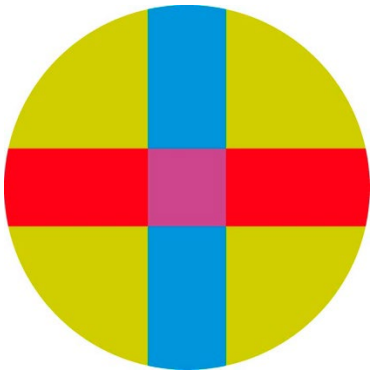
I

UNIVERSIDAD SAN PABLO - CEU

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN



TRABAJO FIN DE GRADO

APLICACIÓN PARA LA GESTIÓN DE SINIESTROS DE VEHÍCULOS

VEHICLE CLAIMS MANAGEMENT APP

Autor: Gonzalo Daniel Paúl Fantoni
Tutor: Rafael Núñez Hervás

Mayo 2024

UNIVERSIDAD SAN PABLO-CEU

ESCUELA POLITÉCNICA SUPERIOR

División de Ingeniería

Calificación del Trabajo Fin de Grado

Datos del alumno

NOMBRE:

Datos del Trabajo

TÍTULO DEL PROYECTO:

Tribunal calificador

PRESIDENTE:	FDO.:
SECRETARIO:	FDO.:
VOCAL:	FDO.:

Reunido este tribunal el ___ /___ /_____, acuerda otorgar al Trabajo Fin de Grado presentado por D./Dña. _____ la calificación de _____

Resumen

Este proyecto consiste en una aplicación web para la gestión de siniestros de vehículos. Los usuarios pueden registrarse y, una vez validados por un administrador, reportar incidencias detalladas mediante un modelo 3D interactivo. La aplicación distingue entre roles de usuario (asegurado), trabajador de la aseguradora y administrador, ofreciendo funcionalidades específicas para cada uno. Los trabajadores pueden revisar, analizar y gestionar incidencias con el apoyo de Drools, un motor de reglas que automatiza el análisis y la detección de fraudes. Los administradores, además, pueden asignar roles y permisos a los usuarios. El despliegue se realiza mediante Docker, facilitando la instalación y ejecución de la aplicación. La integración con Drools se maneja a través de una API, permitiendo un análisis eficiente de los siniestros reportados.

Palabras Clave

- **Django**: Un framework de desarrollo web en Python que facilita la creación de aplicaciones web robustas y escalables.
- **Vehicle Incident Management**: Gestión de siniestros de vehículos, incluyendo la creación, revisión y resolución de incidencias.
- **Drools**: Un motor de reglas de negocio que permite automatizar decisiones basadas en reglas predefinidas.
- **Rules Engine**: Sistema que aplica reglas lógicas para realizar decisiones automáticas.
- **API Integration**: Proceso de conectar diferentes sistemas o aplicaciones mediante APIs para permitir la comunicación y el intercambio de datos.
- **Docker Deployment**: Uso de Docker para desplegar y ejecutar aplicaciones en contenedores, asegurando un entorno consistente.
- **User Roles**: Diferentes permisos y accesos asignados a usuarios basados en su rol (asegurado, trabajador, administrador).
- **Fraud Detection**: Proceso de identificar y prevenir actividades fraudulentas en la gestión de siniestros.
- **Interactive 3D Model**: Modelo 3D interactivo utilizado para reportar y visualizar daños en un vehículo.
- **Incident Reporting**: Proceso de documentar y reportar siniestros de vehículos a través de la aplicación.
- **Insurance Claims**: Reclamaciones de seguros presentadas por los asegurados para compensar los daños sufridos en un siniestro.

Abstract

This project involves the development of a web application using Django to manage vehicle accidents. Users can register and, once validated by an administrator, report detailed incidents through an interactive 3D model. The application differentiates between user roles (insured), insurance company worker, and administrator, providing specific functionalities for each. Workers can review, analyze, and manage incidents with the support of Drools, a rules engine that automates the analysis and fraud detection process. Additionally, administrators can assign roles and permissions to users. The deployment is handled via Docker, simplifying the installation and execution of the application. The integration with Drools is managed through a custom API, enabling efficient analysis of reported incidents.

Keywords

- **Django**: A web development framework in Python that facilitates the creation of robust and scalable web applications.
- **Vehicle Incident Management**: The process of managing vehicle incidents, including the creation, review, and resolution of claims.
- **Drools:** A business rules engine that automates decision-making based on predefined rules.
- **Rules Engine**: A system that applies logical rules to make automated decisions.
- **API Integration**: The process of connecting different systems or applications via APIs to enable communication and data exchange.
- **Docker Deployment**: Using Docker to deploy and run applications in containers, ensuring a consistent environment.
- **User Roles**: Different permissions and access levels assigned to users based on their roles (insured, worker, administrator).
- **Fraud Detection**: The process of identifying and preventing fraudulent activities in incident management.
- **Interactive 3D Model**: An interactive 3D model used for reporting and visualizing vehicle damage.
- **Incident Reporting**: The process of documenting and reporting vehicle incidents through the application.
- **Insurance Claims**: Claims submitted by the insured to compensate for damages sustained in an incident.
- **Automated Analysis**: The automatic analysis of data and incidents using technologies like Drools to improve efficiency and accuracy.

Índice de contenidos

Índice de figuras

Capítulo 1

Introducción

1.1 Contexto del TFG

El presente documento tiene como propósito la descripción del trabajo llevado a cabo durante el desarrollo del Trabajo Fin de Grado del grado en Ingeniería en Sistemas de Información.

Dicho TFG ha consistido en el desarrollo de una aplicación web que permita a aseguradoras gestionar siniestros de vehículos. Para la elaboración de esta aplicación se han llevado a cabo las siguientes tareas:

- Desarrollo de una aplicación web.
- Creación de un modelo de decisión a partir de reglas de negocio.
- Desarrollo de un gestor de datos de usuario.

1.2 Objetivos

Los siniestros y reclamaciones son una preocupación constante en el sector asegurador, impactando significativamente tanto en los recursos económicos como humanos de las compañías. Según estadísticas recientes, el costo de gestión de siniestros en el sector asegurador ascendió a aproximadamente 35 mil millones de euros en 2023, representando un desafío creciente para la eficiencia operativa de las aseguradoras.

En este contexto, la tecnología juega un papel crucial. La digitalización y automatización de los procesos de reclamación no solo prometen reducir costos, sino también mejorar la experiencia del usuario, acelerando los tiempos de respuesta y aumentando la transparencia en el manejo de cada caso.

El principal objetivo de este proyecto es desarrollar una aplicación web que pueda conectarse a un motor de decisión, y permita ejecutar reglas de negocio de forma dinámica. Esta solución busca optimizar el proceso de alta de siniestros, implementando lógica compleja de manera eficiente y flexible para adaptarse a los distintos escenarios que puedan presentarse en la gestión de reclamaciones.

Actualmente, las soluciones basadas en reglas de negocio como Drools han ganado popularidad en la industria, no solo por su capacidad de manejar lógica compleja, sino también por su integración fácil con otras plataformas de desarrollo. Esta integración es esencial para crear aplicaciones robustas y escalables que puedan responder efectivamente a las necesidades del mercado asegurador.

Ilustración 1: 'El fraude al seguro español. Datos 2021'

Esta aplicación no solo está diseñada para mejorar la eficiencia operativa, sino también para proporcionar herramientas que permitan a los ajustadores y gestores de reclamos realizar evaluaciones más precisas y fundamentadas. Esto es posible gracias a la implementación de reglas dinámicas que pueden ajustarse en tiempo real según las necesidades del negocio y los cambios en la normativa vigente.

Capítulo 2

Gestión del proyecto

2.1 Modelo de ciclo de vida

El proyecto se articuló a través de la integración de dos componentes esenciales con características y requerimientos distintos.

Inicialmente, se abordó la construcción de una aplicación web. Este componente se trató como un proyecto clásico de software, donde la atención se centró en lograr una implementación sólida, segura y escalable.

Paralelamente, se desarrolló un sistema de reglas de negocio utilizando el motor Drools. Este elemento del proyecto, por su complejidad y especificidad técnica, demandó un enfoque diferente. Se necesitaba una solución que permitiera la adaptabilidad y la modificación rápida de reglas para responder de manera efectiva a los dinámicos requerimientos del sector asegurador.

Dado que estos dos elementos son fundamentales pero distintos en su naturaleza, es crucial diferenciarlos al discutir el ciclo de vida del proyecto. Aunque interdependientes, cada uno requería una metodología y un plan de desarrollo específicos para optimizar su funcionamiento y eficacia.

Selección de Metodología de Desarrollo

Para el desarrollo de la aplicación web, se seleccionó una metodología ágil, optando específicamente por el marco de trabajo Scrum. Esta metodología fue complementada por prácticas de DevOps que facilitan la integración y entrega continuas, esenciales para mantener el ritmo de las actualizaciones y la adaptabilidad del sistema.

La decisión de adoptar un enfoque ágil se basó en varios criterios clave:

- Personal: Como desarrollador único, era esencial elegir un enfoque que permitiera gestionar eficientemente todas las facetas del proyecto, desde el diseño hasta la implementación y mantenimiento.
- Cultura de trabajo: Trabajando solo, era necesario un método que se adaptaba bien a la autogestión y a un ritmo de trabajo flexible.
- Tamaño del proyecto: El tamaño reducido del equipo (un solo miembro) hacía más viable una metodología ágil que favorece equipos pequeños y menos burocráticos.
- Criticidad: Si bien la aplicación maneja procesos críticos, las estrategias de contingencia y los sistemas de respaldo adecuados ayudan a mitigar los riesgos asociados.

Este enfoque permitió la implementación rápida y efectiva del sistema, con la flexibilidad necesaria para ajustar o rediseñar elementos conforme evolucionaban las necesidades del proyecto. La metodología ágil facilitó no solo el desarrollo iterativo, sino también la capacidad de reaccionar proactivamente a los desafíos técnicos y operativos que surgieron durante el proyecto.

2.2 Planificación

La planificación de este proyecto se ha realizado de manera detallada para asegurar una ejecución eficiente y ordenada de todas las actividades. Se ha utilizado un enfoque estructurado para identificar las etapas, tareas específicas, recursos necesarios, y las dependencias entre tareas. A continuación, se describen las fases planificadas:

1. **Fase de Investigación y Adquisición de Conocimientos (Inicio: 1 de febrero - Fin: 28 de febrero)**
   - Objetivo: Adquirir conocimientos profundos sobre web frameworks y Drools, y comprender el entorno regulatorio y operativo de la gestión de siniestros.
   - Actividades Principales:

  - Estudio de la documentación técnica de Django (framework elegido) y Drools.
  - Revisión de las normativas actuales en gestión de siniestros y procesos de reclamación.
- Recursos: Acceso a documentación técnica, cursos en línea, y asesoría del tutor.

2. **Desarrollo del Marco de Trabajo (Inicio: 1 de marzo - Fin: 15 de abril)**
   - Objetivo: Establecer la arquitectura base de la aplicación y desarrollar las funcionalidades principales.
   - Actividades Principales:
     - Configuración del entorno de desarrollo.
     - Implementación de las estructuras de datos y modelos en Django.
     - Desarrollo de la API para integración con Drools.
   - Recursos: Software de desarrollo, servidor de pruebas.
3. **Implementación del Motor de Reglas Drools (Inicio: 15 de abril - Fin: 31 de abril)**
   - Objetivo: Desarrollar e integrar el motor de reglas que manejará la lógica de negocio específica para la gestión de siniestros.
   - Actividades Principales:
     - Diseño de reglas de negocio.
     - Integración y pruebas de las reglas con la aplicación.
   - Recursos: Drools, documentación específica de reglas de negocio.
4. **Pruebas y Validación (Inicio: 1 de mayo - Fin: 4 de junio)**
   - Objetivo: Asegurar que la aplicación cumple con todos los requisitos funcionales y no funcionales.
   - Actividades Principales:
     - Pruebas unitarias y de integración.
     - Pruebas de rendimiento y seguridad.
   - Recursos: Herramientas de testing, casos de prueba.

5. **Documentación y Entrega Final (Inicio: 1 de febrero - Fin: 4 de junio)**
   - Objetivo: Documentar exhaustivamente el proyecto y preparar la entrega final.
   - Actividades Principales:
     - Elaboración de la documentación técnica y del usuario.
     - Preparación de la presentación final y entrega del TFG.
   - Recursos: Herramientas de documentación, feedback del tutor.

Herramientas de Gestión de Proyecto Utilizadas:

Diagrama de Gantt: Para ilustrar la línea de tiempo del proyecto y la interdependencia entre tareas.

Diagrama PERT/CPM: Para identificar el camino crítico y las tareas que determinan la duración del proyecto.

Ilustración 2: Diagrama de Gantt para la ilustración de la línea temporal del proyecto

2.3 Presupuesto

Para la ejecución de este proyecto no se han necesitado recursos económicos adicionales, ya que se ha utilizado software gratuito en todas las etapas del desarrollo. Los únicos gastos implicados han sido el consumo de electricidad de los equipos informáticos, especialmente relevante durante las fases de mayor carga computacional, y el costo en términos de las horas invertidas en programación, pruebas y coordinación del proyecto.

2.4 Ejecución

A lo largo del desarrollo del proyecto, se observaron varias divergencias respecto a la planificación inicial, algo bastante común en proyectos de desarrollo de software, especialmente cuando se manejan de manera individual.

Fase de Investigación y Adquisición de Conocimientos: Esta fase se inició y concluyó según lo planificado. La recolección de información y el estudio de las tecnologías a utilizar (Django y Drools) se llevaron a cabo dentro del periodo establecido, finalizando a finales de febrero.

Desarrollo del Marco de Trabajo: Contrario a lo esperado, esta etapa experimentó un retraso significativo. El inicio de la fase de desarrollo se postergó debido a complicaciones inesperadas en la configuración del entorno de desarrollo y algunos desafíos técnicos no anticipados. Como resultado, la duración de esta etapa se extendió, consumiendo más tiempo del estimado inicialmente y afectando las fases subsiguientes del proyecto.

