_Resolutor = i.Id_Resolutor
inner join inc.usuarios usu on i.Id_Usuario = usu.Id_Usuario
inner join inc.empleados em on usu.legajo = em.legajo
inner join inc.areas ar on em.Id_Area = ar.Id_Area;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Nro_Incic	Fecha_Incidente	Prio	Error	Estado	Solucion	Tipo_Solucion	Aplicación	Modulo	Proceso
1	2024-05-10	2	Problema al registar factura. Se generó error 1002 por inconsistencia tipo de dato a...	Abierto	NULL		Sys_Admin	Administración	Ingreso Facturas
2	2024-05-10	2	Error al registrar nuevo Transporte.	Abierto	NULL		Sales System	Pedidos	Despachar
3	2024-05-11	2	Error al registrar pago.	Cerrado	Se accede por la confi...	De_Usuario	Sys_Admin	Pagos	Ingreso Pagos
4	2024-05-15	2	Error al registrar pago.	Abierto	NULL		Sys_Admin	Pagos	Ingreso Pagos

Se procede a eliminar los datos de prueba creados en las tablas Errores, Incidentes y Soluciones. Se constata eliminación:

```
delete from inc.incidentes WHERE (Id_Incidente = '1');
delete from inc.errores where (Id_Error = '1');
delete from inc.incidentes where  (Id_Incidente = '2');
delete from inc.errores where (Id_Error = '2');
delete from inc.soluciones where (Id_Solucion = '1');
delete from inc.incidentes where (Id_Incidente = '3');
delete from inc.errores where (Id_Error = '3');
delete from inc.incidentes where (Id_Incidente = '4');
delete from inc.errores where (Id_Error = '4');

commit;
```

```
select * from inc.incidentes;
select * from inc.soluciones;
select * from inc.errores;
```

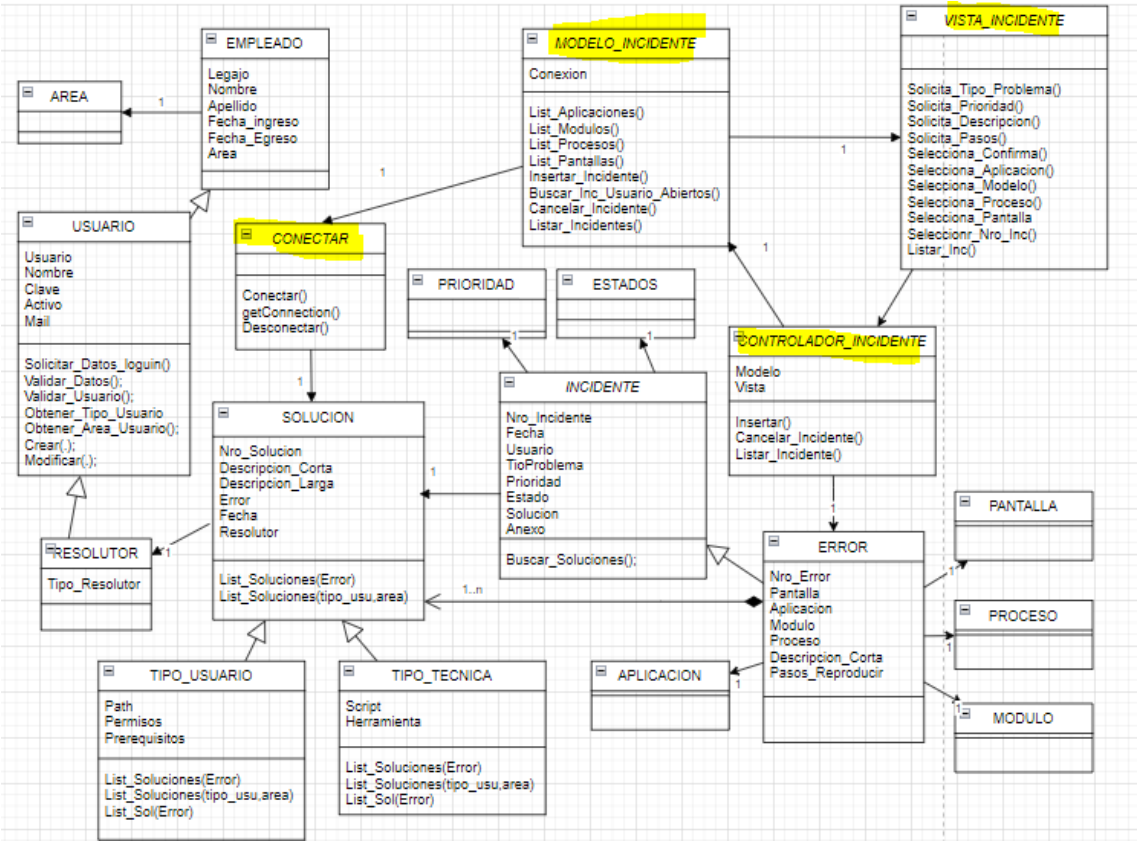
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Id_Error	Id_Pantalla	Codigo	Descripcion	Pasos_Reprod	Id_Tipo_Prob
NULL	NULL	NULL	NULL	NULL	NULL

3. Programación Orientada a Objetos

El desarrollo en esta primera instancia contemplará las siguientes clases principales: Incidentes, Errores y Soluciones Tipo Usuario y Tipo Técnica. Además, se agrega una clase para manejo de Menú y otra para conexión a la base de datos, para obtener información de aplicaciones, módulos, procesos, pantallas, usuario, etc.

También se agregan las clases del modelo **MVC**:



A continuación, se explica la programación de la clase **Incidentes**. Esta clase tiene sus atributos y su correspondiente constructor.

3.1. Clase Incidentes

```
package Incidentes;
//Nombre: Valeria Bijarra
import java.sql.SQLException;

public class Incidentes {
    protected int Nro_Incidente;
    protected  java.sql.Date  Fecha;
    protected int Usuario;
    protected int Tipo_Problema;
    protected int Prioridad;
    protected int Estado;
    protected int Solucion;

    public Incidentes (int Nro_Incidente,
            java.sql.Date  fecha2,
                    int Usuario,
                    int Tipo_Problema,
                    int Prioridad,
                    int Estado,
                    int Solucion
                    ) {
        this.Nro_Incidente= Nro_Incidente;
        this.Fecha= fecha2;
        this.Usuario= Usuario;
        this.Tipo_Problema= Tipo_Problema;
        this.Prioridad= Prioridad;
        this.Estado= Estado;
        this.Solucion = Solucion;
    };
```

Atributos

Constructor

Se realiza **encapsulamiento** con **protected** ya que esta es una clase padre, limitando el acceso a los atributos.

Getters y setters

```
public int getNro_Incidente() {
    return Nro_Incidente;
}

public void setNro_Incidente(int nro_Incidente) {
    Nro_Incidente = nro_Incidente;
}

public java.sql.Date getFecha() {
    return Fecha;
}

public void setFecha( java.sql.Date fecha) {
    Fecha = fecha;
}

public int getUsuario() {
    return Usuario;
}

public void setUsuario(int usuario) {
    Usuario = usuario;
}

public int getTipo_Problema() {
    return Tipo_Problema;
}
```

```
public int getPrioridad() {
    return Prioridad;
}

public void setPrioridad(int prioridad) {
    Prioridad = prioridad;
}

public int getEstado() {
    return Estado;
}

public void setEstado(int estado) {
    Estado = estado;
}
```

La clase **Incidentes** tiene los siguientes métodos:

3.1.1. Buscar_Soluciones()

```
public void Buscar_Soluciones(Errores err_inc, int tipo_usu) throws SQLException {
    if (tipo_usu == 1) {
        Soluciones solu= new Tipo_Usuario();
        solu.List_Soluciones(err_inc);
    } else {
        Soluciones solu= new Tipo_Tecnica();
        solu.List_Soluciones(err_inc);
    }
}
```

Este método dependiendo del tipo de usuario hace uso de las soluciones registradas en caso de ser resolutor, busca soluciones técnicas, y si es usuario busca soluciones de usuario.

El método **Buscar_Soluciones()** recibe como parámetros un objeto **Errores** y el tipo de usuario que puede ser 1 (usuario) o 2 (resolutor). Instancia a la clase **Soluciones** según sea tipo usuario y ejecuta el método **List_Soluciones()** pasando como parámetro el tipo **Errores**.

El parámetro **tipo_usu** permite identificar si las soluciones a mostrar son del tipo usuario o técnicas. Este parámetro se obtiene en el método **Main()**, cuando se realiza el loguin del usuario, que se realiza en la clase **Menú_New**.