Implementación del Motor de Reglas Drools: Aunque se planeó iniciar esta fase a mediados de abril, realmente comenzó a principios de mayo debido a los retrasos acumulados anteriormente. Sin embargo, la implementación del motor de reglas fue más rápida de lo esperado, compensando parcialmente los retrasos anteriores. Esto se debió a una mejor familiaridad con la herramienta y a una integración más fluida de lo previsto.

Pruebas y Validación: Esta etapa también se vio afectada por los retrasos en el desarrollo, comenzando más tarde de lo planeado. Sin embargo, las pruebas revelaron menos problemas de los esperados, lo cual permitió recuperar algo de tiempo.

Documentación y Entrega Final: La documentación del proyecto, que se había previsto llevar a cabo de manera paralela durante todo el desarrollo, no se intensificó hasta finales de abril. Esto llevó a un mes de mayo muy cargado, donde se tuvo que dedicar

un gran esfuerzo para compilar toda la información, redactar adecuadamente los documentos finales y preparar la presentación del proyecto.

Reflexión sobre la Planificación y Ejecución:

Utilizando los mismos diagramas de Gantt y PERT/CPM empleados en la planificación, se pudo visualizar y ajustar el cronograma en tiempo real, lo que facilitó la gestión de los retrasos y la reasignación de recursos cuando fue necesario. Aunque el proyecto no se desarrolló exactamente como se había planificado, los ajustes oportunos permitieron cumplir con los objetivos principales del proyecto dentro de un marco temporal razonable.

Capítulo 3

Análisis

3.1 Especificación de requisitos

Para la organización del desarrollo de este proyecto se ha optado por una metodología ágil, por lo tanto, basada en historias de usuario.

Funcionalidades del sistema

- Los usuarios podrán registrarse y loguearse en la aplicación web, especificando:
  - Nombre y Apellidos
  - Email
  - Contraseña
- Los usuarios una vez registrados tendrán que esperar a la validación de un administrador para poder utilizar la aplicación.
- Los usuarios podrán consultar sus datos personales configurados en el perfil.
- Los usuarios podrán modificar dichos datos, a través de una interfaz personalizada.
- Los usuarios validados (asegurados a partir de ahora), podrán abrir incidencias (siniestros) a partir de un modelo 3d de un vehículo genérico.
- Los asegurados deberán añadir diferentes campos a la hora de crear la incidencia.
- Los asegurados podrán consultar sus incidencias, así como el estado de estas.
- Los administradores de la aplicación tienen una visión global de ella, pudiendo acceder a todos los recursos.
- Los administradores son los únicos capaces de añadir y cambiar el rol a los usuarios.

- Dentro de la aplicación habrá tres roles:
    - Asegurados (usuarios validados).
    - Trabajadores de la aseguradora (usuarios con rol de trabajador).
    - Administradores del sistema (super usuarios).
- Los trabajadores tendrán acceso a un panel de control (dashboard), desde donde podrán observar y filtrar las incidencias creadas por los asegurados.
- Los trabajadores podrán importar datos sobre los asegurados, estos datos serán los siguientes:
    - Historial de incidencias
    - Historial de fraudes
    - Póliza
- Dentro de ese panel de control, los trabajadores podrán acceder a incidencias concretas, y enviar dicha información a Drools.
- Drools mandará de vuelta a los trabajadores las respuestas y recomendaciones sobre dicha incidencia.
- El sistema almacenará las respuestas para su posterior tratamiento.
- Una vez realizado el análisis de la incidencia, el trabajador podrá cambiar el estado de esta.
- Los asegurados podrán descargar en formato PDF su incidencia una vez tramitada.
- El sistema será capaz de almacenar de forma segura todos los datos introducidos y generados.

Interfaz de usuario

Se opta por una multi-interfaz con diseños específicos para cada tipo de usuario. Con esto se consigue facilitar la navegación y la usabilidad según las funciones asignadas a cada rol.

Respecto al cumplimiento de la norma de accesibilidad WCAG, se consigue un nivel A, gracias a las siguientes características:

- Navegación por Tabulación: Esencial para cumplir con el criterio *2.1.1 (Teclado) de WCAG*, que requiere que todas las funcionalidades sean operables a través de una interfaz de teclado.
- Atributos Alt en Imágenes: Cumple con el criterio *1.1.1 (Contenido no textual) de WCAG*, que asegura que todo el contenido no textual tenga un texto alternativo que sirva el mismo propósito.

La aplicación cuenta con una interfaz 3D, difícil de hacer accesible. Aunque no indispensable para la creación de incidencias.

Rendimiento y Capacidad

Para evaluar el rendimiento y la capacidad de la aplicación web desarrollada, se ha utilizado Apache JMeter, una herramienta de software libre diseñada para realizar pruebas de carga y rendimiento en aplicaciones web. JMeter permite simular múltiples usuarios concurrentes realizando diversas acciones en la aplicación, lo que ayuda a identificar posibles cuellos de botella y evaluar la capacidad del sistema para manejar cargas elevadas.

En este proyecto, JMeter se utilizó para simular 10, 20, y 30 solicitudes por segundo en el endpoint de inicio de sesión. Esta prueba de carga nos permitió observar cómo la aplicación maneja múltiples usuarios intentando iniciar sesión simultáneamente, medir los tiempos de respuesta y detectar posibles errores bajo condiciones de alta carga. La configuración incluyó el uso de grupos de hilos, peticiones HTTP configuradas, y diversos listeners para monitorear y analizar los resultados de las pruebas. El análisis de rendimiento se encuentra en el Anexo III – Capacidad y rendimiento.

Seguridad

El sistema deberá de estar protegido frente a inyecciones SQL, y restricciones de comunicación de POSTs desde Django. En relación con la ISO 27000:

- **ISO/IEC 27001**: Es el estándar central para la gestión de la seguridad de la información. Requiere la implementación de un Sistema de Gestión de Seguridad de

la Información (SGSI), que incluye la evaluación de riesgos y la implementación de medidas de seguridad adecuadas basadas en esos riesgos. Las medidas que se van a implementar, como la protección contra inyecciones SQL y la restricción de comunicación de la API, son parte de las buenas prácticas que ayudan a mitigar riesgos específicos y podrían integrarse en un SGSI conforme a ISO/IEC 27001.

- **ISO/IEC 27002**: Proporciona directrices y mejores prácticas para controles de seguridad. Las configuraciones que se han aplicado pueden alinearse con las recomendaciones sobre control de acceso, desarrollo de sistemas seguros y gestión de incidentes.

Interoperabilidad

A continuación, se definen una serie de requisitos de interoperabilidad entre Django y Drools:

1. *Interfaz de Comunicación API*
    a. Requisito Funcional: Implementar una API RESTful en la aplicación Django para facilitar la comunicación bidireccional con Drools.
    b. Requisito Técnico: Asegurar que los datos enviados y recibidos entre Django y Drools sean en formato JSON. Esto incluye que las estructuras de datos de Django a Drools y viceversa coincidan con las definiciones de las clases de objetos Java y los modelos en Django, para un mapeo directo y preciso.
2. *Seguridad de la Comunicación*
    a. Requisito Funcional: Configurar la API para permitir únicamente peticiones POST desde la dirección IP donde se ejecuta Django, y configurar Django para recibir respuestas POST desde Drools.
    b. Requisito Técnico: Todas las comunicaciones entre Django y Drools deben ser cifradas utilizando HTTPS, asegurando la confidencialidad e integridad de los datos intercambiados.

3. *Procesamiento y Respuesta de Datos*
    a. Requisito Funcional: Django debe enviar solicitudes de procesamiento de incidencias a Drools en tiempo real, y recibir respuestas de Drools, que contienen decisiones basadas en reglas de negocio, en formato JSON.
    b. Requisito Técnico: Implementar y configurar adecuadamente los serializadores/deserializadores en Django para gestionar eficientemente la conversión de datos JSON de entrada y salida, garantizando la correcta integración de las respuestas en la interfaz de usuario y los flujos de trabajo.
4. *Mantenimiento de la Interoperabilidad*
    a. Requisito Funcional: Establecer un protocolo de mantenimiento y actualización que permita cambios en las reglas de negocio en Drools o actualizaciones en Django sin comprometer la funcionalidad de interoperabilidad.
    b. Requisito Técnico: Incorporar pruebas de integración automáticas que verifiquen la interoperabilidad y funcionamiento correcto de la API tras cada actualización o modificación del sistema.

Protección de datos

A continuación, los principios del RGPD, que deberá cumplir el sistema:

a. *Legalidad, Lealtad y Transparencia*: Asegurar que los datos personales se procesen de manera legal, justa y transparente.
b. *Limitación de la Finalidad*: Los datos recogidos deben ser usados solo para fines explícitamente definidos y legítimos.
c. *Minimización de Datos*: Recoger solo los datos estrictamente necesarios para los fines para los cuales son procesados.
d. *Exactitud*: Mantener los datos personales exactos y actualizados.
e. *Limitación del Plazo de Conservación*: No retener datos personales más tiempo del necesario.
f. *Integridad y Confidencialidad*: Implementar medidas técnicas y organizativas adecuadas para garantizar la seguridad de los datos personales.

Respecto a los derechos de los usuarios:

a. *Acceso*: Los usuarios deben poder acceder a sus datos personales y obtener copias de estos.
b. *Rectificación*: Los usuarios deben poder corregir datos incorrectos.
c. *Supresión* ("derecho al olvido"): Los usuarios pueden solicitar la eliminación de sus datos personales.
d. *Limitación del tratamiento*: Los usuarios pueden solicitar la limitación del procesamiento de sus datos.
e. *Portabilidad*: Los usuarios deben poder recibir sus datos en un formato estructurado y de uso común.
f. *Oposición*: Derecho a oponerse al procesamiento de sus datos personales.

Requisitos sobre entorno tecnológico y de comunicaciones

Gestor de base de datos:

SQLite3. Utilizado a través de Django, adecuado para entornos de desarrollo y producción con requerimientos moderados de carga.

Sistemas operativos:

La aplicación debe ser compatible con sistemas operativos que puedan ejecutar Python y Django, como Windows, Linux y macOS.

Hardware:

Requerimientos mínimos de servidores incluyen 16 GB de RAM y procesadores de 4 núcleos para manejar eficientemente la carga de usuarios y el procesamiento de datos.

3.2 Análisis de los Casos de Uso

En una etapa prematura del proyecto se diferenciaron tres grandes componentes. La aplicación, la base de datos, y el motor de reglas de negocio. Gracias al *framework* de

Django, se consigue gestionar las dos primeras de forma eficiente. De acuerdo con el manual de Django, este, sigue el patrón MVC (Model View Controller), aunque con unos pequeños ajustes. En la interpretación de Django de MVC, la vista describe los datos que se presentan al usuario. No es necesariamente cómo se ven los datos, pero sí qué datos son presentados. Por el contrario, Ruby on Rails y frameworks similares (puros MVC) sugieren que el trabajo del controlador incluye qué datos presentan al usuario, mientras que la vista es estrictamente cómo los datos se ven, no qué datos son presentados.

Voy a hacer referencia a la aplicación como la unión de todas las clases necesarias para que el programa funcione. Estas clases son las encargadas de llevar la lógica de negocio y la presentación de los datos. Por lo que podríamos partir del siguiente diagrama para observar la relación entre objetos de la aplicación.

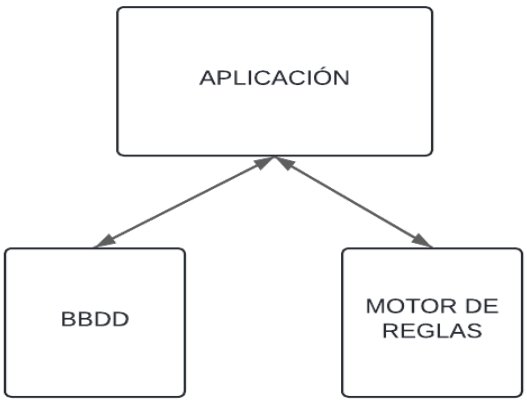


Ilustración 3: Diagrama de clases del software en su totalidad

El motor de reglas actualizará la base de datos a través de la aplicación. Para profundizar en los casos de uso, con los requisitos mencionados previamente se pueden extraer estos diagramas.

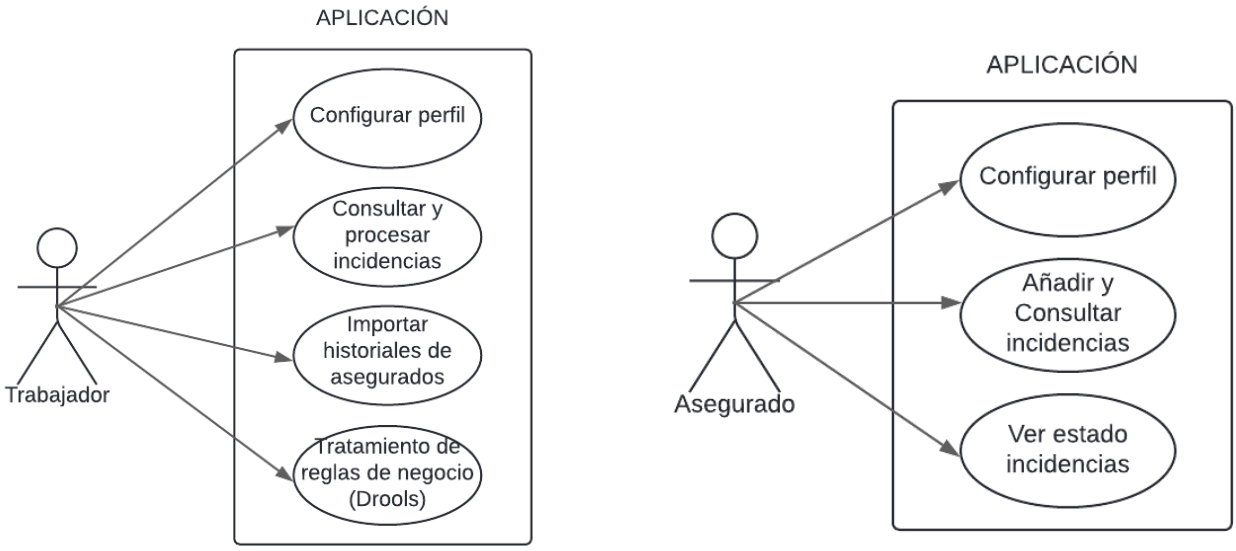


Ilustración 4: Diagramas de caso de uso de los objetos Trabajador y Asegurado

Se diferencia entre asegurado (usuario de la aplicación) y trabajador (trabajador de la aseguradora), y las diferentes acciones que pueden tomar. Se obvia el comportamiento de los administradores, puesto que tienen acceso ilimitado a todas las tareas e información.

También cabe destacar la clase Incidencia, puesto que la lógica de reglas de negocio depende estrictamente de esta. Las incidencias son registradas por los asegurados y enviadas a un panel de control (dashboard), para que la aseguradora pueda consultarlas y tratarlas. Vendría a pasar lo mismo con la clase Póliza e Historial de Incidencias y Fraudes.

A través del siguiente diagrama se puede observar la relación:

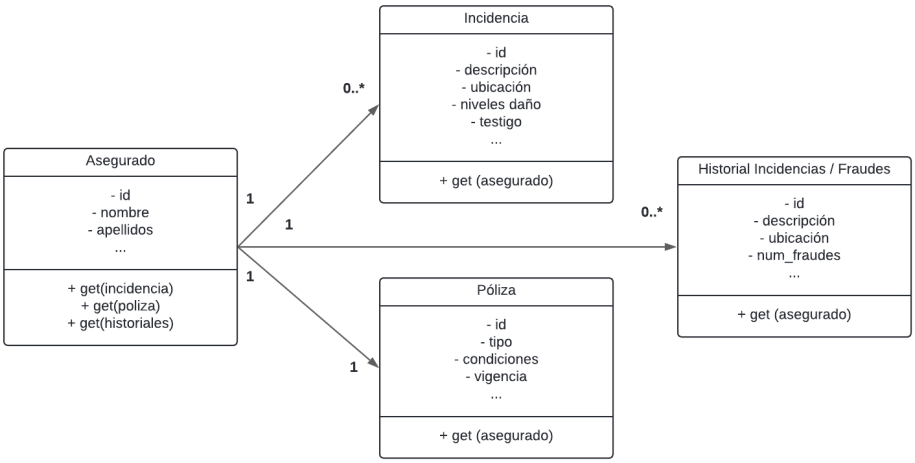


Ilustración 5: Diagrama de clases del asegurado

A su vez, teniendo en cuenta la interacción entre objetos, el siguiente diagrama de secuencias refleja el caso de uso de alta de una incidencia.

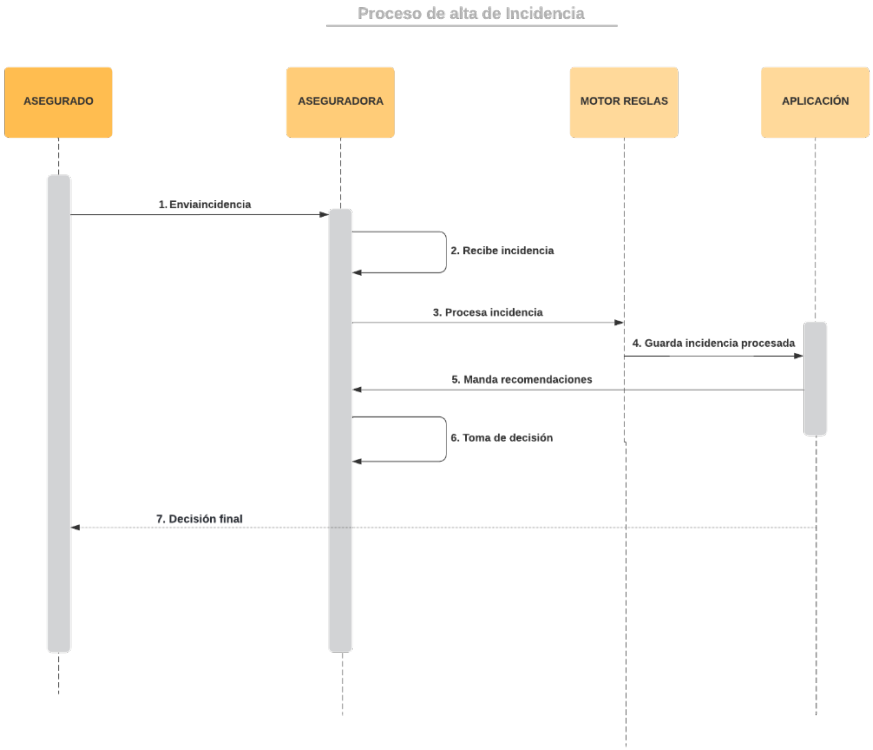


Ilustración 6: Diagrama del proceso de alta de una incidencia (siniestro)

A su vez, una incidencia podrá pasar por diferentes estados. La modificación de estos se hará de forma manual a través de un empleado de la aseguradora. Esta decisión tiene que ser humana, aunque estará apoyada por las decisiones del motor de reglas.

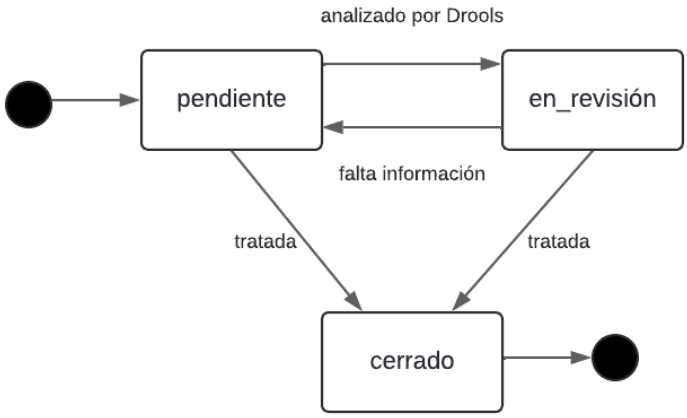


Ilustración 7: Diagrama de estados de una incidencia

3.3 Análisis de seguridad

El sistema desarrollado hace uso de datos de carácter personal y realiza peticiones a otro programa "externo", por lo que se procede a realizar un análisis de la seguridad.

Autenticidad

Objetivo: Garantizar que solo usuarios y sistemas autenticados puedan acceder y operar dentro del entorno local.

Medidas a Implementar:

- *Autenticación Rigurosa*: Se implementan mecanismos de autenticación fuertes dentro de Django para asegurar que solo usuarios autorizados accedan al sistema. Como es el cifrado de contraseñas mediante el algoritmo SHA-256.
- *Validación de Peticiones en Spring Boot:* Spring Boot está configurado para validar rigurosamente las peticiones provenientes del dominio local de Django, asegurando que no se acepten peticiones externas no autorizadas.

Confidencialidad

Objetivo: Proteger la información sensible manejada por la aplicación y Drools de accesos no autorizados.

Medidas a Implementar:

- *Cifrado de Datos*: Utilizar cifrado TLS/SSL para todas las comunicaciones entre Django y Drools, incluso en un entorno local, para evitar la exposición de datos sensibles.
- *Control de Acceso*: Aplicar políticas de control de acceso estrictas en Django para limitar el acceso a la información basado en roles y responsabilidades del usuario.

Integridad

Objetivo: Asegurar que los datos no sean alterados o corrompidos inadvertidamente o de manera malintencionada.

Medidas a Implementar:

- Integridad de Datos: Implementar controles de integridad en la base de datos y aplicar firmas digitales para datos críticos intercambiados entre Django y Drools.
- Registro y Monitoreo de Cambios: Usar registros (logs) detallados para monitorear y auditar todos los cambios en los datos.

Disponibilidad

Objetivo: Asegurar que el sistema esté disponible para los usuarios autorizados cuando sea necesario, sin interrupciones indebidas.

Medidas a Implementar:

- *Resiliencia del Sistema*: Diseñar el sistema para ser resistente a fallos, considerando el uso de técnicas de redundancia y failover incluso en un ambiente local.

- *Backups y Recuperación*: Establecer políticas de backup regulares y planes de recuperación ante desastres para restaurar rápidamente el sistema y los datos en caso de fallo.

Trazabilidad

Objetivo: Mantener un registro completo y preciso de todas las actividades dentro del sistema para permitir la auditoría y el seguimiento de acciones específicas.

Medidas a Implementar:

- *Sistemas de Registro*: Implementar sistemas de logging avanzados que capturen detalles de todas las operaciones realizadas, incluidas las peticiones de usuarios y las respuestas del sistema.
- *Auditorías Periódicas*: Realizar auditorías regulares del sistema para revisar y validar la trazabilidad y las prácticas de registro.

Capítulo 4

Diseño e implementación

4.1 Arquitectura del sistema

La arquitectura del sistema desarrollado se ha elaborado sobre el diagrama de clases expuesto anteriormente, donde diferenciábamos la Aplicación, Base de Datos y Motor de Reglas. Esta arquitectura está diseñada para facilitar la gestión eficiente de siniestros, soportando múltiples roles de usuario, manejo eficiente de los datos y comunicación segura entre componentes del sistema.

En cuanto a los elementos externos, se ha integrado Drools para la gestión y ejecución de reglas de negocio, facilitando decisiones automáticas basadas en datos ingresados en la aplicación y en contraste con los ya definidos. Este motor de reglas se comunica con la aplicación mediante una API desarrollada gracias a Spring Boot, que recibe y envía datos en formato JSON. Para la persistencia de los datos, se utiliza SQLite, un sistema de gestión de bases de datos ligero que almacena la información relativa a las incidencias, usuarios, pólizas y otros datos relevantes para la operación de la aplicación.

Patrón de Arquitectura

Siguiendo las buenas prácticas de desarrollo de aplicaciones web y el patrón arquitectural Model-View-Controller (MVC) recomendado por numerosos frameworks de desarrollo, incluido Django, el diseño de la aplicación se ha estructurado de la siguiente manera:

- Modelo: Encapsula la lógica de datos y las reglas de negocio. En Django, los modelos definen la estructura de las bases de datos y los métodos para acceder y gestionar los

datos. Además, se encargan de la comunicación con el motor de reglas Drools para procesar las decisiones basadas en los datos actuales.

- Vista: Representa la capa de presentación de la aplicación. En Django, las vistas gestionan la recepción de solicitudes HTTP y la respuesta con el renderizado de las plantillas HTML, las cuales presentan la información al usuario de manera estructurada.

- Controlador: En el contexto de Django, el controlador está representado por las vistas (que en otros frameworks se denominan controladores), que actúan como intermediario entre los modelos y las vistas, gestionando la lógica de control, recibiendo inputs del usuario y devolviendo la salida adecuada.

Flujo de Datos

Eventos de Usuario: Los eventos son generados por la interfaz de usuario como respuesta a acciones del usuario, como el alta de un nuevo siniestro o la consulta de incidencias existentes.

Gestión del Estado: En el patrón MVC, el controlador actualiza el modelo y modifica la vista según el cambio de estado derivado de las operaciones de los usuarios, manteniendo una clara separación entre la presentación y la lógica de negocio.

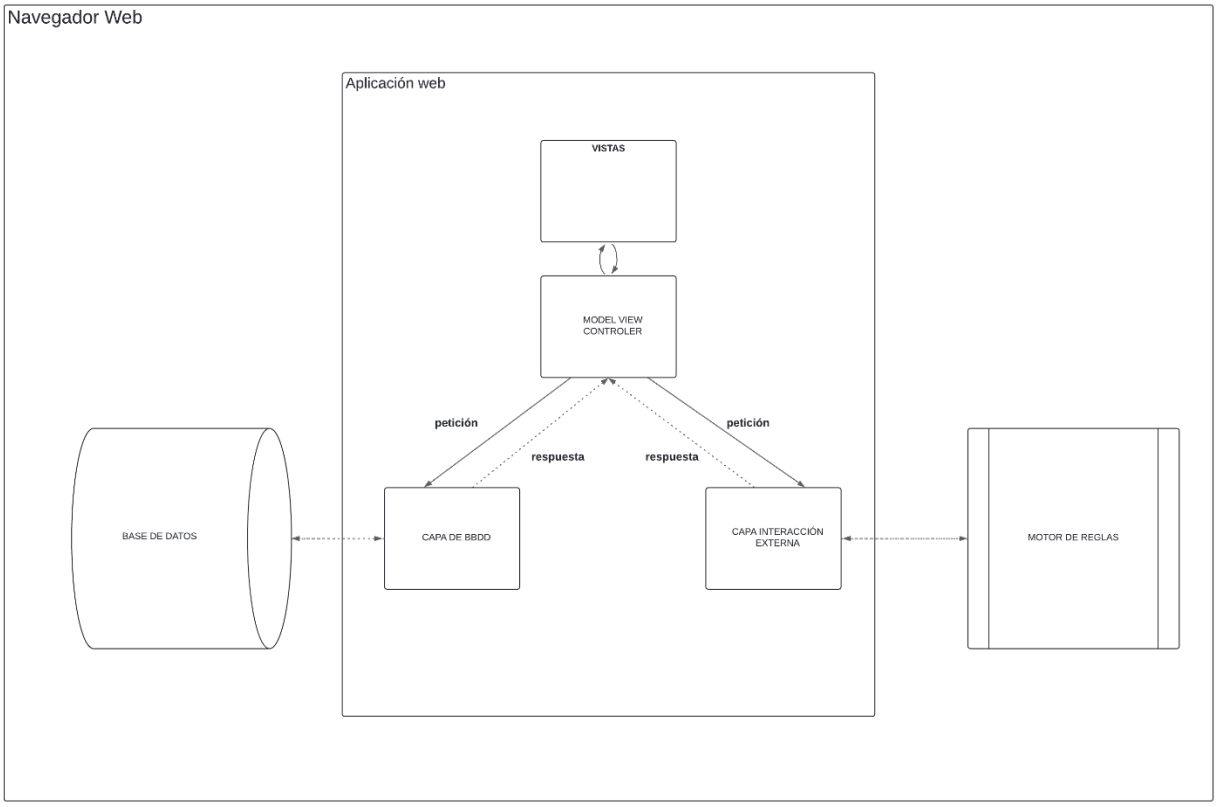


Ilustración 8: Arquitectura del sistema desarrollado

4.1.1 Arquitectura física

Componentes Físicos

La infraestructura física para desplegar y operar la aplicación web es relativamente simple, dado que se basa principalmente en un entorno de desarrollo y ejecución local.