3.2. Clase Errores

Con sus atributos y constructor, hace herencia de atributos de la clase **Incidentes**. A continuación, su declaración:

```
package Incidentes;
//Nombre: Valeria Bijarra
import java.sql.Date;

public class Errores extends Incidentes {
    String Nro_Error;
    int Pantalla;
    int Aplicacion;
    int Modulo;
    int Proceso;
    String Descripcion_Corta;
    String Pasos;
    int Sec_Err;

    public Errores(int Nro_Incidente,  java.sql.Date Fecha, int Usuario,    int Tipo_Problema,
            int Prioridad,int Estado,int Solucion,
            String NumeroError, int Pantalla, int Aplicacion, int Modulo,
            int Procceso, String Descripcion_Corta, String Pasos, int Sec_Err)
    {
        super(Nro_Incidente, Fecha, Usuario, Tipo_Problema, Prioridad, Estado, Solucion);
        this.Nro_Error=NumeroError;
        this.Pantalla=Pantalla;
        this.Aplicacion=Aplicacion;
        this.Modulo=Modulo;
        this.Proceso=Proceso;
        this.Descripcion_Corta=Descripcion_Corta;
        this.Pasos=Pasos;
        this.Sec_Err=Sec_Err;
    }
```

Clase que hereda de Incidentes

Atributos

Constructor

Getters and setters

```
public String getNro_Error() {
    return Nro_Error;
}

public void setNro_Error(String nro_Error) {
    Nro_Error = nro_Error;
}

public int getPantalla() {
    return Pantalla;
}

public void setPantalla(int pantalla) {
    Pantalla = pantalla;
}

public int getAplicacion() {
    return Aplicacion;
}

public void setAplicacion(int aplicacion) {
    Aplicacion = aplicacion;
}

public int getModulo() {
    return Modulo;
}
```

```
public void setModulo(int modulo) {
    Modulo = modulo;
}

public int getProceso() {
    return Proceso;
}

public void setProceso(int proceso) {
    Proceso = proceso;
}

public String getDescripcion_Corta() {
    return Descripcion_Corta;
}

public void setDescripcion_Corta(String descripcion_Corta) {
    Descripcion_Corta = descripcion_Corta;
}

public String getPasos() {
    return Pasos;
}

public void setPasos(String pasos) {
    Pasos = pasos;
}
```

```
public int getSec_Err() {
    return Sec_Err;
}

public void setSec_Err(int sec_Err) {
    Sec_Err = sec_Err;
}
```

3.3. Clase Soluciones

```
package Soluciones;
import java.sql.SQLException;
import java.util.Date;
import java.util.Scanner;
import Incidentes.Errores;

public abstract  class Soluciones {
    protected int Nro_Solucion;
    protected String Desc_Corta;
    protected String Desc_Larga;
    protected int Error;
    protected Date Fecha;
    protected int Resolutor;
```

Atributos

```
    public Soluciones (int Nro_Solucion,    String Desc_Corta,  String Desc_Larga,  int Error,  Date Fecha, int Resolutor){
        this.Nro_Solucion= Nro_Solucion;
        this.Desc_Corta= Desc_Corta;
        this.Desc_Larga= Desc_Larga;
        this.Error= Error;
        this.Fecha= Fecha;
        this.Resolutor= Resolutor;
};
```

Constructor

Se realiza **encapsulamiento** con **protected** ya que esta es una clase padre, limitando el acceso a los atributos.

Métodos de la clase **Soluciones**:

3.3.1. Abstract List_Soluciones()

```
public abstract void List_Soluciones(Errores err) throws SQLException;
public abstract void List_Soluciones(int tipo_usu, int id_area) throws SQLException;
```

Son métodos **Abstractos** que están implementados en las clases hijas (Clase Tipo_Usuario y clase Tipo_Tecnica). Haciendo uso de **polimorfismo** se crearon dos

métodos con el mismo nombre, pero con diferente implementación: El primero recibe el objeto errores para realizar búsqueda de soluciones a partir del error ingresado.
El segundo lista soluciones según tipo de usuario y área del usuario logueado.

3.4. Clase Tipo_Tecnica

Es clase hija de la clase **Soluciones**.

```
package Soluciones;
import java.sql.ResultSet;

public class Tipo_Tecnica extends Soluciones{
    String Script;
    String Herramienta;

    public Tipo_Tecnica(int Nro_Solucion, String Desc_Corta, String Desc_Larga, int Error, Date Fecha, int Resolutor,
            String Script, String Herramienta) {

        super(Nro_Solucion, Desc_Corta, Desc_Larga, Error, Fecha, Resolutor);
        this.Script=Script;
        this.Herramienta=Herramienta;
    }
```

Hereda de clase Soluciones

Atributos

Constructor

Método de la clase **Tipo_Tecnica**:

3.4.1. List_Soluciones(Errores)

```
@Override
public void List_Soluciones(Errores err_inc) {
    try {
    ArrayList<Tipo_Tecnica> Soluciones_T_Encontradas =  List_Sol(err_inc);
    if (Soluciones_T_Encontradas.size()>0) {
        System.out.println("_____________________________________________________________");
        System.out.println("Soluciones Tecnicas Encontradas");
        System.out.println("_____________________________________________________________");
        System.out.println(String.format("%-10s", "N Solución") + " | " +  String.format("%-30s", " Descripción Corta")
        + " | " + String.format("%-90s", "Descripción Larga") + " | " + String.format("%-90s","Script")
        + "   | " + String.format("%-30s","Herramiena"));

       for (int i = 0; i<Soluciones_T_Encontradas.size(); i++)
                System.out.println( String.format("%-10s", Soluciones_T_Encontradas.get(i).Nro_Solucion)  + " | " +
                                    String.format("%-30s", Soluciones_T_Encontradas.get(i).Desc_Corta) + " | " +
                                     String.format("%-90s", Soluciones_T_Encontradas.get(i).Desc_Larga) + " | " +
                                     String.format("%-90s", Soluciones_T_Encontradas.get(i).Script) + " | " +
                                     String.format("%-30s", Soluciones_T_Encontradas.get(i).Herramienta) );

       System.out.println("_____________________________________________________________");
        System.out.println("[1] Regresar al Menú.  [2] Salir." );
        int confirma=teclado.nextInt();
        if (confirma != 1) {
            return;
        }
    } else {System.out.println("No Se encontraron Soluciones.");
    System.out.println("[1] Regresar al menú.  [2] Salir." );
    int confirma=teclado.nextInt();
    if (confirma != 1) {
        return;
    }
    }
```

Recibe el objeto Errores, y llama al método **List_Sol()** para cargar el array de tipo **Tipo_Tecnica** con las soluciones técnicas encontradas.

3.4.2. List_Sol()

```
public  ArrayList<Tipo_Tecnica> List_Sol (Errores err) throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    Conectar db= new Conectar();
    ArrayList<Tipo_Tecnica> soluciones = new ArrayList<Tipo_Tecnica>();
    try {
        ps = db.getConnection().prepareStatement("select Id_Solucion,  Desc_Corta, Desc_Larga, Id_Resolutor, Fecha, "
                + " s.Id_Error as Id_Error,  script, herramienta"
                + " from soluciones s, errores e "
                + " where s.Id_Error = e.Id_Error and s.tipo_usuario = 2  and e.Id_Pantalla = " + err.getPantalla() + ";");
        rs=ps.executeQuery();
        while (rs.next()) {
            int Id_Solucion = rs.getInt("Id_Solucion");
            String Desc_Corta = rs.getString("Desc_Corta");
            String Desc_Larga = rs.getString("Desc_Larga");
            int Id_Resolutor= rs.getInt("Id_Resolutor");
            Date Fecha=rs.getDate("Fecha");
            int Id_Error=rs.getInt("Id_Error");
            String Script=rs.getString("Script");
            String Herramienta=rs.getString("herramienta");
            Tipo_Tecnica sol = new Tipo_Tecnica(Id_Solucion, Desc_Corta, Desc_Larga, Id_Error,  Fecha,  Id_Resolutor,
                                                Script, Herramienta);
            soluciones.add(sol);
        }
    } catch (Exception ext) {System.out.println("Error List_Soluciones_Usuario:" + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        db.desconectar(); }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return soluciones;
}
```

Este método recibe un objeto del tipo **Errores** y a partir de este realiza búsqueda en la base de datos. El ResultSet es almacenado en un nuevo objeto de tipo **Tipo_Tecnica** para luego agregarlo en el array del mismo tipo. Este método es invocado por el método **List_Soluciones**() de la clase **Tipo_Tecnica**.