Hardware y Dispositivos:

- Computadora Personal/Local: La aplicación y todas sus dependencias, incluyendo el servidor web Django y la instancia de Drools, se ejecutan en una computadora personal o un servidor local. Esta computadora debe tener suficiente capacidad de procesamiento y memoria para manejar las aplicaciones de servidor web y el motor de reglas sin problemas de rendimiento.

- Especificaciones Recomendadas:
    - CPU: Intel Core i5 o superior / AMD Ryzen 5 o equivalente.
    - Memoria RAM: Mínimo 8 GB, recomendado 16 GB para un mejor desempeño.
    - Almacenamiento: SSD con al menos 256 GB para un arranque y acceso rápidos a datos.

Topología de Red

Dado que la aplicación opera completamente en un entorno local, la topología de red es minimalista y directa.

Diagrama de Topología de Red:

- Conexión Local: La computadora personal actúa como servidor y cliente, lo que significa que no se requieren conexiones de red complejas o dispositivos de red adicionales como routers o switches externos. La conexión de red se limita a la red interna del dispositivo, lo cual simplifica enormemente la configuración y reduce los riesgos de seguridad relacionados con conexiones externas.

Configuración de Servidores

La configuración del servidor se enfoca en la instalación y ejecución de la pila de software necesaria para la aplicación.

Configuración del Servidor Local:

Sistema Operativo: Puede ser cualquier sistema operativo que soporte Python y Java, como Windows 10/11, macOS, o una distribución de Linux como Ubuntu o CentOS.

- *Software*:
    - Django: Se instala dentro de un entorno virtual de Python para manejar las dependencias de forma aislada.

  - Drools: Ejecutado a través de Maven, también se configura para operar en el entorno local, recibiendo y procesando solicitudes únicamente del servidor Django local.
- *Seguridad*: Implementación de medidas de seguridad básicas en el sistema operativo, como firewalls y actualizaciones regulares de seguridad para proteger los datos y las operaciones del sistema.

4.1.2 Arquitectura lógica

Introducción

La arquitectura lógica de la aplicación está diseñada para separar claramente las responsabilidades entre la interfaz de usuario, la lógica de procesamiento y la gestión de datos. Esto no solo mejora la organización del código y facilita el mantenimiento, sino que también optimiza el rendimiento y la seguridad.

Capa de Presentación (Vista)

Descripción: Esta capa es responsable de todas las interacciones con el usuario, incluyendo la visualización de datos, la recolección de entradas de usuario y la presentación de resultados procesados. En Django, esto se maneja a través de *templates* HTML que son renderizados por las vistas basadas en las solicitudes de los usuarios.

Componentes:

- Templates HTML/CSS/JavaScript: Define la estructura y el estilo de la interfaz de usuario.
- Vistas de Django: Funcionan como controladores en el patrón MVC y manejan la lógica de decisión sobre qué datos enviar a la interfaz de usuario y qué plantilla utilizar.

Capa de Lógica de Negocio (Controlador)

Descripción: Encargada de procesar la información, aplicar reglas de negocio, tomar decisiones y coordinar las acciones entre la capa de presentación y la capa de datos. Esta capa también incluye la integración con el sistema Drools para el procesamiento de reglas de negocio.

Componentes:

- Vistas/Controladores en Django: Aquí se implementan los flujos de control, se toman decisiones basadas en la entrada del usuario y se invocan las acciones correspondientes en la capa de modelo.
- Cliente API para Drools: Gestiona la comunicación entre Django y Drools, enviando datos y recibiendo decisiones de reglas a través de solicitudes HTTP.

Capa de Acceso a Datos (Modelo)

Descripción: Esta capa gestiona todo el acceso a la base de datos, incluyendo la creación, lectura, actualización y eliminación de datos (CRUD). Asegura que los datos requeridos por la aplicación sean obtenidos de manera eficiente y segura.

Componentes:

- Modelos de Django: Definen la estructura de la base de datos y proporcionan métodos para acceder y manipular los datos de SQLite3.
- Migraciones: Permiten la modificación segura y controlada del esquema de la base de datos a medida que evoluciona el desarrollo de la aplicación.

Comunicación y Gestión de Datos

Integración con Drools: Se envían y reciben datos estructurados en JSON para el procesamiento de reglas. Este intercambio se maneja de manera segura y eficiente para asegurar que las decisiones de negocio se basen en los datos más actualizados y precisos.

Seguridad y Autenticación: Se implementan mecanismos para asegurar que solo usuarios autenticados y autorizados puedan acceder a ciertas funciones y datos dentro de la aplicación, usando el sistema de autenticación y permisos de Django.

4.2 Diseño de datos

Descripción General

El diseño de la base de datos para la aplicación es crítico para asegurar un manejo eficiente y seguro de la información relacionada con los usuarios y sus actividades. Este diseño se basa en un modelo relacional estructurado alrededor de la tabla principal Usuario y sus relaciones con diversas entidades que describen aspectos de su interacción con la aseguradora.

Diagrama Entidad-Relación Extendido

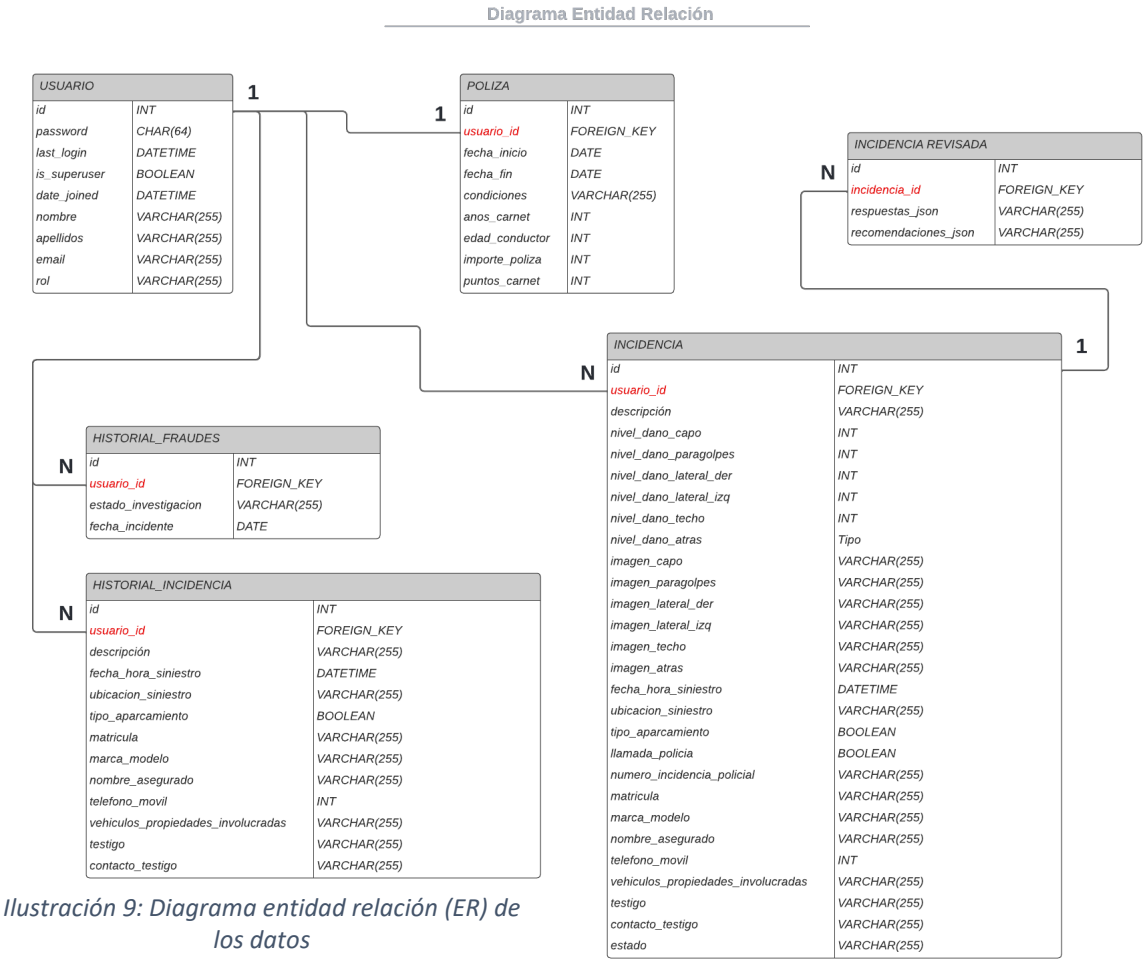


Ilustración 9: Diagrama entidad relación (ER) de los datos

Proceso de Normalización

La normalización es un componente crítico en el diseño de bases de datos relacionales, destinado a reducir redundancias y mejorar la integridad de los datos. El proceso de normalización se divide en varias 'formas normales' que son pasos sistemáticos para eliminar anomalías en la base de datos.

Primera Forma Normal (1NF)

Objetivo: La primera forma normal (1NF) busca simplificar los registros en la base de datos asegurando que:

- Clave Primaria: Cada tabla tiene una clave primaria única, que identifica inequívocamente cada fila o registro en la tabla. Esto es crucial para mantener la integridad de los datos, permitiendo que cada elemento de datos pueda ser accedido precisamente.
- Atributos Atómicos: Todos los atributos en la tabla deben ser atómicos, lo que significa que no pueden ser descompuestos en partes más pequeñas que tengan significado por sí solos dentro del contexto de la base de datos. Por ejemplo, un campo como "nombre completo" debe dividirse en "nombre" y "apellidos" para garantizar que cada atributo contenga solo una pieza de información.
- Ausencia de Grupos Repetitivos: No debe haber grupos de columnas repetitivas o matrices de datos que puedan ser separadas en distintas tablas para eliminar la duplicidad y facilitar la modificación de los datos sin errores.

Segunda Forma Normal (2NF)

Objetivo: La segunda forma normal (2NF) se construye sobre la base de la 1NF para aumentar la cohesión de los datos en las tablas.

- Dependencia Completa: En 2NF, es crucial que cada atributo no clave en una tabla sea dependiente de toda la clave primaria para casos de claves compuestas. Esto significa que cada atributo no clave debe estar relacionado con toda la clave primaria, y no solo con parte de ella.

- Eliminación de Dependencias Parciales: Si un atributo no clave depende solo de una parte de una clave primaria compuesta, entonces ese atributo debe separarse en una tabla diferente. Por ejemplo, si en la tabla que registra la revisión de incidencias, algún atributo como "recomendaciones" depende solo del "id del usuario" y no del "id de la incidencia", debe separarse para cumplir con 2NF.

Tercera Forma Normal (3NF)

Objetivo: La tercera forma normal (3NF) se enfoca en eliminar la dependencia transitiva de los atributos no clave.

- Dependencias Transitivas: Un atributo no clave no debe depender de otro atributo no clave. Por ejemplo, si en una tabla de incidencias, el "nombre del asegurador" depende del "id del asegurador" que a su vez depende de la "id de la incidencia", entonces esto constituye una dependencia transitiva.
- Separación de Entidades: Los atributos que causan dependencias transitivas deben moverse a una tabla separada para asegurar que todas las dependencias sean directas y no indirectas.

4.3 Diseño de la interfaz de usuario

Flujo de Navegación

Comenzando desde la página de inicio de sesión o registro, el usuario puede crearse una cuenta o iniciar sesión con una ya existente. Ambas peticiones se manejan desde una única página. A través de un switch se pasará del ‘Log In’ al ‘Sign Up’. En este punto se divide el flujo de navegación, dependiendo del tipo de usuario que haya iniciado sesión (Ilustración 10: Wireframe Login/Registro)

Empezando por el usuario que acaba de registrarse, este, como se ha mencionado anteriormente en el presente documento, carecerá de acciones. Simplemente tendrá acceso a un blog, con la información relativa a la empresa aseguradora.

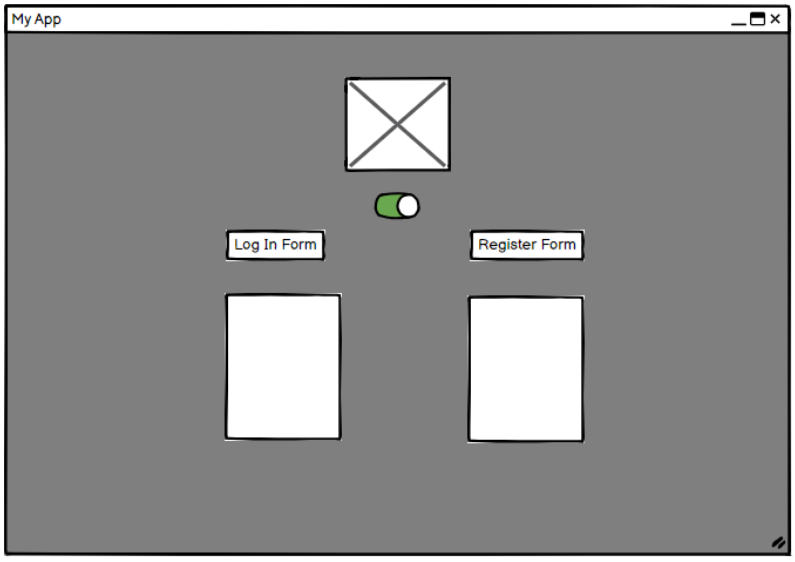


Ilustración 10: Wireframe Login/Registro

Para poder llevar a cabo acciones, un administrador deberá concederle un 'rol' (más adelante veremos cómo se hace).

Si a esta persona se le concede el rol de *usuario*, pasará a ser un asegurado, y tendrá acceso a dos pestañas (Ilustración 11: Wireframe Home usuario):

- Nueva Incidencia: desde donde se podrá crear una incidencia relativa a un siniestro. Deberá rellenar los campos correspondientes para que la incidencia sea válida. A parte en esta página el asegurado podrá ayudarse de un modelo 3D de un vehículo, y a través de este, señalar los niveles de daño en 6 diferentes partes. Una vez haga *click* en una parte del vehículo se desplegará un nivelador de daño en la parte izquierda de la pantalla, y a su vez, un botón para poder adjuntar una imagen de dicha parte. Con todos los datos rellenados, el asegurado enviará la incidencia (Ilustración 12: Wireframe 'Nueva Incidencia').
- Mis Incidencias: desde donde se podrán consultar las incidencias enviadas previamente. Cuando la incidencia esté revisada por un trabajador de la aseguradora, esta cambiará de estado y habilitará la función de descarga de incidencia en formato PDF. Este PDF incluirá toda la información relativa a la incidencia, así como su estado (Ilustración 13: Wireframe 'Mis incidencias').

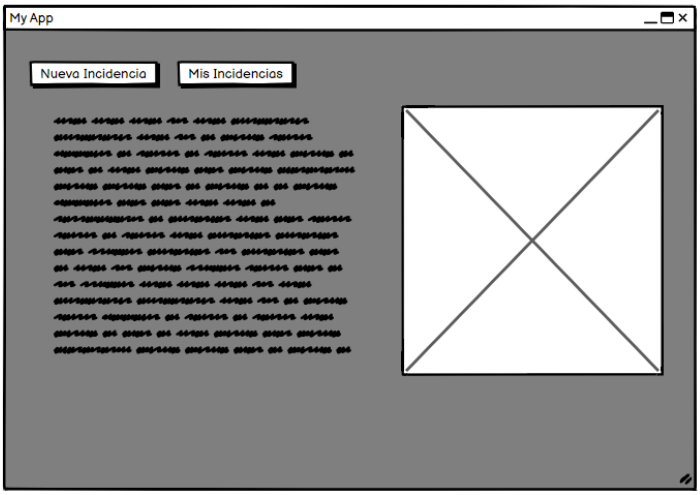


Ilustración 11: Wireframe Home usuario

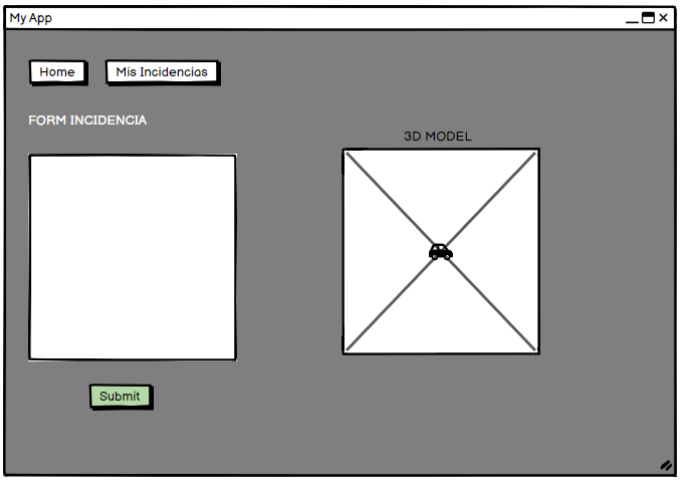


Ilustración 12: Wireframe 'Nueva Incidencia'

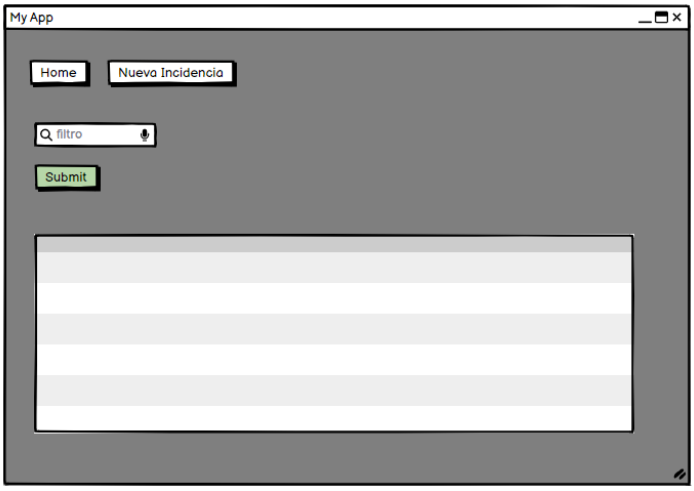


Ilustración 13: Wireframe 'Mis incidencias'

Hasta aquí el punto de vista del asegurado. Se continúa con los trabajadores de la aseguradora.

Si el administrador concede el rol de 'worker' al usuario recién registrado, este pasará a ver accesibles tres pestañas (Ilustración 14: Wireframe Home trabajador):

- Dashboard: desde donde podrán consultar las incidencias de los asegurados a través de un filtro de diferentes campos (Ilustración 15: Wireframe 'Dashboard'). Una vez elegida la incidencia a resolver, el trabajador hará *click* en ella, y se abrirán los detalles de la mismas. En este punto se puede analizar la incidencia, u observar el análisis (Ilustración 16: Wireframe 'Detalle incidencia')
- Historial de usuarios: desde donde podrán importar en formato CSV, incidencias o fraudes de un usuario (Ilustración 17: Wireframe 'Historial usuarios'). También se podrán observar las incidencias y los fraudes de los usuarios.
- Gestión póliza: desde donde podrán modificar las pólizas de los asegurados.(Ilustración 18: Wireframe 'Gestión póliza')

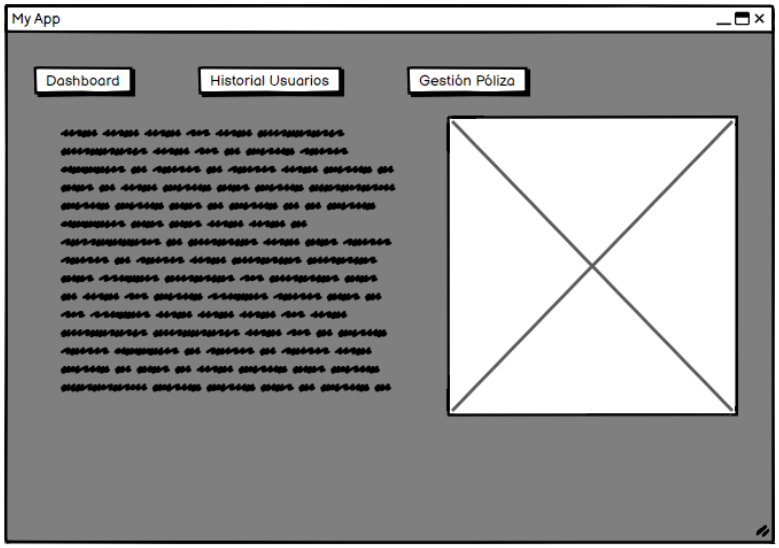


Ilustración 14: Wireframe Home trabajador

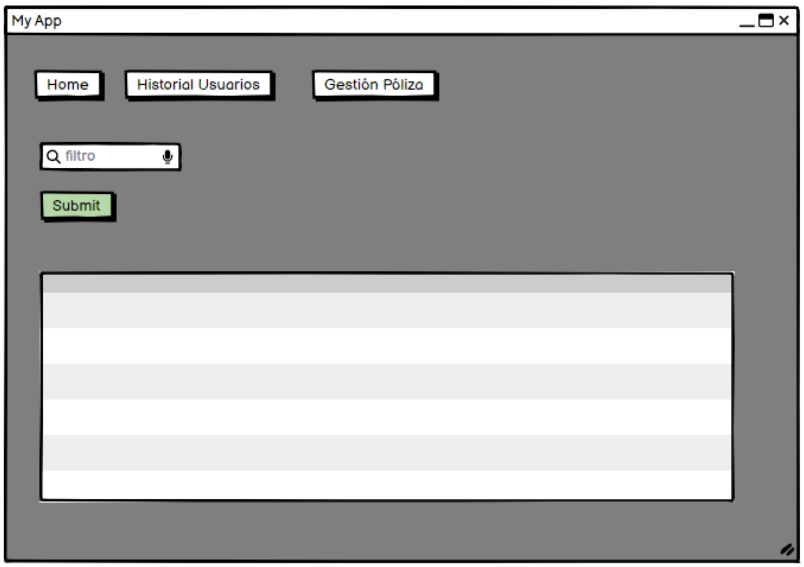


Ilustración 15: Wireframe 'Dashboard'

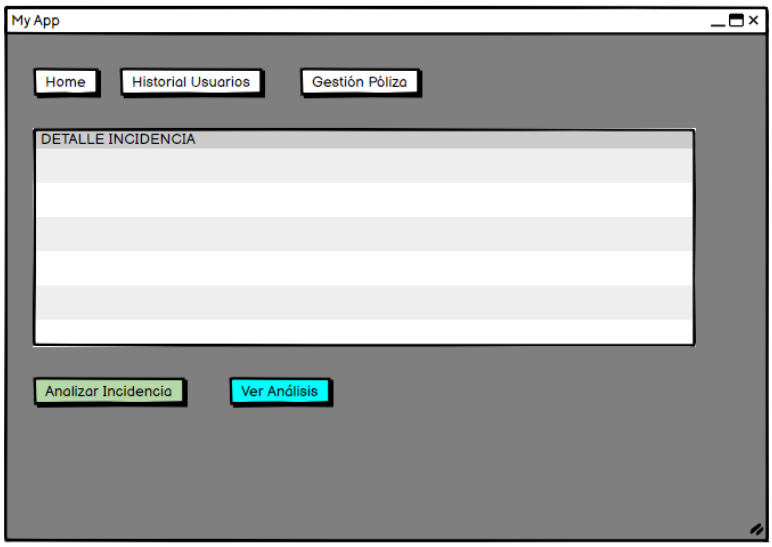


Ilustración 16: Wireframe 'Detalle incidencia'

Ilustración 17: Wireframe 'Historial usuarios'

Ilustración 18: Wireframe 'Gestión póliza'

Hasta aquí el punto de vista de la aseguradora. Se continúa con los administradores del software.

Los administradores tendrán acceso a todas las características mencionadas previamente (Ilustración 19: Wireframe Home administrador), y aparte, a una pestaña 'Permisos usuarios' desde donde podrán cambiar el rol de todos los usuarios del sistema (Ilustración 20: Wireframe 'Permisos usuario'). Desde esta pestaña también se podrán ver los usuarios pendientes de verificación.

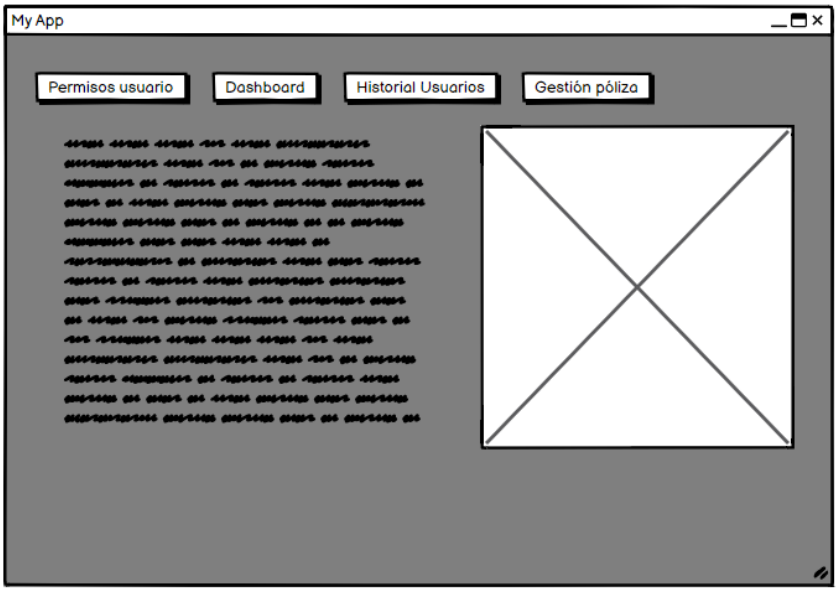


Ilustración 19: Wireframe Home administrador

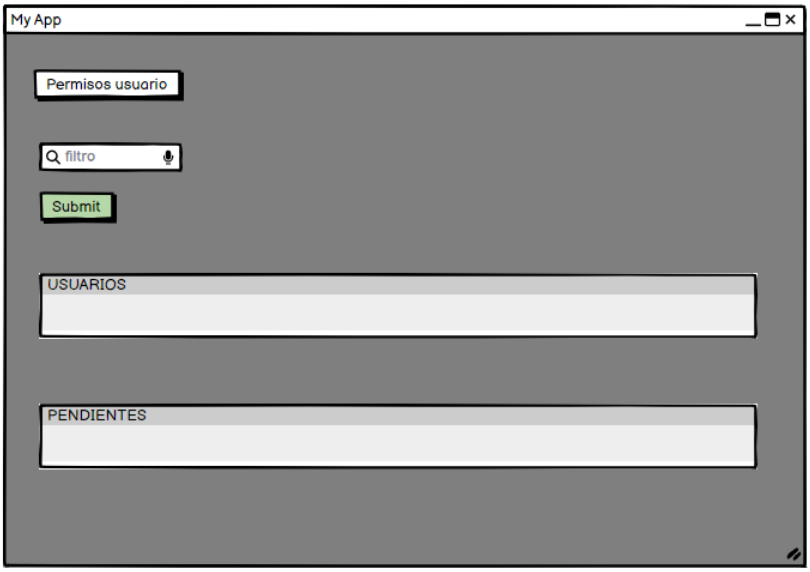


Ilustración 20: Wireframe 'Permisos usuario'

4.4 Diagrama de clases

A lo largo de este documento se ha ido describiendo cada clase que interactúa en este software. Se ha construido de forma incremental, creando y añadiendo los componentes necesarios y modificando el diseño cuando se necesitaba. En todo

momento se ha pretendido reducir la complejidad al programar, teniendo en cuenta un análisis previo a la ejecución. La aplicación en su totalidad se compone de tres grandes pilares:

- Aplicación web: parte visual, donde el usuario interactúa. Se ha diferenciado entre los roles que pueden asumir los usuarios, las acciones, y los atributos de los mismos.
- Base de datos: parte gestora de los datos. Se han diferenciado los diferentes modelos de tablas, así como la relación entre ellas.
- Motor de reglas: parte oculta del software encargada de apoyar la toma de decisiones de la aseguradora.