3.4.3. List_Soluciones(Tipo_Usu, Id_Area)

```
@Override
public void List_Soluciones(int tipo_usu, int id_area) throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    Conectar db= new Conectar();

     try {
            Aplicaciones apl = new Aplicaciones();
            int aplicacion_sel= apl.seleccionar_Apliacion(apl);
            Modulos mod= new Modulos();
            int modulo_Sel = mod.seleccionar_Modulo(mod, aplicacion_sel);
            Procesos  prod= new Procesos();
            int proceso_Sel = prod.seleccionar_Proceso(prod, modulo_Sel);
            Pantallas  pant= new Pantallas();
            int pantalla_Sel = pant.seleccionar_Pantalla(pant, proceso_Sel);

             System.out.println("_____________________________________________________________");
             System.out.println("Soluciones Tecnicas Encontradas");
             System.out.println("_____________________________________________________________");


             System.out.println(String.format("%-10s", "Solucion") + " | " +  String.format("%-30s", " Descripción Inc")    +
                    " | " + String.format("%-20s", "Aplicacion") + " | " + String.format("%-20s","Modulo")  +
                    " | " + String.format("%-20s","Proceso") + " | " + String.format("%-20s","Pantalla") +
                    " | " + String.format("%-20s","Desc_Area") + " | " + String.format("%-90s","Descripción Sol") +
                    " | " + String.format("%-50s","Script") + " | " + String.format("%-50s","Herramienta") +
                    " | " + String.format("%-50s","Path") + " | " + String.format("%-50s","Permisos") +
                    " | " + String.format("%-50s","Prerequisitos"));
```

```
                ps = db.getConnection().prepareStatement("select sol.id_solucion, err.Descripcion, "
                        + "   ap.desc_aplicacion , mo.Desc_Modulo , pro.Desc_Proceso , pan.Desc_Pantalla,  "
                        + "   ar.Desc_Area, Sol.Desc_Larga,  sol.desc_path, sol.permisos, sol.Prerequisitos, sol.script, "
                        + "   sol.herramienta   "
                        + " from   inc.incidentes i "
                        + " inner join inc.errores err on i.Id_Error= err.Id_Error    "
                        + " inner join inc.estados est on i.Id_Estado = est.Id_Estado "
                        + " inner join inc.pantallas pan on err.Id_Pantalla = pan.Id_Pantalla  "
                        + " inner join inc.procesos pro on pan.Id_Proceso = pro.Id_Proceso "
                        + " inner join inc.modulos mo on pro.Id_Modulo = mo.Id_Modulo "
                        + " inner join inc.aplicaciones ap on mo.Id_Aplicacion = ap.Id_Aplicacion  "
                        + " inner join  inc.Soluciones sol on sol.Id_Solucion = i.Id_Solucion "
                        + " inner join  inc.resolutores res on res.Id_Resolutor = i.Id_Resolutor "
                        + " inner join inc.usuarios usu on i.Id_Usuario = usu.Id_Usuario  "
                        + " inner join inc.empleados em on usu.legajo = em.legajo  "
                        + " inner join inc.areas ar on em.Id_Area = ar.Id_Area  "
                        + " where     err.Id_Pantalla =" + pantalla_Sel );

       rs=ps.executeQuery();
```

```
        while (rs.next()) {

                System.out.println( String.format("%-10s",rs.getInt("id_solucion"))  + " | " +
                        String.format("%-30s", rs.getString("Descripcion")).substring(0, 30) + " | " +
                        String.format("%-20s", rs.getString("Desc_Aplicacion")) + " | " +
                        String.format("%-20s", rs.getString("Desc_Modulo")) + " | " +
                        String.format("%-20s", rs.getString("Desc_Proceso")) + " | " +
                        String.format("%-20s", rs.getString("Desc_Pantalla")) + " | " +
                        String.format("%-20s", rs.getString("Desc_Area")) + " | " +
                        String.format("%-50s", rs.getString("Desc_Larga")).substring(0, 50) + " | " +
                        String.format("%-50s", rs.getString("script")).substring(0, 50) + " | " +
                        String.format("%-30s", rs.getString("herramienta")).substring(0, 30) + " | " +
                        String.format("%-30s", rs.getString("desc_path")).substring(0, 30) + " | " +
                        String.format("%-30s", rs.getString("permisos")).substring(0, 30) + " | " +
                        String.format("%-30s", rs.getString("Prerequisitos")).substring(0, 30));
        }

        System.out.println("");
        System.out.println("_____________________________________________________________");
        System.out.println("[1] Consultar Soluciones nuevamente  [2] Salir." );
        int confirma=teclado.nextInt();
        if (confirma != 1) {
            db.desconectar();
            return;
        } else
        {
            List_Soluciones (tipo_usu, id_area);
        }
```

Este método despliega las soluciones encontradas para el tipo técnico según los parámetros ingresados por el usuario.

3.5. Clase Tipo_Usuario

Clase hija, Hereda de clase Soluciones.

```
package Soluciones;

import java.sql.ResultSet;

public class Tipo_Usuario extends Soluciones{
    String Path;
    String Permisos;
    String Prerequisitos;

    public Tipo_Usuario(int Nro_Solucion, String Desc_Corta, String Desc_Larga, int Error, Date Fecha, int Resolutor,
            String path, String Permisos, String Prerequisitos) {

        super(Nro_Solucion, Desc_Corta, Desc_Larga, Error, Fecha, Resolutor);
        this.Path=path;
        this.Permisos=Permisos;
        this.Prerequisitos=Prerequisitos;
    }
```

Hereda de Soluciones

Atributos

Constructor

Métodos de la clase **Tipo_Usuario**

3.5.1. List_Solucione(Errores)

```
    public void List_Soluciones(Errores err_inc) {
        try {
        ArrayList<Tipo_Usuario> Soluciones_U_Encontradas =  List_Sol(err_inc);   //List_Soluciones_Usuario
        if (Soluciones_U_Encontradas.size()>0) {
            System.out.println("_____________________________________________________________________");
            System.out.println("Soluciones de Usuario Encontradas");
            System.out.println("_____________________________________________________________________");
            System.out.println(String.format("%-10s", "N Solución") + " | " +  String.format("%-30s", " Descripción Corta")
            + " | " + String.format("%-90s", "Descripción Larga") + " | " + String.format("%-30s","Permisos")
            + "   | " + String.format("%-60s","Path") + " | " +  String.format("%-30s","Prerequisitos"));

           for (int i = 0; i<Soluciones_U_Encontradas.size(); i++)
                   System.out.println( String.format("%-10s", Soluciones_U_Encontradas.get(i).Nro_Solucion)  + " | " +
                                       String.format("%-30s", Soluciones_U_Encontradas.get(i).Desc_Corta) + " | " +
                                        String.format("%-90s", Soluciones_U_Encontradas.get(i).Desc_Larga) + " | " +
                                        String.format("%-30s", Soluciones_U_Encontradas.get(i).Permisos) + " | " +
                                        String.format("%-60s", Soluciones_U_Encontradas.get(i).Path) + " | " +
                                        String.format("%-30s", Soluciones_U_Encontradas.get(i).Prerequisitos));

            System.out.println("_____________________________________________________________________");
            System.out.println("[1] Regresar al Menú.  [2] Salir." );
            int confirma=teclado.nextInt();
            if (confirma != 1) {
                return;
            }

            } else {System.out.println("No Se encontraron Soluciones.");
                    System.out.println("[1] Regresar al menú.  [2] Salir." );
                    int confirma=teclado.nextInt();
                    if (confirma != 1) {
                        return;
                    }
            }
        } catch (Exception ext) {System.out.println("Error List_Soluciones_Usuario:" + ext.getMessage());}
    }
```

Recibe el objeto **Errores**, y llama al método **List_Sol(Errores)** para cargar el array de tipo **Tipo_Usuario** con las soluciones de usuario encontradas.

3.5.2. List_Sol(Errores)

```
public  ArrayList<Tipo_Usuario> List_Sol (Errores err) throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    Conectar db= new Conectar();
    ArrayList<Tipo_Usuario> soluciones = new ArrayList<Tipo_Usuario>();
    try {   ps = db.getConnection().prepareStatement("select Id_Solucion,  Desc_Corta, Desc_Larga, Id_Resolutor, Fecha, "
                + " s.Id_Error as Id_Error, Desc_Path,  Permisos, Prerequisitos"
                + " from soluciones s, errores e "
                + " where s.Id_Error = e.Id_Error and s.tipo_usuario  = 1  and e.Id_Pantalla = " + err.getPantalla() + ";");
        rs=ps.executeQuery();
        while (rs.next()) {
            int Id_Solucion = rs.getInt("Id_Solucion");
            String Desc_Corta = rs.getString("Desc_Corta");
            String Desc_Larga = rs.getString("Desc_Larga");
            int Id_Resolutor= rs.getInt("Id_Resolutor");
            Date Fecha=rs.getDate("Fecha");
            int Id_Error=rs.getInt("Id_Error");
            String Desc_Path=rs.getString("Desc_Path");
            String Permisos=rs.getString("Permisos");
            String Prerequisitos= rs.getString("Prerequisitos");
            Tipo_Usuario sol = new Tipo_Usuario(Id_Solucion, Desc_Corta, Desc_Larga, Id_Error,  Fecha,  Id_Resolutor,
                               Desc_Path,  Permisos, Prerequisitos);
            soluciones.add(sol);
        }
    } catch (Exception ext) {System.out.println("Error List_Soluciones_Usuario:" + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        db.desconectar();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return soluciones;
}
```

Busca soluciones en la base de datos a partir del parámetro que recibe (objeto **Errores**), crea un nuevo objeto del tipo **Tipo_Usuario** y retorna un array del mismo tipo del objeto creado. Este método es invocado por el método **List_Soluciones**() de la misma clase **Tipo_Usuario**.