Para poder entender la relación entre estos tres componentes me apoyo en el siguiente esquema, y a continuación de este se explicará la interacción entre todas las clases.

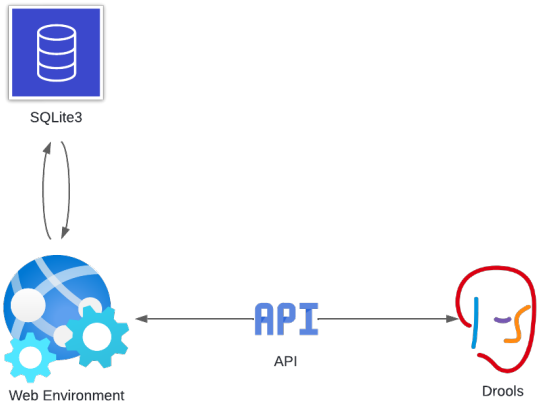


Ilustración 21: Diagrama interacción clase software

- Aplicación web: interactúa con SQLite3 y con Drools. Dependiendo del rol de usuario y su id, la web mostrará unos modelos de la base de datos u otros. Una vez filtrado el rol, dependiendo de la acción que se tome, se actualizarán las tablas correspondientes. En el siguiente diagrama se puede observar estas acciones simplificadas:

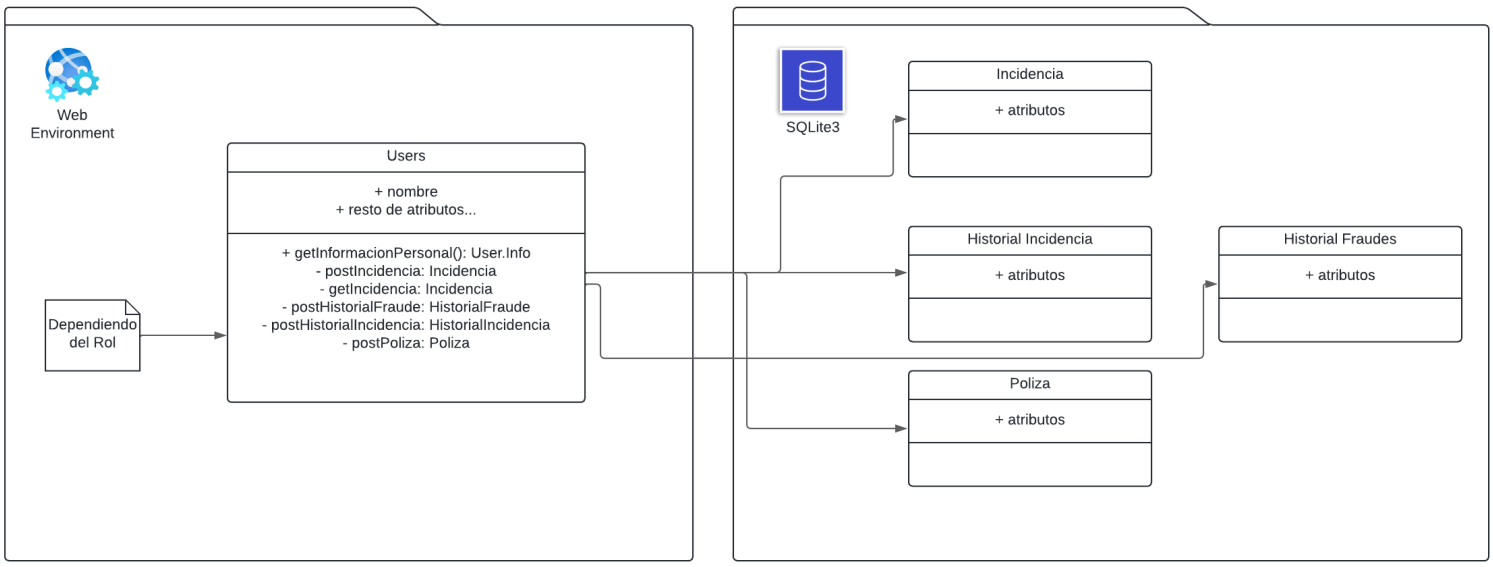


Ilustración 22: Diagrama interacción web - bbdd

La interacción con Drools es a través de una API. En este punto interactúan los tres componentes. El procedimiento es el siguiente:

- Al analizar una incidencia, sea crea un objeto JSON, que contiene atributos sobre la incidencia, y aparte, sobre el asegurado. Los datos del asegurado son obtenidos de las tablas Historial Incidencias, Historial Fraudes y Póliza. Por lo tanto, antes de enviarle a Drools el objeto, hay una consulta a SQLite3 para obtener dichos datos.
- Una vez enviados a Drools, este ejecuta las reglas y almacena en dos arrays las conclusiones. Pueden ser *respuestas* o *recomendaciones*. Una vez almacenadas, las envía de vuelta en formato JSON a la web y seguidamente son almacenadas en la base de datos.

El objeto JSON que se manda desde la web debe coincidir con los parámetros establecidos en la clase Java de Drools, *Siniestro*. Esta clase está compuesta por:

```java
public class Siniestro {
    private LocalDateTime fechaYHora;
    private String ubicacion;
    private String tipoAparcamiento;
    private String llamadaPolicia;
    private String matricula;
    private String numeroIncidenciaPolicial;
    private String testigo;
    private String contactoTestigo;
    private String vehiculosPropiedadesInvolucrados;
    private int reincidenciaUltimoAnno;
    private int reincidencia3Anno;
    private int fraudes;
    private String tipoCobertura;
    private LocalDate inicioPoliza;
    private LocalDate finPoliza;
    private int edadConductor;
    private int anosCarnet;
    private double importePoliza;
    private String usoVehiculo;
    private int puntosCarnet;
    private String extrasPoliza;
```

Ilustración 23: Clase 'Siniestro' en Java (Drools)

A continuación, el esquema que modela la interacción con Drools.

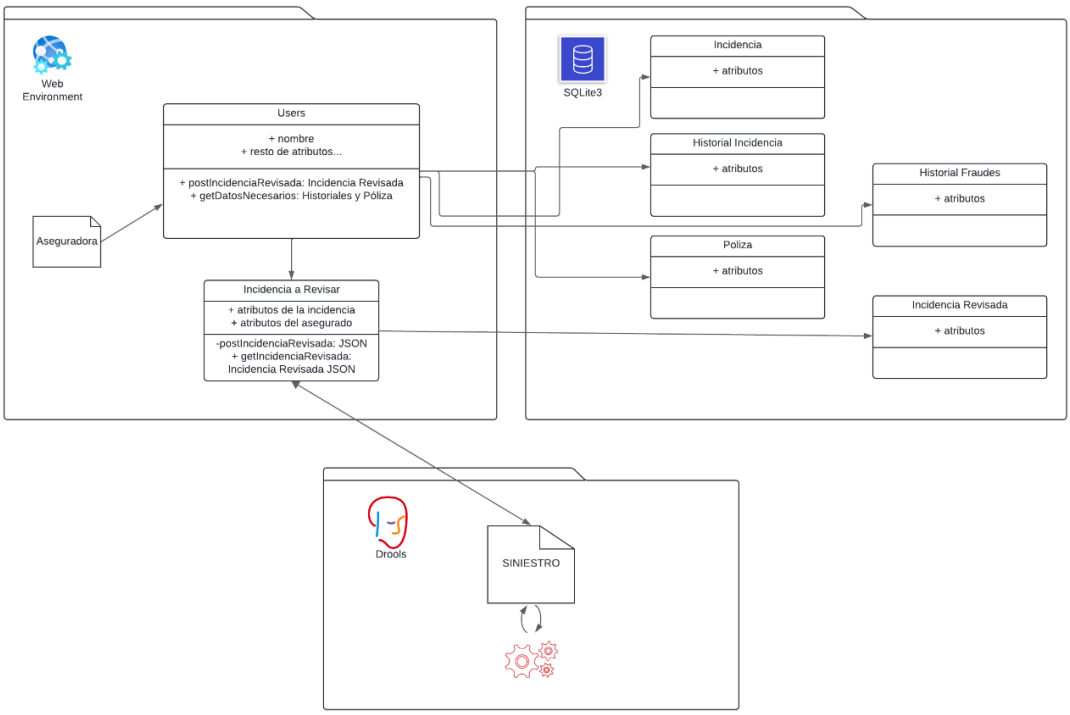


Ilustración 24: Diagrama interacción web - bbdd - drools

Como se ha mencionado a lo largo de este documento la comunicación que se establece entre la aplicación web y Drools se hace mediante una API. Esta API se divide en dos partes, ambas están configuradas como cliente y servidor.

1. **Web**

enviarDatosDrools()

Esta función prepara y envía datos de incidencia en formato JSON a una API de Drools para procesar un siniestro. La estructura de la función es la siguiente:

- Preparación de Datos: Primero, construye un objeto JavaScript *datosIncidencia*. Estos datos incluyen detalles sobre un siniestro, como ubicación, fechas, matrícula del vehículo, información sobre el testigo, detalles de la póliza de seguro, y más.
- Envío de Datos a Drools: Utiliza la función fetch para realizar una petición POST a Drools, enviando *datosIncidencia* en formato JSON.
- Manejo de la Respuesta de Drools: Al recibir una respuesta de Drools, primero verifica si la respuesta fue exitosa. Si es así, procesa la respuesta esperando que también sea en formato JSON. Luego, registra la respuesta y prepara un nuevo objeto respuestasRecomendaciones que incluye la identificación de la incidencia y la información recibida de Drools.
- Envío de Datos Procesados a Django: Realiza otra petición POST a un endpoint de Django, incluyendo un token CSRF obtenido de las cookies para protección contra ataques de solicitud de sitio cruzado (CSRF). Aquí es donde se utiliza la función getCookie(name).
- Manejo de la Respuesta de Django: Similar al manejo de la respuesta de Drools, verifica la respuesta y procesa los datos recibidos, registrando cualquier información relevante.

getCookie(name)

Esta función ayuda a obtener el valor de una cookie específica por su nombre, lo cual es crucial para la seguridad de las peticiones entre cliente y servidor. Es utilizada en este

contexto para obtener el token CSRF necesario para las peticiones POST a Django, asegurando que la petición sea genuina y autorizada.

2. Drools

DroolsConfig (Configuración de Drools)

- KieContainer Bean: Esta configuración inicia un contenedor de reglas KIE, que es necesario para contener y manejar las sesiones de reglas.
- KieSession Bean: Se crea una sesión de KIE a partir del contenedor, especificando ksession-rules, es el nombre de la sesión de reglas definida en el archivo kmodule.xml. Esta sesión se usa para insertar datos y disparar reglas.
- corsConfigurer Bean: Configura políticas CORS (Cross-Origin Resource Sharing) para la API, permitiendo solicitudes desde el origen http://127.0.0.1:8000. Esto es crucial para permitir que el frontend interactúe con el backend sin violar la política de mismo origen.

SiniestroController (Controlador para el procesamiento de siniestros)

- Dependencias: Utiliza KieSession para la lógica de reglas y ObjectMapper para serialización/deserialización de objetos JSON.
- Endpoint /procesar-siniestro (POST): Este método maneja solicitudes POST que reciben datos de siniestro en formato JSON. El flujo de procesamiento es el siguiente:
    - Deserialización: Convierte la cadena JSON recibida en un objeto Siniestro usando ObjectMapper.
    - Ejecución de Reglas: Inserta el objeto Siniestro en la sesión de Drools, configura listas globales para almacenar respuestas y recomendaciones, y dispara las reglas definidas.
    - Respuesta: Almacena las respuestas y recomendaciones generadas en un mapa y devuelve este resultado como una respuesta HTTP.
    - Manejo de Errores: Si ocurre algún error durante el proceso, captura la excepción, registra el error y envía una respuesta HTTP de error con detalles del mismo.

4.5 Entorno de construcción

Descripción General

Este proyecto se ha desarrollado utilizando una combinación de tecnologías modernas y frameworks probados para asegurar robustez, eficiencia y facilidad de mantenimiento. A continuación, se detalla cada componente del entorno de construcción.

Herramientas y Frameworks

- Django:
  - Descripción: Un framework de desarrollo web de alto nivel escrito en Python, conocido por su simplicidad y capacidad para desarrollar aplicaciones web rápidamente.
  - Uso: Gestión de la lógica del servidor, modelos de datos, y el renderizado de las vistas HTML.
- Bootstrap:
  - Descripción: Un framework de front-end para desarrollar aplicaciones web y sitios móviles responsivos con HTML, CSS y JS.
  - Uso: Utilizado para diseñar la interfaz de usuario de la aplicación, asegurando que sea estética y funcionalmente atractiva en dispositivos de todos tamaños.
- Visual Studio Code:
  - Descripción: Un editor de código fuente desarrollado por Microsoft, que soporta múltiples lenguajes de programación y frameworks.
  - Uso: IDE principal para el desarrollo, que ofrece potentes herramientas de edición, gestión del código fuente y debugging.
- Spring Boot:
  - Descripción: Framework de Java para la creación de aplicaciones de microservicios, con configuración mínima.