3.5.3. List_Soluciones_Usuario(Tipo_Usuario, Id_Area)

```
@Override
public void List_Soluciones(int tipo_usu, int id_area)   {
        java.sql.PreparedStatement ps= null;
        ResultSet rs = null;
        Conectar db= new Conectar();
         try {
                Aplicaciones apl = new Aplicaciones();
                int aplicacion_sel= apl.seleccionar_Apliacion(apl);
                Modulos mod= new Modulos();
                int modulo_Sel = mod.seleccionar_Modulo(mod, aplicacion_sel);
                Procesos  prod= new Procesos();
                int proceso_Sel = prod.seleccionar_Proceso(prod, modulo_Sel);
                Pantallas  pant= new Pantallas();
                int pantalla_Sel = pant.seleccionar_Pantalla(pant, proceso_Sel);

                 System.out.println("_____________________________________________________________________");
                 System.out.println("Soluciones de Usuario Encontradas");
                 System.out.println("_____________________________________________________________________");

                 System.out.println(String.format("%-10s", "Solucion") + " | " +  String.format("%-30s", " Descripción")    +
                            " | " + String.format("%-20s", "Aplicacion") + " | " + String.format("%-20s","Modulo")  +
                            " | " + String.format("%-20s","Proceso") + " | " + String.format("%-20s","Pantalla") +
                            " | " + String.format("%-20s","Desc_Area") + " | " + String.format("%-90s","Descripción") +
                            " | " + String.format("%-50s","Path") + " | " + String.format("%-50s","Permisos") +
                            " | " + String.format("%-50s","Prerequisitos"));


                   ps = db.getConnection().prepareStatement("select sol.id_solucion, err.Descripcion, "
                           + "   ap.desc_aplicacion , mo.Desc_Modulo , pro.Desc_Proceso , pan.Desc_Pantalla,  "
                           + "   ar.Desc_Area, Sol.Desc_Larga,  sol.desc_path, sol.permisos, sol.Prerequisitos, sol.script,
                           + " from   inc.incidentes i "
                           + " inner join inc.errores err on i.Id_Error= err.Id_Error    "
                           + " inner join inc.estados est on i.Id_Estado = est.Id_Estado "
                           + " inner join inc.pantallas pan on err.Id_Pantalla = pan.Id_Pantalla  "
```

```
                        + " inner join inc.procesos pro on pan.Id_Proceso = pro.Id_Proceso "
                        + " inner join inc.modulos mo on pro.Id_Modulo = mo.Id_Modulo "
                        + " inner join inc.aplicaciones ap on mo.Id_Aplicacion = ap.Id_Aplicacion  "
                        + " left join  inc.Soluciones sol on sol.Id_Solucion = i.Id_Solucion "
                        + " left join  inc.resolutores res on res.Id_Resolutor = i.Id_Resolutor "
                        + " inner join inc.usuarios usu on i.Id_Usuario = usu.Id_Usuario  "
                        + " inner join inc.empleados em on usu.legajo = em.legajo  "
                        + " inner join inc.areas ar on em.Id_Area = ar.Id_Area  "
                        + " where   ar.Id_Area = " + id_area
                        + " and err.Id_Pantalla =" + pantalla_Sel
                        + " and sol.tipo_usuario = " + tipo_usu + ";");
            rs=ps.executeQuery();

            while (rs.next()) {
                    System.out.println( String.format("%-10s",rs.getInt("id_solucion"))  + " | " +
                            String.format("%-30s", rs.getString("Descripcion")).substring(0, 30) + " | " +
                            String.format("%-20s", rs.getString("Desc_Aplicacion")) + " | " +
                            String.format("%-20s", rs.getString("Desc_Modulo")) + " | " +
                            String.format("%-20s", rs.getString("Desc_Proceso")) + " | " +
                            String.format("%-20s", rs.getString("Desc_Pantalla")) + " | " +
                            String.format("%-20s", rs.getString("Desc_Area")) + " | " +
                            String.format("%-90s", rs.getString("Desc_Larga")).substring(0, 90) + " | " +
                            String.format("%-50s", rs.getString("desc_path")).substring(0, 50) + " | " +
                            String.format("%-50s", rs.getString("permisos")).substring(0, 50) + " | " +
                            String.format("%-50s", rs.getString("Prerequisitos")).substring(0, 50));
                    }
                     }

            System.out.println("");
            System.out.println("_____________________________________________________________________");
            System.out.println("[1] Consultar Soluciones nuevamente  [2] Salir." );
            int confirma=teclado.nextInt();
            if (confirma != 1) {
                db.desconectar();
                return;
            } else
            {
                List_Soluciones (tipo_usu, id_area);
            }


        } catch (Exception ext) {System.out.println("Error en Listar_Soluciones:" + ext.getMessage());}
    finally {
        try {
            ps.close();
            rs.close();
            db.desconectar();
            }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
        }
```

3.6. Clase Menu_New

Esta clase contiene el método principal del sistema, el **main()**. El método **Main()** contiene las llamadas a los métodos de loguin del usuario y el menú principal del sistema.

Métodos de la clase **Menu_New**:

3.6.1. Main()

```
package Menu;
//Nombre: Valeria Bijarra

import java.sql.SQLException;

    public class Menu_New {
        static Scanner teclado = new Scanner(System.in);
        public static void main(String args[]) throws SQLException {
        {
            int Id_usu = 0;
            int Tipo_usu=0;
            int Id_area=0;
            try {
                Usuarios usu  = new Usuarios();
                int[][] datos_usuario = new int[1][3];
                datos_usuario= usu.Solicitar_Datos_loguin();
                Id_usu= datos_usuario[0][0];
                Tipo_usu = datos_usuario[0][1];
                Id_area = datos_usuario[0][2];
                Seleccionar_Menu(Id_usu, Tipo_usu, Id_area);

            } catch (SQLException ex) {
            System.err.println("Error: " + ex.getMessage());
        }}}
```

Solicita información de loguin de usuario antes de cargar el menú.

La información que retorna el método **Solicitar_Datos_Loguin()**, es almacenada en una matriz. A partir de esto, se obtiene información del usuario: Id_Usuario, Tipo_Usuario y Id_Area.

3.6.2. Seleccionar_Menu()

```
public static void Seleccionar_Menu(int id_usu, int tipo_usu, int id_area) throws SQLException {
    boolean salir = false;

        while  (!salir)
        {
            try {
                int opcion = Menu();
                    switch (opcion) {
                    case 1:
                        System.out.println("_____________________________________________________________");
                        System.out.println("INGRESAR INCIDENTE");
                        System.out.println("_____________________________________________________________");
                        try { /*se realilza conexión a la base de datos y se nstancia a Controlador_Incidente para insertar
                                Conectar conexion= new Conectar();
                                Controlador_Incidente controlador = new Controlador_Incidente(conexion);
                                controlador.Insertar(id_usu, tipo_usu);
                                conexion.desconectar();
                        }  catch (SQLException e) {
                                System.err.println("Error de conexión a la base de datos: " + e.getMessage());
                                }
                        break;
                    case 2:
                        System.out.println("_____________________________________________________________");
                        System.out.println("MODIFICAR INCIDENTE-SIN DESARROLLAR");
                        System.out.println("_____________________________________________________________");
                        break;
                    case 3:
                        System.out.println("_____________________________________________________________");
                        System.out.println("CANCELAR INCIDENTE");
                        System.out.println("_____________________________________________________________");
```

```
                        try {/*se realiza conexión a la base de datos y se instancia a Controlador_Incidente para cancelar un 
                                Conectar conexion= new Conectar();
                                Controlador_Incidente controlador = new Controlador_Incidente(conexion);
                                controlador.Cancelar_Incidente(id_usu, id_area, tipo_usu);
                                conexion.desconectar();
                        }  catch (SQLException e) {
                            System.err.println("Error de conexión a la base de datos: " + e.getMessage());
                        }
                        break;
                    case 4:
                        System.out.println("_____________________________________________________");
                        System.out.println("RESOLVER INCIDENTE-SIN DESARROLLAR");
                        System.out.println("_____________________________________________________");
                        break;
                    case 5:
                        System.out.println("_____________________________________________________");
                        System.out.println("LISTAR INCIDENTES");
                        System.out.println("_____________________________________________________");

                        try {/*se realiza conexión a la base de datos y se instancia a Controlador_Incidente para listar incid
                                Conectar conexion= new Conectar();
                                Controlador_Incidente controlador = new Controlador_Incidente(conexion);
                                controlador.Listar_Incidentes(id_usu, id_area, tipo_usu);
                                conexion.desconectar();
                            }catch (SQLException e) {
                            System.err.println("Error de conexión a la base de datos: " + e.getMessage());
                            }
                        break;
                    case 6:
                        System.out.println("_____________________________________________________");
                        System.out.println("LISTAR SOLUCIONES");
                        System.out.println("_____________________________________________________");

                        if (tipo_usu == 1) {
                            Soluciones solu= new Tipo_Usuario();
                            solu.List_Soluciones(id_usu, id_area);
                        } else {
                            Soluciones solu= new Tipo_Tecnica();
                            solu.List_Soluciones(id_usu, id_area);
                        }

                        break;
                    case 7:
                        System.out.println("_____________________________________________________");
                        System.out.println("CONFIGURACIONES");
                        System.out.println("_____________________________________________________");
                        break;
                    case 0:
                        System.out.println("Saliendo de la aplicación.");
                        salir=true;
                        break;
                    default:
                        System.out.println("Opción no encontrada.");
                        }
                }catch (InputMismatchException ex) {
                    System.out.println("Debe ingresar obligatoriamente un número entero.");
                    teclado.next();
                    salir=false;
                }
            }
        teclado.next();
```

Permite que el usuario ingrese una opción de menú valida. Tiene manejo de excepción del tipo *InputMismatchException*, por si el usuario ingresa un valor diferente a numérico.

Dependiendo de la opción de menú que seleccione el usuario se instancian las diferentes clases y métodos del sistema.

3.6.3. Menu()

Despliega las opciones de menú principal:

```
public static int Menu() {
    //se imprimen opciones de menú
    System.out.println("_____________________________________________________");
    System.out.println("GESTIÓN DE INCIDENTES - OLEOSIN            ");
    System.out.println("_____________________________________________________");
    System.out.println("MENU PRINCIPAL              ");
    System.out.println("_____________________________________________________");
    System.out.println("[1] Crear Incidente");
    System.out.println("[2] Modificar Incidente-sin desarrollar");
    System.out.println("[3] Cancelar Incidente");
    System.out.println("[4] Resolver Incidente-sin desarrollar");
    System.out.println("[5] Listar Incidentes");
    System.out.println("[6] Listar Soluciones");
    System.out.println("[0] Salir");
    System.out.println("_____________________________________________________");
    System.out.println("Seleccione una opción.");
    return teclado.nextInt();

    }
```

3.7. Clase Usuarios

```
package Seguridad;
//Nombre: Valeria Bijarra
import java.sql.ResultSet;

public class Usuarios {
    protected int Id_Usuario;
    protected String Usuario;
    protected String Estado;
    protected String Clave;
    protected int Legajo;

    public Usuarios(int Id_Usuario,
     String Usuario,
     String Estado,
     String Clave) {
        this.Id_Usuario= Id_Usuario;
        this.Usuario=Usuario;
        this.Estado = Estado;
        this.Clave = Clave;
    }
```

Atributos

Constructor

Se realiza **encapsulamiento** con **protected** limitando el acceso a los atributos.

A continuación, los métodos de la clase Usuarios

3.7.1. Solicitar_Datos_Loguin()

```
public int[][] Solicitar_Datos_loguin() throws SQLException {
    Scanner teclado = new Scanner (System.in);

    System.out.println("_____________________________________________________");
    System.out.println("GESTIÓN DE INCIDENTES - OLEOSIN            ");
    System.out.println("_____________________________________________________");
    System.out.println("ACCESO              ");
    System.out.println("_____________________________________________________");

    System.out.println("Usuario:");
    String usuario=teclado.nextLine();
    System.out.println("Clave:");
    String clave=teclado.nextLine();

    int id_u = Validar_Datos(usuario, clave);
    int tipo_u = Obtener_Tipo_Usuario(id_u);
    int id_area = Obtener_Area_Usuario(id_u);

    int[][] infoUsuario = new int[1][3];
    infoUsuario[0][0]=  id_u;
    infoUsuario[0][1]= tipo_u;
    infoUsuario[0][2]=id_area;

    return infoUsuario;
}
```

Solicita información de loguin de usuario para retornar a la clase **Menu_New** método **main()** una matriz con información de **id_u** (Id_usuario), **tipo_u** (tipo de usuario 1=usuario, 2=resolutor) e **id_area**.

3.7.2. Validar_Datos()

```
public int Validar_Datos(String usuario, String clave) throws SQLException {
    int id_usu=0;
    if (usuario!= null && clave != null) {
        id_usu=Validar_Usuario(usuario, clave);

        if (id_usu== 0) {
            System.out.println("Datos incorrectos.");
            Solicitar_Datos_loguin();
        }
    }
    return id_usu;
}
```

Recibe información de la clase **Menu_New**, método **Main()** usuario y clave y llama al método **Validar_Usuario()** para corroborar información en la base de datos. Retorna Id_Usuario al método **Solicitar_Datos_loguin()**. En caso de que la información sea incorrecta, vuelve a solicitar información.

3.7.3. Validar_Usuario()

```
public  int Validar_Usuario (String usuario, String clave) throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    Conectar db= new Conectar();
    int Id_Usuario=0;
    try {
        ps = db.getConnection().prepareStatement("select Id_Usuario  from usuarios where usuario =  "  + "'"+ usuario + "'"
                                                 + " and clave = " + clave
                                                 + ";");
        rs=ps.executeQuery();

        while (rs.next()) {

            Id_Usuario = rs.getInt("Id_Usuario");
        }

    } catch (Exception ext) {System.out.println("Error Validar Usuario:" + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        db.desconectar();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return Id_Usuario;
}
```

Valida información ingresada por el usuario (usuario y clave) en la tabla **usuarios**. Retorna id_usuario al método **Validar_Datos()**.

3.7.4. Obtener_Tipo_Usuario()

```
public int Obtener_Tipo_Usuario(int id_usu)  throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    Conectar db= new Conectar();
    int Tipo_Usuario=0;
    try {
        ps = db.getConnection().prepareStatement("select Tipo_Usuario  "
                                                 + " from usuarios "
                                                 + " where Id_Usuario = " + id_usu);
        rs=ps.executeQuery();

        while (rs.next()) {
            Tipo_Usuario = rs.getInt("Tipo_Usuario");
        }


    } catch (Exception ext) {System.out.println("Error :Obtener_Tipo_Usuario " + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        db.desconectar();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return Tipo_Usuario;
}
```

Obtiene tipo de usuario de la tabla **usuarios** y lo retorna al método **Solicitar_Datos_loguin()**.

3.7.5. Obtener_Area_Usuario()

```
public int Obtener_Area_Usuario(int id_usu)  throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    Conectar db= new Conectar();
    int Id_Area=0;
    try {
        ps = db.getConnection().prepareStatement("select Id_Area  "
                                                 + " from usuarios u, empleados e "
                                                 + " where u.legajo = e.legajo and u.Id_Usuario = " + id_usu);
        rs=ps.executeQuery();

        while (rs.next()) {
            Id_Area = rs.getInt("Id_Area");
        }


    } catch (Exception ext) {System.out.println("Error :Obtener_Area_Usuario " + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        db.desconectar();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return Id_Area;
}
```

Busca id_area del usuario en la tabla de **empleados** y lo retorna al método **Solicitar_Datos_loguin()**.

3.8. Clase Empleados

```
package Seguridad;
//Nombre: Valeria Bijarra
import java.util.Date;

public class Empleados extends Usuarios{   Hereda Atributos y métodos de la clase Usuarios
    String Nommbre;
    String Apellido;
    Date Fecha_Ingreso;                    Atributos
    Date Fecha_Egreso;
    int Area;
    int Legajo;

    public Empleados(int Id_Usuario, String Usuario, String Estado, String Clave,
            String Nommbre, String Apellido,  Date Fecha_Ingreso,  Date Fecha_Egreso, int Area, int Legajo) {

        super( Id_Usuario,  Usuario,  Estado,  Clave);
                                                          Constructor
        this.Nommbre=Nommbre;
        this.Apellido=Apellido;
        this.Fecha_Ingreso=Fecha_Ingreso;
        this.Fecha_Egreso = Fecha_Egreso;
        this.Area = Area;
        this.Legajo = Legajo;
    }
```

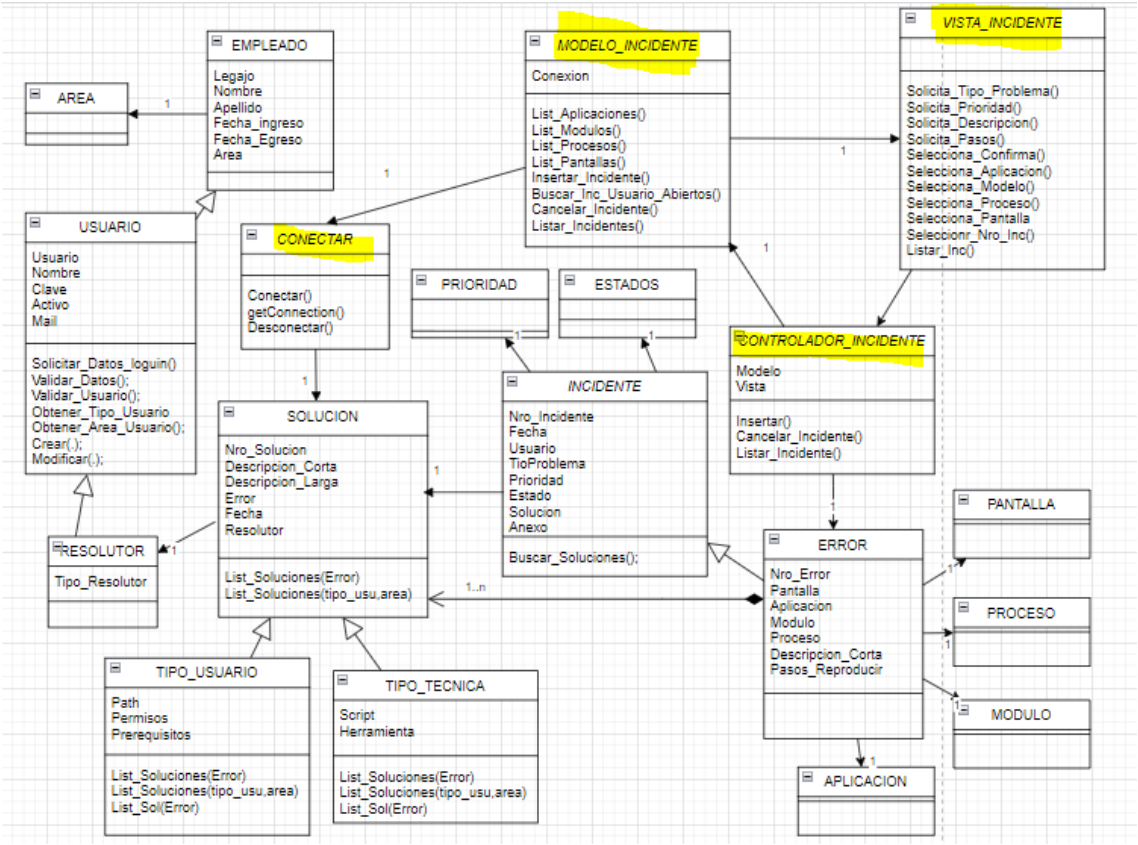
4. Patrones de Diseño MVC y JDBC

Un patrón de diseño de software proporciona una estructura predefinida que ayuda a resolver de manera eficiente los problemas que se presentan en el desarrollo de una aplicación. Para el desarrollo del sistema de gestión de incidentes, se definió utilizar el patrón MVC (modelo-vista-controlador). De esta forma se cumplirá con el requerimiento no funcional **RNFS-BD02**- Para la persistencia y consulta de datos en la BD.