  - Uso: Implementación de la API que facilita la comunicación entre la interfaz de usuario y el motor de reglas Drools.
- Drools:
  - Descripción: Un motor de reglas de negocio basado en Java que permite la definición de reglas de negocio complejas fuera del código de la aplicación.
  - Uso: Manejo de la lógica de negocios para el procesamiento de siniestros y decisiones automáticas.
- Maven:
  - Descripción: Una herramienta de gestión y comprensión de proyectos software.
  - Uso: Usado para gestionar las dependencias de proyecto, la construcción y despliegue de la aplicación Spring Boot.

Librerías y Plugins

- KIE (Knowledge Is Everything) Workbench:
  - Uso: Herramienta dentro de Drools para la gestión de reglas, procesos, y el despliegue del repositorio.
- HTML, CSS, JavaScript:
  - Descripción: Tecnologías estándar para el desarrollo web.
  - Uso: Creación de páginas web interactivas y estilizadas.
- JQuery:
  - Descripción: Una librería de JavaScript rápida y concisa que simplifica el manejo del HTML document traversal, event handling, animating, y Ajax.
  - Uso: Utilizada para simplificar los scripts del lado del cliente y mejorar la interactividad del usuario.

Simuladores y Herramientas de Pruebas

- JUnit:
  - Descripción: Un framework de pruebas para Java, usado para realizar pruebas unitarias y de integración.

  - Uso: Pruebas del backend, especialmente los componentes de Spring Boot.
- Selenium:
  - Descripción: Un framework para pruebas de aplicaciones web en varios navegadores y plataformas.
  - Uso: Pruebas automatizadas de la interfaz de usuario.

Control de Versiones

- Git:
  - Descripción: Sistema de control de versiones distribuido para manejar todo desde pequeños a muy grandes proyectos con velocidad y eficiencia.
  - Uso: Gestión del código fuente y colaboración.

4.6 Referencia al repositorio de software

El software se encuentra disponible en su totalidad en el siguiente repositorio.

https://github.com/gonzalopaul/Aplicacion_TFG

Capítulo 5

Validación del sistema

5.1 Plan de pruebas

El objetivo del Plan de Pruebas es garantizar que la aplicación web cumpla con los requisitos funcionales y no funcionales establecidos, asegurando su correcto funcionamiento, rendimiento y accesibilidad. Este plan incluye diferentes tipos de pruebas, tales como pruebas unitarias, de integración, End-to-End, de estrés, de carga y de accesibilidad. A continuación, se detalla cada tipo de prueba y los criterios para su realización.

Pruebas Unitarias

Descripción: Verifican el comportamiento de unidades individuales de código, como funciones o métodos de los modelos.

Criterios:

- Cada función y método crítico debe tener al menos una prueba unitaria asociada.
- Se deben cubrir tanto los casos de uso comunes como los bordes de los inputs esperados.
- Implementación: Utilizando unittest y pytest, se crean pruebas para los modelos Incidencia, Poliza y HistorialFraudes, asegurando que cada campo se guarda y se maneja correctamente.

Pruebas de Integración

Descripción: Validan la interacción entre diferentes componentes del sistema.

Criterios:

- Verificación de la correcta comunicación entre la aplicación web (Django) y el motor de reglas (Drools) a través de la API.
- Pruebas de flujo de datos desde la interfaz de usuario hasta el motor de reglas y viceversa.

Implementación: Se crean pruebas para las vistas que manejan la creación, actualización y eliminación de siniestros, asegurando que los datos se manejan adecuadamente.

Pruebas End-to-End (E2E)

Descripción: Simulan escenarios de uso real para asegurar que todo el sistema funciona correctamente desde la perspectiva del usuario.

Criterios:

- Validación de los flujos principales de usuario, como la creación, actualización y eliminación de siniestros.
- Asegurar que la interfaz de usuario responde correctamente a las acciones del usuario.

Implementación: Utilizando Selenium para automatizar la interacción con la aplicación web, incluyendo el inicio de sesión y la creación de una incidencia, asegurando que el flujo de trabajo completo del usuario es funcional.

Pruebas de Estrés y Carga

Descripción: Evaluan el rendimiento del sistema bajo condiciones de uso intensivo.

Criterios:

- Simulación de múltiples usuarios accediendo y operando en la aplicación simultáneamente.
- Medición de tiempos de respuesta y uso de recursos bajo cargas intensivas.

Implementación: Utilizando Locust para simular múltiples usuarios interactuando con la aplicación, midiendo el rendimiento y asegurando que el sistema puede manejar cargas elevadas.

Todas estas pruebas han sido creadas y testadas sobre el código. Los archivos se pueden encontrar en el repositorio bajo el directorio /test.

Capítulo 6

Conclusiones y líneas futuras

6.1 Conclusiones

Como estudiante del último año de Ingeniería de Sistemas de Información, la realización de este proyecto ha sido una experiencia enriquecedora y desafiante. El desarrollo de una aplicación web completa utilizando Django, junto con la integración de un motor de reglas Drools, ha permitido no solo aplicar los conocimientos adquiridos a lo largo de la carrera, sino también explorar nuevas tecnologías y enfoques para resolver problemas complejos en el ámbito de la gestión de siniestros vehiculares.

La aplicación desarrollada ofrece una solución integral para la gestión de siniestros de vehículos, abarcando desde el registro y validación de usuarios hasta la creación y visualización de distintos roles y puntos de vista: trabajadores de la aseguradora, asegurados y administradores. Esta diversidad de roles asegura que cada usuario tenga acceso a la información y las funcionalidades pertinentes a sus necesidades y responsabilidades específicas, mejorando así la eficiencia y efectividad en la gestión de siniestros.

Uno de los aspectos más destacables del proyecto ha sido la integración de Drools, un motor de reglas que permite la automatización del proceso de alta de siniestros y la detección de fraudes. A través de una API casera, la aplicación web envía datos relevantes a Drools, que luego procesa esta información y devuelve respuestas y recomendaciones. Esta integración ha permitido automatizar tareas críticas, reduciendo la carga de trabajo manual y minimizando el riesgo de errores humanos. La capacidad

de detectar fraudes de manera automatizada es un avance significativo que puede ahorrar costos y tiempo, además de aumentar la confiabilidad del sistema.

El uso de Django ha sido fundamental para el desarrollo del proyecto, proporcionando una estructura robusta y modular que facilita la implementación de funcionalidades complejas. La capacidad de Django para gestionar bases de datos de manera eficiente ha sido crucial para el almacenamiento y recuperación de grandes volúmenes de datos relacionados con los siniestros. Además, la implementación de pruebas unitarias e integrales ha asegurado que el sistema sea confiable y esté libre de errores.

En resumen, este proyecto ha demostrado cómo la combinación de tecnologías avanzadas puede resultar en una solución poderosa y eficiente para la gestión de siniestros vehiculares. La experiencia adquirida en el desarrollo de esta aplicación será invaluable para futuros desafíos profesionales en el campo de la ingeniería de sistemas de información.

6.2 Líneas futuras

Mirando hacia el futuro, el software desarrollado tiene un gran potencial para evolucionar y expandirse, incorporando nuevas funcionalidades y mejoras que pueden aumentar aún más su utilidad y eficiencia. Algunas de las direcciones más prometedoras para futuras implementaciones incluyen:

- Integración con Sistemas de Geolocalización:

Incorporar sistemas de geolocalización que permitan a los usuarios registrar siniestros directamente desde el lugar del incidente mediante el uso de coordenadas GPS. Esto no solo mejoraría la precisión de los datos, sino que también podría facilitar la asistencia en tiempo real, enviando alertas automáticas a servicios de emergencia o grúas cercanas.

- Análisis Predictivo y Machine Learning:

Implementar algoritmos de *machine learning* para mejorar la detección de fraudes y predecir posibles siniestros basados en patrones históricos. El análisis predictivo podría identificar comportamientos de alto riesgo y recomendar acciones preventivas tanto para los asegurados como para la aseguradora.

- Expansión de la API:

Desarrollar una API más robusta y completa que permita la integración con otros sistemas y servicios externos, como plataformas de terceros para la validación de documentos, sistemas de pago, y servicios de atención al cliente. Esto podría convertir la aplicación en una plataforma central para la gestión de siniestros, aumentando su versatilidad y capacidad de integración.

- Mejora de la Experiencia del Usuario:

Implementar una interfaz de usuario más intuitiva y accesible, utilizando tecnologías modernas como React o Vue.js. Además, desarrollar aplicaciones móviles nativas para iOS y Android que permitan a los usuarios gestionar siniestros de manera más cómoda y rápida desde sus dispositivos móviles.

- Automatización Avanzada de Procesos:

Ampliar el uso de Drools para incluir más reglas y escenarios, automatizando no solo la detección de fraudes sino también otros aspectos del proceso de gestión de siniestros, como la asignación de peritos, la gestión de reparaciones y la comunicación con los clientes.

- Evaluación de Impacto Ambiental:

Integrar módulos que evalúen el impacto ambiental de los siniestros y las reparaciones, promoviendo prácticas más sostenibles dentro del sector de seguros de vehículos. Esto podría incluir la recomendación de talleres que utilicen prácticas ecológicas o materiales reciclados.

Bibliografía

[1] Baptiste, D. (2018). Django 2 by Example: Build powerful and reliable Python web applications from scratch. Packt Publishing Ltd.

[2] Django Software Foundation. (n.d.). Django Documentation. Retrieved from https://docs.djangoproject.com/en/3.2/

[3] McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

[4] Red Hat. (n.d.). Drools Documentation. Retrieved from https://www.drools.org/documentation.html

[5] Sayfan, J. (2019). Mastering Python Networking: Your one-stop solution to using Python for network automation, DevOps, and Test-Driven Development, 2nd Edition. Packt Publishing Ltd.

[6] Apache Software Foundation. (n.d.). Maven: The Complete Reference. Retrieved from https://maven.apache.org/guides/

[7] Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.

[8] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

[9] MacCaw, A. (2011). JavaScript Web Applications: jQuery Developers' Guide to Moving State to the Client. O'Reilly Media.

[10] Collins, D. (2002). Designing Object-Oriented User Interfaces. Addison-Wesley Professional.

[11] Burns, M. (2020). Practical Python Projects: 19 Projects for Your Portfolio. Apress.

[12] Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.

Anexo I – Manuales

Introducción

Este manual está diseñado para guiar a los usuarios a través de la aplicación de gestión de siniestros de vehículos desarrollada con Django y Drools. La aplicación está disponible en el repositorio de GitHub: Aplicacion_TFG. A continuación, se detallan los pasos necesarios para la descarga, instalación y uso de la aplicación.

Descarga e Instalación

- Clonar el Repositorio

```
git clone https://github.com/gonzalopaul/Aplicacion_TFG.git
cd Aplicacion_TFG
```

- Ejecución del Docker:

La aplicación está configurada para ejecutarse mediante Docker. Para iniciar la aplicación, ejecute el siguiente comando:

```
docker-compose up --build
```

- Acceso a la Aplicación:

Una vez que el Docker esté en funcionamiento, puede acceder a la aplicación desde su navegador web en la siguiente URL:

http://localhost:8000

Registro y Login

- Registro de Usuario:

Para comenzar a utilizar la aplicación, debe registrarse como nuevo usuario. Complete el formulario de registro con su información personal.

- Login de Usuario:

Si ya tiene una cuenta, puede iniciar sesión utilizando sus credenciales. Tenga en cuenta que los nuevos usuarios no podrán utilizar la aplicación hasta que un administrador los verifique y les asigne un rol.

- Cuenta de Administrador:

Existe una cuenta de administrador preconfigurada:

```
Email: admin@autoaid.com

Contraseña: Junio2001
```

Roles y Funcionalidades

La aplicación distingue entre tres tipos de usuarios: Trabajador (Worker), Asegurado (Usuario) y Administrador. Dependiendo del rol asignado, los usuarios tendrán acceso a diferentes funcionalidades.

Funcionalidades del Trabajador (Worker)

Dashboard:

- Filtrado de Incidencias: Los trabajadores pueden filtrar incidencias generadas por los usuarios. Las incidencias que cumplen con los criterios de filtro se mostrarán en una tabla.
- Detalles de Incidencia: Al hacer clic en una incidencia, se puede ver un resumen detallado y tres opciones:
    - Enviar Datos: Envía la incidencia a Drools para su análisis.
    - Ver Análisis: Muestra las respuestas y recomendaciones de Drools. También permite cambiar el estado de la incidencia a 'Pendiente', 'En revisión' o 'Cerrado'.

- Eliminar Incidencia: Permite eliminar la incidencia seleccionada.

Historial de Usuarios:

- Filtrado por Usuario: Seleccione un usuario y haga clic en 'Filtrar' para ver su historial de incidencias y fraudes.
- Importación de CSV: Importar el historial de incidencias y fraudes mediante archivos CSV. Los CSV deben seguir un formato específico.

Póliza:

- Editar Póliza: Permite editar los detalles de la póliza de un asegurado, incluyendo tipo de póliza, fechas de inicio y fin, edad del conductor, años de carnet, puntos del carnet y el importe de la póliza.

Funcionalidades del Asegurado (Usuario)

Nueva Incidencia:

- Crear Incidencia: Rellene los campos solicitados para crear una nueva incidencia. Utilice el modelo 3D del vehículo para indicar daños específicos y adjuntar imágenes.

Mis Incidencias:

- Ver Incidencias: Observe el estado de sus incidencias. Una vez que un trabajador cierre una incidencia, puede descargar un PDF con los detalles de la misma.

Funcionalidades del Administrador

Permisos de Usuarios:

- Gestión de Roles: Verifique y asigne roles a los usuarios. Los administradores pueden seleccionar un usuario y cambiar su rol a 'Worker', 'Usuario' o 'Administrador'.