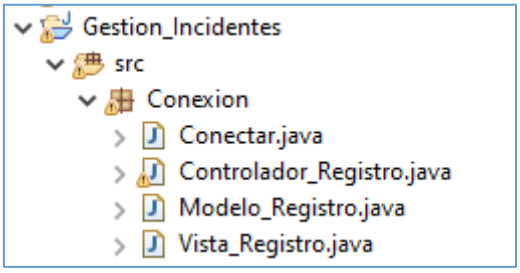
MVC es un patrón de diseño que se estructura mediante 3 componentes: modelo, vista, controlador. Este patrón tiene como principio que cada componente no se puede combinar dentro de una misma clase.

En este proyecto se va a crear un paquete independiente que utiliza el patrón **MVC,** solo para el registro de Incidentes, cancelación y búsqueda de incidentes en la base de datos.

Se agrega el modelo controlador MVC al diagrama de clases y se reestructura la clase de Incidente definida en la etapa de análisis (los métodos que consultan base de datos).



4.1. Paquete de conexión a la BD:



4.2. Clase Conectar

La clase conectar contiene los atributos necesarios para realizar la conexión con la base de datos y los métodos correspondientes para **conectar()**, **desconectar()** y **getConnection()**.

```
package Conexion;
//Nombre: Valeria Bijarra
import java.sql.Connection;


public class Conectar {
    static String db = "inc";
    static String login = "root";
    static String password = "Vb0336305";
    static String url = "jdbc:mysql://localhost:3306/" + db +  "?autoReconnect=true&useSSL=false";

    Connection  connection = null;

    public Conectar() {
        try {
            connection=DriverManager.getConnection(url, login, password);
            if (connection != null){
            }
            }catch (SQLException ex) {System.out.println(ex.getMessage());
        } catch (Exception ex) {System.out.println(ex.getMessage());}
    }

    public Connection getConnection() {
        return connection;
    }

    public void desconectar() {
        try{
            connection.close();
        } catch (Exception ex) {System.out.println(ex.getMessage());}
    }
}
```

4.3. Clase Controlador_Incidente

Es parte del patrón MVC y se encarga de controlar las interacciones con la base de datos. Se sitúa entre el modelo que gestiona la base de datos y la vista que gestiona la interfaz de usuario.

Contiene dos atributos privados y el construtor.

Modelo: Es una instancia de la clase **Modelo_Incidente**. Se utiliza para gestionar los datos, consultas e inserciones necesarias.

Vista: Es una instancia de la clase **Vista_Incidente**: Se utiliza para solicitar información y mostrar información al usuario.

```
package Conexion;
import java.sql.SQLException;

public class Controlador_Incidente {
    private Modelo_Incidente modelo;
    private Vista_Incidente vista;

    public Controlador_Incidente(Conectar conexion) {
        this.modelo = new Modelo_Incidente(conexion);
        this.vista = new Vista_Incidente();
    }
```

4.3.1.Método Insertar:

```
public void Insertar(int usu, int tipo_usu) throws SQLException {
    /*solicita información al usuario */
    int tipo= vista.Solicita_Tipo_Problema();
    int sec_inc = modelo.Incr_Id_Incidentes();
    int sec_err = modelo.Incr_Id_Errores();
    LocalDate fecha = LocalDate.now();  //obtiene fecha del sistema;
    java.sql.Date fecha_actual = java.sql.Date.valueOf(fecha);
    int prioridad= vista.Solicita_Prioridad();
    int estado = 1; /*Abierto*/
    Incidentes nuevo_incidente = new Incidentes(sec_inc,fecha_actual, usu, tipo, prioridad , estado, 0);

    String descripcion=vista.Solicita_Descripcion();
    String pasos = vista.Solicita_Pasos();
    String codigo = vista.Solicita_Codigo();
    int aplicacion_sel= vista.seleccionar_Apliacion(modelo);
    int modulo_Sel = vista.seleccionar_Modulo(modelo, aplicacion_sel);
    int proceso_Sel = vista.seleccionar_Proceso(modelo, modulo_Sel);
    int pantalla_Sel = vista.seleccionar_Pantalla(modelo, proceso_Sel);

    int confirma= vista.Seleccionar_Confirma();
    if (confirma == 1) {
        Errores Nuevo_Error = new Errores(nuevo_incidente.getNro_Incidente(), nuevo_incidente.getFecha(),
                nuevo_incidente.getUsuario(), nuevo_incidente.getTipo_Problema(),  nuevo_incidente.getPrioridad(),
                nuevo_incidente.getEstado(), nuevo_incidente.getSolucion() ,
                codigo, pantalla_Sel,  aplicacion_sel,  modulo_Sel,  proceso_Sel,  descripcion,  pasos, sec_err);

        modelo.Insertar_Incidente(Nuevo_Error);  /*realiza insert en la bd*/
        Nuevo_Error.Buscar_Soluciones(Nuevo_Error, tipo_usu);       /*busca soluciones coincidentes según el tipo de usuario*/
        } else {
                int conf=vista.Seleccionar_Confirma_Reg_Incidente();
                if (conf == 1) {
                     Insertar(usu, tipo_usu);
                } else {return;
```

El método permite llamar a métodos de la clase **Vista_Incidente**, para que el usuario ingrese información del incidente: descripción, aplicación, modulo, proceso, pantalla, usuario, etc. Donde, además, se realiza validación de la información ingresada, para evitar errores en la inserción de las tablas Incidentes y Errores.

El método crea los objetos **nuevo_Incidente** y **nuevo_error,** para enviar el objeto a la clase **Modelo_Incidente** e insertar en la base de datos con el método **Insertar_Incidente()**.

Finalmente se realiza búsqueda de las soluciones con el método **Buscar_Soluciones**(**Inc**, **tipo_usu**) de la clase **Soluciones**.

4.3.2. Cancelar_Incidente()

```
public void Cancelar_Incidente(int id_usu, int id_area, int tipo_usu) throws SQLException {
    int nro = 0;
    int inc_ok=0;

    ArrayList<Integer> List_Inc_Abi_Area =  modelo.Buscar_Inc_Usuario_Abiertos(id_usu, id_area, tipo_usu); /*busca incidentes ab
    try {
        if (List_Inc_Abi_Area.size()>0) {
            nro=vista.Seleccionar_Nro_Incidente();   /*solicita al usuario núero de incidente para cancelar*/
            for (int x=0;  (x < List_Inc_Abi_Area.size() && inc_ok==0); x++) {

                if (List_Inc_Abi_Area.get(x).equals(nro)) { /*recorre el array y lo compara con el npumero ingresado por el usua
                    modelo.Cancela_Incidente(nro); /*modifica estado del incidente en la bd*/
                    System.out.println("Incidente CANCELADO. NRO : " +  nro + ".");
                    List_Inc_Abi_Area.remove(x);   //eliminar del array el nro de incidente
                    inc_ok = 1;
                    int confirma= vista.Seleccionar_Cancela_Otro_Incidente();

                    if (confirma != 1) {
                        return;
                    } else
                    {
                        Cancelar_Incidente(id_usu, id_area, tipo_usu);
                    }
                }
            }
            if (inc_ok == 0) {
                int confirma=vista.Seleccionar_mje_nro_Inc_Incorrecto_Cancel();
                if (confirma != 1) {
                    return;
                } else
                {
                    Cancelar_Incidente(id_usu, id_area,  tipo_usu);  //vuelve a llamar al método para cancelar otro incidente
```

El método realiza una búsqueda de incidentes abiertos por el usuario con el método **Buscar_Inc_Usuario_Abiertos()** de la clase **Modelo_Incidente**. El resultado es mostrado por pantalla. Los ID de incidentes encontrados, son almacenados en un ArrayList, esto permite luego recorrer el array, encontrar el incidente ingresado por el usuario para realizar la cancelación en **modelo.cancela_Incidente(nro).**
Al confirmar la cancelación del incidente, se muestra mensaje al usuario y se elimina el **Id_Incidente** del Array_List.

El sistema pregunta si desea Cancelar otro incidente o retornar al menú.

4.3.3.Listar_Incidentes()

```
public void Listar_Incidentes(int id_usu, int id_area, int id_tipo_usu) throws SQLException {
    try {
        vista.Listar_Inc(modelo.Listar_Incidentes(id_usu, id_area, id_tipo_usu)); /*busca incidentes y los lista */
        int confirma=vista.Seleccionar_retorna_Menu();
        if (confirma != 1) {
            return;
        }else {Listar_Incidentes (id_usu, id_area, id_tipo_usu); }
    } catch (Exception ext) {System.out.println("Error Buscar_Inc_Usuario_Abiertos:" + ext.getMessage());}
}
```

4.4. Clase Vista_Incidente

Es parte del modelo **MVC** y se utiliza para manejar la iteracción con el usuario con el fin de registrar nuevos incidentes en la base de datos.

Hace uso de una variable de tipo Scanner para que el usuario pueda ingresar información.

Posee todos los métodos para seleccionar opciones preseteadas como Tipo de problema, prioridad, aplicaciones, módulos, procesos, pantallas, etc.