Perfil de Usuario

Todos los usuarios pueden:

- Modificar Perfil: Actualizar su información personal.
- Eliminar Cuenta: Eliminar su cuenta de manera permanente.

Drools

Drools es un motor de reglas que se utiliza para automatizar el análisis de incidencias y la detección de fraudes.

Estructura del Objeto Siniestro:

```
public class Siniestro {

    private LocalDateTime fechaYHora;

    private String ubicacion;

    private String tipoAparcamiento;

    private String llamadaPolicia;

    private String matricula;

    private String numeroIncidenciaPolicial;

    private String testigo;

    private String contactoTestigo;

    private String vehiculosPropiedadesInvolucrados;

    private int reincidenciaUltimoAnno;

    private int reincidencia3Anno;

    private int fraudes;

    private String tipoCobertura;

    private LocalDate inicioPoliza;

    private LocalDate finPoliza;

    private int edadConductor;

    private int anosCarnet;

    private double importePoliza;

    private String usoVehiculo;

    private int puntosCarnet;

    private String extrasPoliza;
```

```
}
```

Reglas de Drools:

- Las reglas se definen en el archivo siniestroRules.drl en el directorio main/resources/java/com/autoaid/drools/. Este archivo debe ser editado directamente para modificar las reglas aplicadas.

Añadir Nuevos Modelos 3D

Los nuevos modelos 3D de vehículos se pueden añadir en el directorio vehicle_visualization/templates. Configure los puntos interactivos (hotspots) y su funcionalidad según sea necesario.

Creación de Superusuarios

Para crear superusuarios en la aplicación de Django, utilice el siguiente comando en la terminal:

```
python manage.py createsuperuser
```

Anexo II – Reglas Drools

A continuación, se procede a explicar las reglas que hay por defecto en el archivo .drl del proyecto Drools.

Introducción

La elección de las reglas implementadas en Drools para la gestión de siniestros vehiculares se basa en la necesidad de optimizar el proceso de tramitación de siniestros, garantizar la precisión en la evaluación de los casos y mejorar la eficiencia operativa de la aseguradora. Cada regla ha sido cuidadosamente diseñada para abordar aspectos críticos de la tramitación de siniestros, desde la verificación de la vigencia de las pólizas hasta la identificación de posibles fraudes.

Reglas Seleccionadas y Su Funcionalidad

Fecha de Accidente Superior a 90 Días

- Justificación: Esta regla ayuda a identificar siniestros que fueron notificados mucho tiempo después de ocurridos, lo cual puede ser un indicativo de intentos de fraude o de situaciones que requieren una verificación adicional.
- Funcionalidad para la Empresa: Permite a la aseguradora priorizar la revisión de siniestros reportados tardíamente y tomar decisiones informadas sobre su validación.

Confirmar que el Siniestro Ocurrió en un Aparcamiento

- Justificación: La ubicación del siniestro puede influir significativamente en la evaluación del mismo. Siniestros ocurridos en aparcamientos pueden tener condiciones específicas.
- Funcionalidad para la Empresa: Facilita la identificación de siniestros con características particulares, permitiendo una evaluación más precisa y específica.

Testigos Presentes

- Justificación: La presencia de testigos puede ser crucial para la verificación de los detalles del siniestro.
- Funcionalidad para la Empresa: Proporciona información adicional para la validación de los siniestros, lo que puede reducir el tiempo de resolución y aumentar la precisión en la evaluación de los casos.

Reclamaciones Frecuentes

- Justificación: Un alto número de reclamaciones en un corto período puede ser un indicativo de comportamiento fraudulento o de problemas recurrentes.
- Funcionalidad para la Empresa: Permite a la aseguradora detectar patrones sospechosos y realizar investigaciones adicionales para prevenir fraudes.

Llamada a Policía en Siniestros Graves

- Justificación: En siniestros con daños significativos, es esperable que se haya llamado a la policía. La ausencia de un reporte policial puede ser sospechosa.
- Funcionalidad para la Empresa: Ayuda a identificar siniestros que requieren una revisión adicional debido a la falta de reporte policial, mejorando la precisión en la tramitación.

Fraude por Reincidencia

- Justificación: La reincidencia en siniestros puede ser un fuerte indicativo de fraude.
- Funcionalidad para la Empresa: Facilita la detección de fraudes al identificar asegurados con un alto número de siniestros en un período determinado.

Discrepancias en Reporte de Daño

- Justificación: Inconsistencias en el reporte de daños sin testigos o reporte policial pueden indicar la necesidad de una revisión más exhaustiva.
- Funcionalidad para la Empresa: Mejora la identificación de siniestros que requieren una verificación adicional, reduciendo el riesgo de aprobar reclamaciones fraudulentas.

Siniestros Frecuentes en el Mismo Vehículo

- Justificación: Múltiples siniestros reportados para el mismo vehículo pueden indicar problemas con el vehículo o intentos de fraude.
- Funcionalidad para la Empresa: Permite a la aseguradora investigar posibles problemas recurrentes con un vehículo y tomar decisiones informadas sobre la cobertura.

Priorizar Siniestros con Heridos

- Justificación: Los siniestros con posibles heridos requieren atención prioritaria para garantizar que se toman las medidas necesarias.
- Funcionalidad para la Empresa: Asegura que los casos más críticos reciban atención inmediata, mejorando la respuesta y el servicio al cliente.

Revisión Rápida para Siniestros Menores

- Justificación: Los siniestros sin daños significativos pueden ser procesados más rápidamente, liberando recursos para casos más complejos.
- Funcionalidad para la Empresa: Mejora la eficiencia operativa al permitir una tramitación rápida de siniestros menores, reduciendo el tiempo de resolución.

Pérdida de Puntos

- Justificación: La pérdida de puntos puede ser un indicador de la gravedad del siniestro y de la conducta del conductor.
- Funcionalidad para la Empresa: Facilita la evaluación de la gravedad del siniestro y permite tomar decisiones sobre la cobertura y las acciones necesarias.

Estado de Póliza

- Justificación: Verificar la vigencia de la póliza es esencial para determinar la cobertura del siniestro.
- Funcionalidad para la Empresa: Asegura que solo se tramiten siniestros que estén cubiertos por pólizas vigentes, evitando errores y fraudes.

Siniestro con Daños a Propiedades

- Justificación: Los daños a propiedades pueden incrementar la complejidad y el costo del siniestro.
- Funcionalidad para la Empresa: Permite una evaluación precisa y completa de los daños, mejorando la gestión de reclamaciones y la comunicación con los propietarios afectados.

Siniestro Durante la Vigencia de la Póliza

- Justificación: Es crucial verificar que el siniestro ocurrió durante la vigencia de la póliza para determinar la elegibilidad de la reclamación.
- Funcionalidad para la Empresa: Garantiza que solo se procesen siniestros que ocurren dentro del período de cobertura, protegiendo a la aseguradora contra reclamaciones inválidas.

Validar Información del Conductor

- Justificación: La información del conductor, como la edad y los años con carnet, es fundamental para evaluar la responsabilidad y la cobertura.
- Funcionalidad para la Empresa: Mejora la precisión en la evaluación de siniestros al asegurar que se cumplan los criterios de elegibilidad del conductor.

Comprobar Cobertura de la Póliza

- Justificación: Diferentes tipos de cobertura pueden afectar la evaluación y tramitación del siniestro.
- Funcionalidad para la Empresa: Permite una evaluación precisa y equitativa de los siniestros en función de la cobertura contratada por el asegurado.

Evaluar el Importe de la Póliza

- Justificación: Un importe bajo de la póliza puede requerir una revisión más detallada del siniestro.
- Funcionalidad para la Empresa: Ayuda a identificar siniestros que pueden necesitar una evaluación adicional debido a la baja cobertura contratada.

Conclusión

Estas reglas de Drools han sido seleccionadas y diseñadas para cubrir una amplia gama de situaciones y aspectos críticos en la tramitación de siniestros vehiculares. Su implementación no solo mejora la precisión y eficiencia de la evaluación de siniestros, sino que también proporciona una estructura robusta para la detección de fraudes, la priorización de casos y la mejora del servicio al cliente. Al adoptar estas reglas, la aseguradora puede optimizar sus operaciones, reducir riesgos y garantizar un proceso de tramitación de siniestros más efectivo y confiable. En el caso de que se quieran añadir nuevas reglas, se hará editando el archivo .drl. Habrá que tener en cuenta el formato de los datos que se mandan desde la aplicación web, y cómo los recibe Drool

Anexo III – Capacidad y rendimiento

La prueba de carga realizada en la aplicación web utilizando Apache JMeter, configurada para simular 10 solicitudes por segundo en el endpoint de inicio de sesión, produjo los siguientes resultados, los cuales se detallan a continuación:

Desglose de los Resultados

- # Muestras (Samples): 1792

Este valor indica el número total de solicitudes que se realizaron durante la prueba. En este caso, se realizaron un total de 1792 solicitudes de inicio de sesión.

- Media (Average): 465 ms

El tiempo promedio de respuesta para las solicitudes fue de 465 milisegundos. Este valor es crucial para entender el tiempo que toma el servidor en procesar una solicitud de inicio de sesión y responder.

- Mín (Min): 28 ms

El tiempo de respuesta más rápido registrado fue de 28 milisegundos, lo que sugiere que algunas solicitudes fueron procesadas de manera extremadamente eficiente.

- Máx (Max): 892 ms

El tiempo de respuesta más lento registrado fue de 892 milisegundos. Este valor es importante para identificar posibles cuellos de botella o momentos en los que el servidor experimentó una carga elevada.

- Desv. Estándar (Std. Dev.): 103.05 ms

La desviación estándar de 103.05 milisegundos indica la variabilidad en los tiempos de respuesta. Una desviación estándar más baja sugeriría tiempos de respuesta más consistentes.

- % Error: 0.56%

Este valor indica el porcentaje de solicitudes que resultaron en error. Un porcentaje de error de 0.56% es relativamente bajo, pero aún es importante investigar las causas de estos errores para mejorar la estabilidad de la aplicación.

- Rendimiento (Throughput): 21.3/sec

El rendimiento de 21.3 solicitudes por segundo supera las 10 solicitudes por segundo configuradas inicialmente, indicando que el servidor pudo manejar una carga mayor sin degradación significativa del rendimiento.

- Kb/sec: 126.95

Esta métrica mide la cantidad de datos transferidos por segundo. En este caso, se transfirieron 126.95 kilobytes por segundo.

- Sent KB/sec: 4.26

Esta métrica mide la cantidad de datos enviados por el cliente por segundo, que en este caso fue de 4.26 kilobytes por segundo.

- Media de Bytes (Avg. Bytes): 6093.0

El tamaño promedio de las respuestas fue de 6093 bytes.

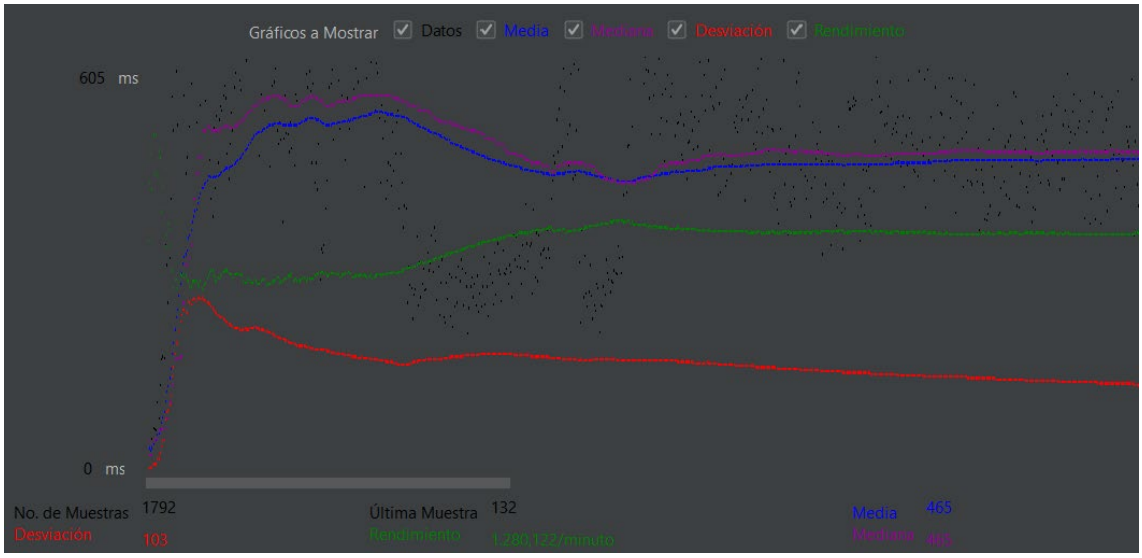


Ilustración 25: Gráfica rendimiento 10 peticiones / segundo

Ilustración 26: Gráfica rendimiento 20 peticiones / segundo

Al comparar los resultados de las pruebas de carga realizadas con 10 y 20 solicitudes por segundo, se observan las siguientes diferencias clave:

Resultados con 10 Solicitudes por Segundo:

Tiempo Promedio de Respuesta: 465 ms

% de Errores: 0.56%

Rendimiento: 21.3 solicitudes/segundo

Resultados con 20 Solicitudes por Segundo:

Tiempo Promedio de Respuesta: 861 ms

% de Errores: 1.84%

Rendimiento: 23.0 solicitudes/segundo

Análisis Comparativo

Al incrementar la carga de 10 a 20 solicitudes por segundo, el tiempo promedio de respuesta casi se duplicó, pasando de 465 ms a 861 ms, indicando una mayor latencia en el procesamiento de solicitudes bajo mayor carga. Además, el porcentaje de errores también aumentó de 0.56% a 1.84%, sugiriendo que la aplicación comienza a mostrar más fallos a medida que se incrementa la carga. Sin embargo, el rendimiento en términos de solicitudes por segundo mostró un ligero incremento de 21.3 a 23.0 solicitudes por segundo, indicando que el sistema aún puede manejar una carga mayor, aunque con una mayor tasa de error y tiempos de respuesta más elevados.