4.4.1.Métodos de Vista_Incidente

```
package Conexion;
import java.sql.SQLException;


public class Vista_Incidente {
    Scanner teclado = new Scanner(System.in);

    public int Solicita_Tipo_Problema() {
            System.out.println("Ingrese tipo de problema:");
            System.out.println("[1] Sistemas [2] IT:");
            int tipo=Integer.parseInt(teclado.nextLine());
            if (tipo !=1) {
                if (tipo !=2) {
                    System.out.println("Tipo de problema incorrecto.");
                    Solicita_Tipo_Problema();
                 }}
            return tipo;
    }

    public int Solicita_Prioridad() {
        System.out.println("Seleccione Prioridad.");
        System.out.println("[1] Alta  [2] Media  [3] Baja ");
        int prioridad= Integer.parseInt(teclado.nextLine());

        if (prioridad !=1) {
            if (prioridad !=2) {
                if (prioridad !=3) {
                    Solicita_Prioridad();
                }
             }}
        return prioridad;
    }
```

Métodos que permiten visualizar mensajes de confirmación (se muestran algunos en forma de ejemplo):

```
public int Seleccionar_Confirma() {
    System.out.println("Desea confirmar?");
    System.out.println("[1]Si    [2]No ");
    return teclado.nextInt();
}


public int Seleccionar_Confirma_Reg_Incidente() {
    System.out.println("_____________________________________________________________");
    System.out.println("[1] Registrar otro Incidente. [2] Salir" );
    return teclado.nextInt();
}


public int Seleccionar_Cancela_Otro_Incidente() {
    System.out.println("Desea cancelar otro incidente?");
    System.out.println("_____________________________________________________________");
    System.out.println("[1] Cancelar Incidente    [2] Salir." );
    return teclado.nextInt();
}
```

Métodos que permiten seleccionar aplicación, módulo, proceso, pantalla. Todos ellos reciben como parámetro un objeto del tipo **Modelo_Incidente**, para realizar la búsqueda en la base de datos:

```
public int  seleccionar_Apliacion(Modelo_Incidente modelo) throws SQLException
{
    System.out.println("Seleccione aplicación:");
    ArrayList<Integer> Lst_Aplicaciones = modelo.List_Aplicaciones();   //se almacena en un array list la lista
        int aplic=teclado.nextInt();
        int apl_ok = 0;

        for (int x=0;  x < Lst_Aplicaciones.size(); x++) {

            if (Lst_Aplicaciones.get(x) == aplic) {
                apl_ok = 1;
            }
        }
        if (apl_ok == 0) {
            System.out.println("Aplicación seleccionada no existe.");
            seleccionar_Apliacion(modelo);
        }
        return aplic;
}
```

```
public int  seleccionar_Modulo(Modelo_Incidente modelo, int aplic) throws SQLException
{
    System.out.println("Seleccione Modulo:");
    ArrayList<Integer> Lst_Modulos = modelo.List_Modulos(aplic);   //se almacena en un array list la lista
        int modul=teclado.nextInt();
        int mod_ok = 0;

        for (int x=0;  x < Lst_Modulos.size(); x++) {

            if (Lst_Modulos.get(x) == modul) {
                mod_ok = 1;
            }
        }
        if (mod_ok == 0) {
            System.out.println("Moulo seleccionado no existe.");
            seleccionar_Modulo(modelo, aplic);
        }
        return modul;
}
```

```
public int  seleccionar_Proceso(Modelo_Incidente modelo, int modu) throws SQLException
{
    System.out.println("Seleccione Proceso:");
    ArrayList<Integer> Lst_Procesos =  modelo.List_Procesos(modu);   //se almacena en un array list la
        int pro=teclado.nextInt();
        int pro_ok = 0;

        for (int x=0;  x < Lst_Procesos.size(); x++) {

            if (Lst_Procesos.get(x) == pro) {
                pro_ok = 1;
            }
        }
        if (pro_ok == 0) {
            System.out.println("Proceso seleccionado no existe.");
            seleccionar_Proceso(modelo, modu);
        }
        return pro;
}
```

```
public int  seleccionar_Pantalla(Modelo_Incidente modelo, int proc) throws SQLException
{
    System.out.println("Seleccione Pantalla:");
    ArrayList<Integer> Lst_Pantallas = modelo.List_Pantallas(proc);   //se almacena en un array list la l
        int pan=teclado.nextInt();
        int pan_ok = 0;

        for (int x=0;  x < Lst_Pantallas.size(); x++) {

            if (Lst_Pantallas.get(x) == pan) {
                pan_ok = 1;
            }
        }
        if (pan_ok == 0) {
            System.out.println("Pantalla seleccionada no existe.");
            seleccionar_Pantalla(modelo, proc);
        }
        return pan;
}

public int Seleccionar_Nro_Incidente() {
    System.out.println("Seleccione Número de Incidente:");
    return teclado.nextInt();
}
```

4.4.2.Metodo Listar_Inc()

```
public void Listar_Inc (ArrayList<String> Data) {
     System.out.println("_____________________________________________________________________________");
     System.out.println("Incidentes Encontrados");
     System.out.println("_____________________________________________________________________________");
     System.out.println(String.format("%-10s", "Incidente") + " | " +  String.format("%-15s", " Fecha")
     + " | " +  String.format("%-10s", " Prior.")  + " | " + String.format("%-90s", "Descripción")
     + " | " + String.format("%-20s","Estado")  + "| " + String.format("%-20s","Aplicacion")
     + " | " + String.format("%-20s","Modulo") + " | " + String.format("%-20s","Proceso")
     + " | " + String.format("%-20s","Pantalla") + " | " + String.format("%-20s","Usuario")
     + " | " + String.format("%-20s", "Area"));

     for (int x=0;  (x < Data.size()); x++) {
           System.out.println(Data.get(x));
       }
}
```

4.5. Clase Modelo_Incidente

Es parte del patrón **MVC**, se utiliza para manejar la lógica que permite la interacción con la base de datos. El modelo se encarga de representar y gestionar los datos relacionados con la carga y listado de incidentes.

En esta clase se realiza la importación de paquetes que contienen las clases necesarias para establecer conexiones a la base de datos, manejar errores y gestionar conjuntos de resultados.

Esta clase tiene importada la clase **java.sql.SQLException**, que permite obtener información de errores relacionados al acceso a la base de datos y manipulación de sentencias SQL a través del método**: get_message().**

```
package Conexion;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import Incidentes.Errores;

public class Modelo_Incidente {
    private Conectar conexion;

    public Modelo_Incidente(Conectar conexion) {
        this.conexion= conexion;
    }
```

Esta clase contiene atributo privado y el constructor.

conexión: permite mantener una única instancia de conexión a la base de datos, que se utilizará en los métodos para interactuar con la base de datos.

4.5.1.Metodo List_Aplicaciones()

Obtiene listado de aplicaciones de la base de datos

```
public  ArrayList<Integer> List_Aplicaciones () throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    ArrayList<Integer> aplic = new ArrayList<Integer>();

    try {
        ps = conexion.getConnection().prepareStatement("select *  from aplicaciones ");
        rs=ps.executeQuery();

        while (rs.next()) {
            System.out.println("[" + rs.getInt("Id_Aplicacion") + "]" + " " + rs.getString("Desc_Aplicacion"));
            aplic.add(rs.getInt("Id_Aplicacion"));
        }

    } catch (Exception ext) {System.out.println("Error List_Aplicaciones:" + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return aplic;
}
```

4.5.2.List_Modulos(Id_Aplicacion)

Obtiene listado de módulos de la base de datos

```
public  ArrayList<Integer> List_Modulos (int aplic) throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    ArrayList<Integer> modul = new ArrayList<Integer>();
    try {
        ps = conexion.getConnection().prepareStatement("select *  from modulos where Id_Aplicacion = " + aplic);
        rs=ps.executeQuery();

        while (rs.next()) {
            System.out.println("[" + rs.getInt("Id_Modulo") + "]" + " " + rs.getString("Desc_Modulo"));
            modul.add(rs.getInt("Id_Modulo"));
        }

    } catch (Exception ext) {System.out.println("Error List_Modulos:" + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return modul;
}
```

4.5.3. List_Procesos(Id_Modulo)

Obtiene listado de procesos de la base de datos.

```
public  ArrayList<Integer> List_Procesos (int mod) throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    ArrayList<Integer> proc = new ArrayList<Integer>();

    try {
        ps = conexion.getConnection().prepareStatement("select *  from procesos where Id_modulo = " + mod);
        rs=ps.executeQuery();

        while (rs.next()) {
            System.out.println("[" + rs.getInt("Id_Proceso") + "]" + " " + rs.getString("Desc_Proceso"));
            proc.add(rs.getInt("Id_Proceso"));
        }

    } catch (Exception ext) {System.out.println("Error List_Procesos:" + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return proc;
}
```

4.5.4.List_Pantallas(Id_Proceso)

Obtiene listado de pantallas de la base de datos

```
public  ArrayList<Integer> List_Pantallas (int pro) throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    ArrayList<Integer> pan = new ArrayList<Integer>();

    try {
        ps = conexion.getConnection().prepareStatement("select *  from pantallas where Id_proceso = " + pro);
        rs=ps.executeQuery();

        while (rs.next()) {
            System.out.println("[" + rs.getInt("Id_Pantalla") + "]" + " " + rs.getString("Desc_Pantalla"));
            pan.add(rs.getInt("Id_Pantalla"));
        }

    } catch (Exception ext) {System.out.println("Error List_Pantalla:" + ext.getMessage());}
 finally {
    try {
        ps.close();
        rs.close();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
    return pan;
}
```

4.5.5.Insertar_Incidente(Errores)

Recibe el objeto Errores y realiza inserción en las tablas Incidentes y Errores.

```
public void Insertar_Incidente(Errores Inc) throws SQLException {
    java.sql.PreparedStatement ps= null;
    try {
        ps= conexion.getConnection().prepareStatement("INSERT INTO Errores (Id_Error, Id_Pantalla, Codigo, Descripcion, "
                + "Pasos_Reprod, Id_Tipo_Prob) VALUES (?, ?, ?, ?, ?, ?)");
        ps.setInt(1, Inc.getSec_Err());
        ps.setInt(2, Inc.getPantalla());
        ps.setString(3,Inc.getNro_Error());
        ps.setString(4, Inc.getDescripcion_Corta());
        ps.setString(5,Inc.getPasos());
        ps.setInt(6, Inc.getTipo_Problema());
        ps.executeUpdate();
        ps.close();

        ps = conexion.getConnection().prepareStatement("INSERT INTO Incidentes (Id_Incidente, Id_Usuario, Fecha, Anexo, "
                + "Id_Error, Prioridad, Id_Estado) VALUES (?, ?, ?, ?, ?, ?, ?)");
        ps.setInt(1, Inc.getNro_Incidente());
        ps.setInt(2, Inc.getUsuario());
        ps.setDate(3, (java.sql.Date) Inc.getFecha());
        ps.setString(4,null);
        ps.setInt(5, Inc.getSec_Err());
        ps.setInt(6, Inc.getPrioridad());
        ps.setInt(7, Inc.getEstado());
        ps.executeUpdate();

        System.out.println("Incidente registrado con Exito en la base de datos. NRO : " + Inc.getNro_Incidente() + ".");

    } catch (Exception ext) {System.out.println("Error al Crear_Incidente:" + ext.getMessage());}
 finally {
    try {
        ps.close();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
}
```

4.5.6. Buscar_Inc_Usuario_Abiertos()

Este método permite obtener los incidentes en estado Abierto del área a la que pertenece el usuario logueado, muestra información de los incidentes por consola.
En caso de ser resolutor visualiza todos los incidentes.
Busca información de los incidentes abiertos, en la base de datos para que el usuario pueda Cancelarlos. Retorna un ArrayList con los Id_Incidentes obteidos. Este arrayList es usado por el método **Cancelar_Incidente()** de la clase **Controlador_Incidente.**

```
public ArrayList<Integer>  Buscar_Inc_Usuario_Abiertos (int Id_usu, int Id_area, int tipo_usu) throws SQLException {
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;

    ArrayList<Integer> Incidentes_Usuario = new ArrayList<Integer>();
    ArrayList<String> Listado = new ArrayList<String>();

    String condicion = "";
    if (tipo_usu == 2) {
        condicion= " ar.Id_Area= ar.Id_Area ";
    } else {
        condicion= " ar.Id_Area= " + Id_area;
    }
    try {
        ps = conexion.getConnection().prepareStatement("select i.Id_Incidente, i.Fecha , i.prioridad, err.Descripcion, "
                + " Est.Desc_Estado, ap.desc_aplicacion , mo.Desc_Modulo , pro.Desc_Proceso , pan.Desc_Pantalla,  "
                + " usu.Usuario , ar.Desc_Area "
                + " from   inc.incidentes i "
                + " inner join inc.errores err on i.Id_Error= err.Id_Error    "
                + " inner join inc.estados est on i.Id_Estado = est.Id_Estado "
                + " inner join inc.pantallas pan on err.Id_Pantalla = pan.Id_Pantalla  "
                + " inner join inc.procesos pro on pan.Id_Proceso = pro.Id_Proceso "
                + " inner join inc.modulos mo on pro.Id_Modulo = mo.Id_Modulo "
                + " inner join inc.aplicaciones ap on mo.Id_Aplicacion = ap.Id_Aplicacion  "
                + " left join inc.resolutores res on res.Id_Resolutor = i.Id_Resolutor "
                + " inner join inc.usuarios usu on i.Id_Usuario = usu.Id_Usuario  "
                + " inner join inc.empleados em on usu.legajo = em.legajo  "
                + " inner join inc.areas ar on em.Id_Area = ar.Id_Area  "
                + " where i.id_estado not in (2,4) and " + condicion);
        rs=ps.executeQuery();
```

```
            while (rs.next()) {
                Incidentes_Usuario.add(rs.getInt("id_Incidente"));
                Listado.add(String.format("%-10s",  rs.getInt("Id_Incidente"))  + " | " +
                String.format("%-15s", rs.getDate("Fecha")) + " | " +
                String.format("%-10s", rs.getInt("Prioridad")) + " | " +
                String.format("%-90s", rs.getString("Descripcion")).substring(0, 90) + " | " +
                String.format("%-20s", rs.getString("Desc_Estado")) + " | " +
                String.format("%-20s", rs.getString("Desc_Aplicacion")) + " | " +
                String.format("%-20s", rs.getString("Desc_Modulo")) + " | " +
                String.format("%-20s", rs.getString("Desc_Proceso")) + " | " +
                String.format("%-20s", rs.getString("Desc_Pantalla")) + " | " +
                String.format("%-20s", rs.getString("Usuario")) + " | " +
                String.format("%-20s", rs.getString("Desc_Area")));
            }
            Vista_Registro vista = new Vista_Registro();
            vista.Listar_Inc (Listado) ;

        } catch (Exception ext) {System.out.println("Error Buscar_Inc_Usuario_Abiertos:" + ext.getMessage());}
     finally {
        try {
            ps.close();
            rs.close();
            }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
        }
        return Incidentes_Usuario;
    }
```

4.5.7.Cancela_Incidente()

Cambia de estado el incidente a “cancelado”.

```
    public void Cancela_Incidente(int num) throws SQLException {
        java.sql.PreparedStatement ps= null;
        try {
        ps = conexion.getConnection().prepareStatement("update incidentes set id_estado = 2 where id_incidente = "+ num);
        ps.executeUpdate();
        ps.close();
    } catch (Exception ext) {System.out.println("Error al Crear_Incidente:" + ext.getMessage());}
     finally {
        try {
            }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
        }
    }
```

4.5.8.Listar_Incidentes()

Lista los incidentes según sea el área del usuario logueado. En caso de ser resolutor lista todos los incidentes.

```
public ArrayList<String> Listar_Incidentes(int id_usu, int id_area, int tipo_usu) throws SQLException {   /*PASAR A WORD V
    java.sql.PreparedStatement ps= null;
    ResultSet rs = null;
    ArrayList<String> Listado = new ArrayList<String>();
    String condicion = "";
    if (tipo_usu == 2) {
        condicion= " ar.Id_Area= ar.Id_Area ";
    } else {
        condicion= " ar.Id_Area= " + id_area;
    }

     try {

        ps = conexion.getConnection().prepareStatement("select i.Id_Incidente, i.Fecha , i.prioridad, err.Descripcion, "
                + " Est.Desc_Estado, ap.desc_aplicacion , mo.Desc_Modulo , pro.Desc_Proceso , pan.Desc_Pantalla,  "
                + " usu.Usuario , ar.Desc_Area "
                + " from   inc.incidentes i "
                + " inner join inc.errores err on i.Id_Error= err.Id_Error    "
                + " inner join inc.estados est on i.Id_Estado = est.Id_Estado "
                + " inner join inc.pantallas pan on err.Id_Pantalla = pan.Id_Pantalla  "
                + " inner join inc.procesos pro on pan.Id_Proceso = pro.Id_Proceso "
                + " inner join inc.modulos mo on pro.Id_Modulo = mo.Id_Modulo "
                + " inner join inc.aplicaciones ap on mo.Id_Aplicacion = ap.Id_Aplicacion  "
                + " left join inc.resolutores res on res.Id_Resolutor = i.Id_Resolutor "
                + " inner join inc.usuarios usu on i.Id_Usuario = usu.Id_Usuario  "
                + " inner join inc.empleados em on usu.legajo = em.legajo  "
                + " inner join inc.areas ar on em.Id_Area = ar.Id_Area  "
                + " where  " + condicion
                + " and i.Id_Estado  <> 2 "
                + " order by 1" );
        rs=ps.executeQuery();
```

```
        while (rs.next()) {
            Listado.add(String.format("%-10s",  rs.getInt("Id_Incidente"))  + " | " +
                    String.format("%-15s", rs.getDate("Fecha")) + " | " +
                    String.format("%-10s", rs.getInt("Prioridad")) + " | " +
                    String.format("%-90s", rs.getString("Descripcion")).substring(0, 90) + " | " +
                    String.format("%-20s", rs.getString("Desc_Estado")) + " | " +
                    String.format("%-20s", rs.getString("Desc_Aplicacion")) + " | " +
                    String.format("%-20s", rs.getString("Desc_Modulo")) + " | " +
                    String.format("%-20s", rs.getString("Desc_Proceso")) + " | " +
                    String.format("%-20s", rs.getString("Desc_Pantalla")) + " | " +
                    String.format("%-20s", rs.getString("Usuario")) + " | " +
                    String.format("%-20s", rs.getString("Desc_Area")));
        }

    } catch (Exception ext) {System.out.println("Error Listar_Incidentes:" + ext.getMessage());}
finally {
    try {
        ps.close();
        rs.close();
        }catch (Exception ext) {System.out.println("Error :" + ext.getMessage());}
    }
     return Listado;
}
```

Luego de realizadas algunas pruebas de inserción, se puede verificar la persistencia de la información consultando en la base de datos: