Seguros Axarquía

GESTOR DE SEGUROS

1. Estudio del problema y análisis del sistema.

1.1 Introducción.

Los corredores de seguros y administradores de fincas enfrentan una **gestión ineficiente** debido al uso de Excel para almacenar información crítica y la **dispersión de** la comunicación a través de **canales** como email o WhatsApp. Esta fragmentación provoca **pérdida de datos** y **demoras** en la atención de siniestros. La ausencia de una plataforma centralizada dificulta la coordinación y reduce la calidad del servicio al cliente.

1.1.1 Descripción del proyecto.

El proyecto consiste en el desarrollo de una plataforma web destinada a la gestión de seguros de edificios y comunidades de propietarios. **El sistema centraliza la gestión de las pólizas, desde su creación y administración hasta el registro y seguimiento de siniestros,** incorporando además un **chat en tiempo real** dentro de cada póliza y siniestro, para facilitar la interacción entre clientes, administradores de fincas y corredores o compañía aseguradora.

1.1.2 Propuesta de valor para el usuario.

En primer lugar, mejora significativamente la eficiencia al **eliminar el uso de herramientas y canales dispersos**. Con todo el proceso centralizado, se evita la pérdida de información y se reduce el tiempo invertido en tareas repetitivas.

En segundo lugar, **ofrece un control y visibilidad totales** sobre el estado de las pólizas y los siniestros, con accesibilidad desde cualquier dispositivo con conexión a internet. Esta característica facilita el **seguimiento continuo** y la toma de decisiones basadas en información actualizada.

En tercer lugar, la plataforma incorpora un sistema de **mensajería instantánea** que permite una comunicación fluida, directa y registrada entre los clientes, los administradores de fincas y los corredores de seguros, lo que incrementa la calidad del servicio y reduce tiempos de espera.

1.2 Funciones y rendimientos deseados

1.2.1 Qué queremos conseguir con el sistema a implementar.

El principal objetivo funcional del sistema es **automatizar el proceso de administración de pólizas y siniestros**, facilitando una gestión clara, estructurada y accesible desde una plataforma web. Se espera que el sistema reduzca considerablemente el número de

incidencias mal gestionadas y las pérdidas de información que actualmente se producen por la dispersión de medios.

Asimismo, se pretende ofrecer una **herramienta útil** tanto para los administradores de fincas como para los corredores de seguros, que permita consultar el estado de las pólizas, registrar nuevos siniestros y hacer un seguimiento en tiempo real de cada caso. El sistema debe ser capaz de soportar múltiples comunidades y clientes a la vez.

1.3 Análisis del mercado y plan de marketing:

1.3.1 Segmento de mercado al que va dirigido el proyecto.

El sistema está especialmente **orientado a corredores** de seguros y **aseguradores** que gestionan múltiples propiedades y pólizas. El perfil incluye a administradores de fincas que necesitan una herramienta confiable para **gestionar los seguros** de varias comunidades y a corredores de seguros que actúan como **intermediarios entre las aseguradoras** y los **administradores o propietarios**. También se contempla a **las propias comunidades** de propietarios como beneficiarios indirectos, ya que obtendrán una mejor atención y tiempos de respuesta más rápidos ante siniestros. La plataforma se adapta tanto a **pequeñas empresas como a medianas** estructuras que manejan un gran volumen de datos y necesitan controlarlos con precisión.

1.3.2 Análisis de la competencia del sector.

Actualmente, existen diversas soluciones tecnológicas en el mercado que cubren parcialmente las necesidades de este tipo de gestión, como **CRMs genéricos o herramientas de gestión documental.** Sin embargo, estas aplicaciones no están diseñadas específicamente para el entorno de seguros de comunidades, por lo que carecen de funcionalidades especializadas como el registro detallado de siniestros vinculados a pólizas concretas o la interacción directa con múltiples partes desde la misma plataforma.

También existen **soluciones específicas** que generalmente pertenecen a compañías que se dedican al sector por lo que ellos hacen la gestión de los seguros quitando gran parte del trabajo **pero a cambio el cliente solo se lleva una porción mínima de los seguros**.

Muchas empresas aún dependen de **herramientas como Excel y Access combinadas con canales de comunicación como WhatsApp y correo electrónico,** lo que dificulta enormemente la trazabilidad y el seguimiento eficiente de los casos. El sistema propuesto llena este vacío ofreciendo una solución especializada, integrada y accesible.

1.3.3 Estrategias de captación y fidelización de usuarios.

Se parte de un cliente colaborador al que esta versión demo es gratuita para poder disponer de una fase de pruebas y mejoras potenciales para ir puliendo el proyecto. Este cliente tendrá beneficios a futuro para la incorporación de nuevos módulos.

La estrategia de captación de usuarios se basa en ofrecer esta **versión demo funcional** que permita a los potenciales clientes experimentar las ventajas del sistema antes de su adquisición. Una vez satisfechos con la demo se pasaría a la fase inicial, en la cual se imparten **sesiones de formación** online y **soporte técnico** personalizado que facilite la adopción tecnológica.

Para la fidelización de usuarios, se implementará un sistema de **actualizaciones continuas** basadas en el **feedback de los clientes**, así como planes de **suscripción flexibles** y escalables que se adapten al tamaño y necesidades de cada cliente. La atención al cliente será un valor diferencial, ofreciendo canales de soporte rápidos y eficaces.

1.4 Objetivos

1.4.1 Qué servicios ofrecerá el sistema una vez implementado.

El sistema permitirá la gestión completa del ciclo de vida de una póliza, desde su registro inicial hasta su cancelación o modificación. Incluirá un módulo para registrar siniestros asociados a pólizas específicas, permitiendo agregar documentación, establecer estados de tramitación y realizar seguimiento. Incluirá además un chat en tiempo real para facilitar la comunicación entre las partes implicadas, permitiendo mantener conversaciones organizadas por expediente y notificando automáticamente los cambios de estado relevantes.

1.5 Planteamiento y evaluación de diversas soluciones

Durante la etapa de análisis se consideraron diferentes alternativas para abordar el problema identificado. Entre ellas se valoró la posibilidad de continuar utilizando herramientas como **Excel y Access**, integradas con canales de comunicación convencionales como el correo electrónico. Sin embargo, esta opción fue descartada debido a la **falta de escalabilidad**, la alta probabilidad de pérdida de datos y la imposibilidad de mantener una trazabilidad fiable.

Otra alternativa fue el uso de **CRMs genéricos o plataformas SaaS existentes**, pero la mayoría **carece de la personalización** necesaria para ajustarse al flujo específico del proceso de gestión de seguros de comunidades.

Finalmente, se optó por una **solución desarrollada a medida** que permita adaptar completamente la funcionalidad a las necesidades del cliente, garantizando control, eficiencia y un entorno único de trabajo.

1.6 Justificación de la solución elegida

Tras un análisis detallado de las alternativas disponibles y considerando las necesidades específicas del cliente, se ha optado por desarrollar la plataforma utilizando **Laravel 12** como framework backend, acompañado de su **starter kit con React.js** para el frontend, e implementando WebSockets para la comunicación en tiempo real.

Esta decisión se fundamenta en los siguientes aspectos:

- **Laravel 12** proporciona una arquitectura **robusta, segura y escalable** para gestionar las operaciones del lado del servidor. Su ecosistema maduro, extensa documentación y **amplia comunidad** ofrecen un entorno de desarrollo eficiente, con una curva de aprendizaje asumible para el equipo. Además, **incluye herramientas** integradas para autenticación, validación, gestión de sesiones y bases de datos, lo que reduce significativamente el tiempo de desarrollo.
- **React.js**, incluido en el starter kit oficial de Laravel, permite crear **interfaces** de usuario **dinámicas, reactivas y altamente adaptables a distintos dispositivos**. Esto garantiza una experiencia de usuario moderna, fluida e intuitiva.
- **WebSockets** se integra como la solución tecnológica para habilitar **comunicación bidireccional y en tiempo real entre los usuarios y el servidor**, elemento clave para funcionalidades como notificaciones instantáneas o chats en vivo.
- **MySQL/MariaDB** ha sido seleccionado como sistema de gestión de bases de datos relacional por su **fiabilidad, flexibilidad, rendimiento comprobado y compatibilidad** total con el stack tecnológico adoptado.

El starter kit de React para Laravel 12 incluye rutas, controladores y vistas necesarias para la autenticación de usuarios, lo cual acelera significativamente la configuración inicial del proyecto.

Este kit integra tecnologías actuales como **Inertia.js 2, React 19, TypeScript, Tailwind CSS y la biblioteca de componentes shadcn/ui.** Inertia permite desarrollar aplicaciones SPA (Single Page Application) utilizando el enrutamiento y los controladores clásicos del backend de Laravel. Esto combina lo mejor de ambos mundos: la productividad y estructura de Laravel con la potencia y experiencia de usuario que ofrece React, junto con la compilación rápida y eficiente proporcionada por Vite.

En conjunto, esta solución permite lograr un equilibrio entre rendimiento, escalabilidad, mantenibilidad y experiencia de usuario, alineándose con los objetivos del proyecto.

Considerando que el desarrollo fue realizado por una única persona, partiendo desde cero en el conocimiento de Laravel y con un plazo limitado de tres meses, esta opción representó

la alternativa más realista, eficiente y viable para alcanzar los objetivos planteados dentro del tiempo disponible.

1.6.1 Análisis DAFO-CAME.

El análisis **DAFO (Debilidades, Amenazas, Fortalezas y Oportunidades)** permite identificar los factores internos y externos que influyen en el éxito del proyecto. A partir de este diagnóstico, la matriz **CAME (Corregir, Afrontar, Mantener, Explotar)** propone estrategias orientadas a la mejora continua, priorizando la toma de decisiones en función del entorno y los recursos disponibles.

1.6.1.1 Análisis DAFO

Fortalezas

La plataforma está diseñada específicamente para un **mercado poco atendido**, lo que permite reducir la competencia directa y enfocarse en necesidades concretas.

Su **interfaz es simple e intuitiva**, facilitando el uso incluso para usuarios sin experiencia técnica.

Además, incorpora funciones clave como la **comunicación en tiempo real** mediante WebSockets, lo que mejora la interacción entre los usuarios y agiliza los procesos.

Gracias al contacto directo del **cliente** se obtiene el **conocimiento** de este sector para hacer más fácil el desarrollo. Con la ventaja que al acabar esta será revisada y testeada por el mismo cliente para hacer **cambios y mejoras**. Esto nos proporciona un producto de **calidad** y la posibilidad de un **desarrollo más rápido**.

Debilidades

Se cuenta con **recursos humanos y técnicos limitados**, ya que el proyecto fue realizado por una sola persona en un periodo corto de tiempo.

Además, al **no** tener **experiencia previa** con Laravel 12 y React, fue necesario apoyarse en documentación externa para avanzar en el desarrollo.

También se identificó la necesidad de **adquirir conocimientos** específicos del sector asegurador para comprender mejor los requisitos funcionales de la plataforma.

Se prevé que los clientes se vean **reacios al cambio** de las herramientas actuales y su adaptación al uso de una aplicación.

Oportunidades

Cada vez **más empresas** están empezando a **usar herramientas digitales** para hacer su trabajo más fácil y rápido, esto abre la oportunidad de adaptar la plataforma a otras áreas similares, como la gestión de documentos o distintos tipos de seguros.

Como muchas **soluciones actuales son antiguas** o poco prácticas, hay una buena oportunidad para destacar con una propuesta más moderna y eficiente.

Nuestra aplicación nos permite **adaptar y hacer cambios** en el código para los diferentes clientes añadiendo nuevas funcionalidades o un diseño adaptado a la marca del cliente.

Amenazas

Existen riesgos asociados a la posible aparición de **competidores con mayor** capacidad de **inversión** en desarrollo y marketing, lo que podría dificultar la posición en el mercado.

También se considera la **resistencia al cambio** por parte de usuarios acostumbrados a realizar sus tareas mediante procesos manuales, lo que puede **ralentizar** la adopción del sistema.

Por último, la **rápida evolución tecnológica** representa una amenaza, ya que la plataforma podría quedar obsoleta si no se actualiza de forma continua o se implementan herramientas de IA(Inteligencia Artificial) que ha día de hoy está en auge.

1.6.1.2 Matriz CAME

Mantener (Fortalezas)

Es fundamental **seguir optimizando la interfaz** para asegurar una experiencia de usuario simple, clara y accesible para todo tipo de perfiles.

Del mismo modo, resulta necesario mantener la aplicación de **buenas prácticas de desarrollo**, incluyendo el uso de patrones, que permiten una estructura del código organizada y fácil de mantener, por lo que **seguir adquiriendo conocimientos** sólidos de los frameworks utilizados es necesario.

Además, **la mejora continua** del código es clave para garantizar un buen rendimiento, facilitar **futuras ampliaciones** y asegurar la calidad y sostenibilidad del proyecto a largo plazo.

Corregir (Debilidades)

Se propone implementar una **guía de usuario** junto con sesiones de **formación básica** para facilitar que los usuarios adopten el sistema con mayor facilidad.

Además, es importante **documentar internamente** los procesos técnicos y funcionales para disminuir la dependencia de fuentes externas.

Por último, **automatizar tareas repetitivas** permitirá aprovechar mejor los recursos humanos limitados disponibles.

Explotar (Oportunidades)

Es recomendable desarrollar **campañas de marketing** enfocadas en promover la digitalización de procesos tradicionales en los sectores de seguros, destacando los beneficios de modernizar sus operaciones y aumentar la eficiencia.

Asimismo, es fundamental **explorar integraciones con otros sistemas utilizados en el sector**, como APIs de aseguradoras o soluciones basadas en inteligencia artificial, lo que ampliará considerablemente el alcance y la funcionalidad de la plataforma.

Por otro lado, ampliar el uso de la plataforma a otras áreas similares dentro del mismo sector representa una oportunidad estratégica de crecimiento con una inversión mínima, aprovechando la base tecnológica ya construida; por ejemplo, mediante la incorporación de un **gestor de presupuestos integrado**.

Afrontar (Amenazas)

Es fundamental diseñar una **propuesta de valor clara**, enfocada en la facilidad de uso y en resolver las necesidades específicas del nicho al que se dirige la plataforma.

También se debe recopilar **feedback de los primeros usuarios** para aplicar mejoras que refuercen su satisfacción y fidelidad.

Además, **planificar un esquema de actualizaciones periódicas** garantizará que la solución se mantenga vigente frente a los cambios tecnológicos y las nuevas demandas del mercado.

1.7 Modelado de la solución.

Contempla los recursos necesarios para llevar a cabo el desarrollo, implementación y puesta en marcha del sistema. Estos se dividen en **recursos humanos, hardware y software**, los cuales han sido seleccionados considerando la viabilidad técnica, económica y temporal del proyecto.

1.7.1 Recursos humanos.

El desarrollo fue asumido por **una sola persona con perfil full-stack**, encargada de todas las etapas: análisis, diseño, programación, pruebas y documentación. Aunque esto supuso un **reto** en cuanto a **carga de trabajo**, permitió un **control completo del proceso** y una integración coherente de todas las partes del sistema.

1.7.2 Recursos hardware.

Para el desarrollo y pruebas del sistema se utilizó un equipo de gama media con las siguientes características mínimas:

- Procesador Intel i5 o equivalente
- 8 GB de memoria RAM
- Disco SSD de 256 GB
- Conexión estable a internet

1.7.3 Recursos software.

Los recursos software utilizados incluyen tanto herramientas de desarrollo como frameworks y servicios necesarios para el funcionamiento del sistema:

- **Backend: Composer** y **Laravel 12** (PHP), que proporciona una arquitectura sólida, segura y con gran soporte de comunidad.
- **Frontend: Node.js** y **React.js**, integrados mediante **Inertia.js** y **Vite**, lo que permite construir una aplicación de una sola página (SPA) moderna, sin necesidad de una API REST convencional.
- **Base de datos: MySQL** o **MariaDB** (a través de **XAMPP** para el entorno de desarrollo), por su fiabilidad, rendimiento y amplia compatibilidad.
- **Diseño UI: Tailwind CSS** como sistema de estilos utilitario, junto con **Lucide** (iconos) y la biblioteca de componentes **shadcn/ui** para una interfaz limpia y coherente.
- **Control de versiones: Git** como sistema de control de versiones, con repositorio alojado en **GitHub** para la gestión del código y el trabajo colaborativo.
- **Entorno de desarrollo: Visual Studio Code**, por su ligereza, extensibilidad y compatibilidad con tecnologías web modernas.
- **Otras herramientas:**
    - **Google Chrome,** para pruebas de compatibilidad y herramientas de desarrollo integradas.
    - **Draw.io,** para la creación de diagramas técnicos y de flujo.
    - **Laravel Broadcast,** junto con **Reverb, Pusher y Echo,** para implementar comunicación en tiempo real mediante WebSockets, esencial para funcionalidades interactivas dentro de la plataforma.

1.8 Business Model Canvas: visión global del proyecto.

El Business Model Canvas es una herramienta visual que permite diseñar, analizar y mejorar modelos de negocio de forma sencilla y estructurada. Facilita identificar los elementos clave de un proyecto, como socios, actividades, recursos, clientes, canales y fuentes de ingresos. Ayudando a entender cómo una empresa crea, entrega y captura valor.

<table>
<tr>
<td rowspan="2">Aliados Clave

Correduría de seguros.
Agentes de seguros.
Compañías de seguros.
Administradores de fincas.
Gestores de siniestros.</td>
<td>Actividades Clave

Gestión de seguros completa con seguimiento de pólizas y siniestros.</td>
<td rowspan="2">Propuesta de Valor

Nadie ofrece un programa de autogestión de comunidades y siniestros sin comisión. La idea es ofrecer una demo y formación gratuita y vean la ventaja para ellos y sus clientes.

Así independientemente de los seguros que tengan solo pagarán una suscripción mensual.</td>
<td>Relación con el Cliente

A través de redes sociales boca a boca, telefónicamente y gestión directa.</td>
<td rowspan="2">Segmentos de Clientes

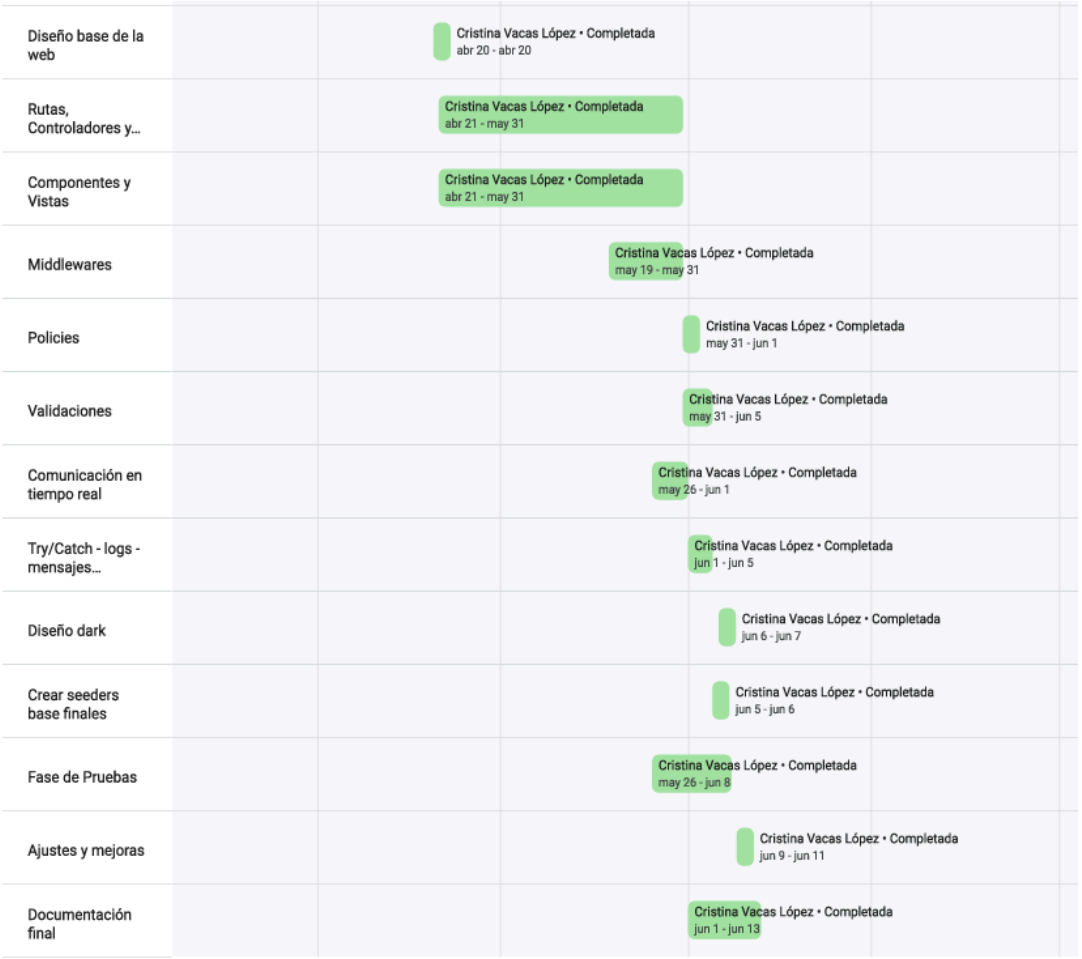
El cliente principal es corredor y administrador de fincas.</td>
</tr>
<tr>
<td>Recursos Clave
Servidor.
Página web.
Ordenadores.
Dispositivos móviles.
Programador Full Stack.
Gestor de siniestros.</td>
<td>Canales
La página web, la venta directa y la visita personal a los administradores. También telefónica y por correo electrónico.</td>
</tr>
<tr>
<td colspan="2">Estructura de Costes
Los gastos más importantes son equipo informático y desarrolladores.</td>
<td colspan="3">Estructura de Ingresos
La forma de obtener los ingresos será a través de las diferentes suscripciones.</td>
</tr>
</table>

Ver Anexo I

1.9 Planificación temporal revisable.

La planificación temporal del proyecto se ha estructurado considerando un plazo de tres meses, durante el cual una sola persona ha llevado a cabo todas las fases del desarrollo. Este cronograma es flexible y puede ajustarse según las necesidades o cambios en el alcance del proyecto. La herramienta principal utilizada para organizar las tareas ha sido el diagrama de Gantt, que permite visualizar de forma clara la duración, el orden y la dependencia entre actividades.

1.9.1 Diagrama de Gantt



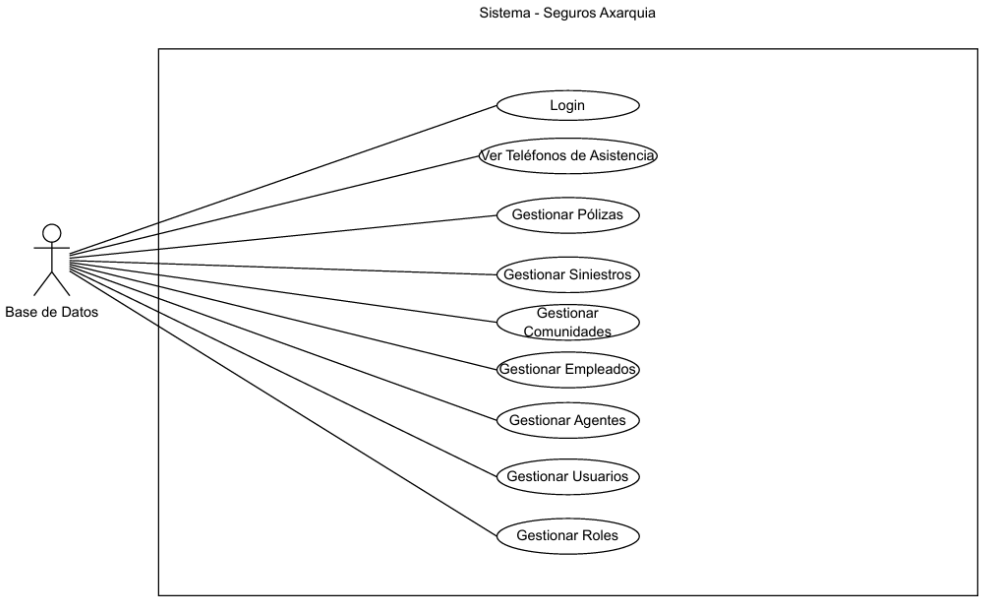
Ver Anexo II

2. Ejecución del proyecto.

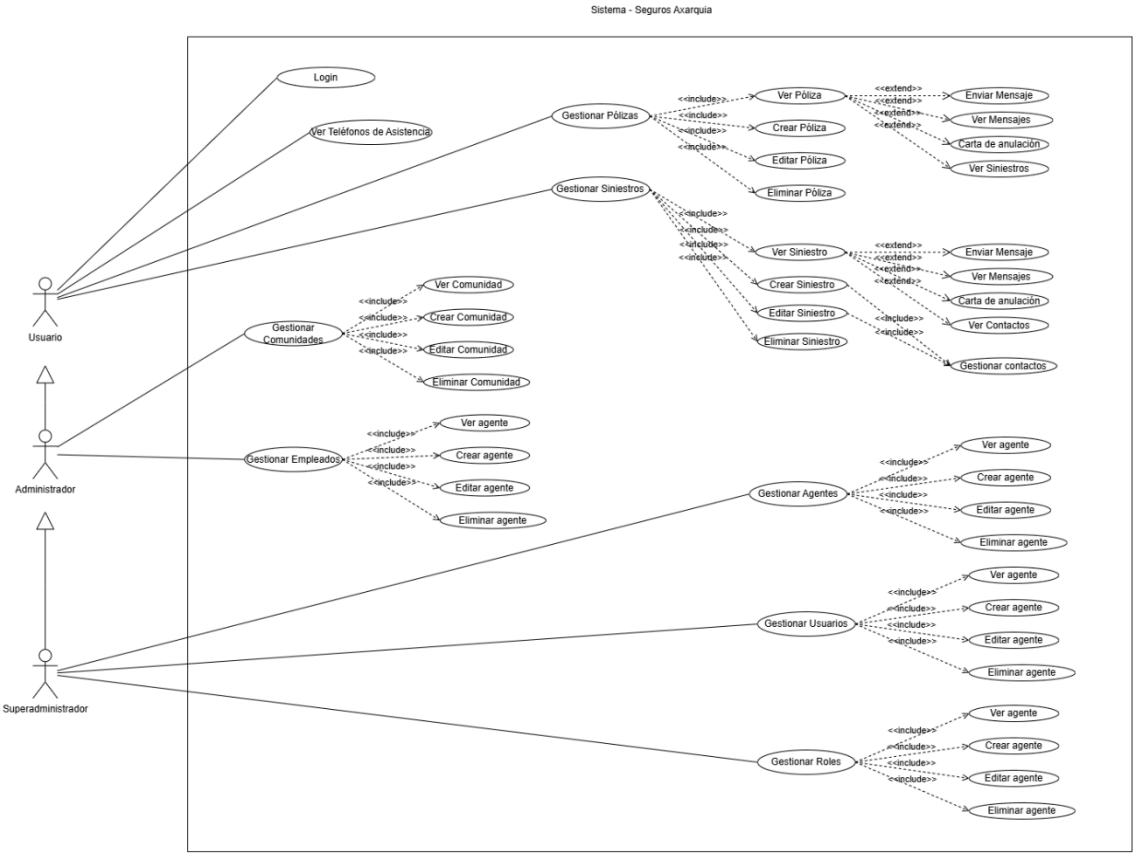
2.1 Elaboración de la documentación técnica.

2.1.1 Casos de uso.

Los casos de uso describen cómo los usuarios interactúan con un sistema para lograr un objetivo específico. En este caso tenemos 3 perfiles de usuario base, nuestra aplicación nos permite que el superadministrador genere nuevos perfiles, pero nos centraremos en estos 3 base. Además en cada caso de uso tenemos la conexión de la base de datos.



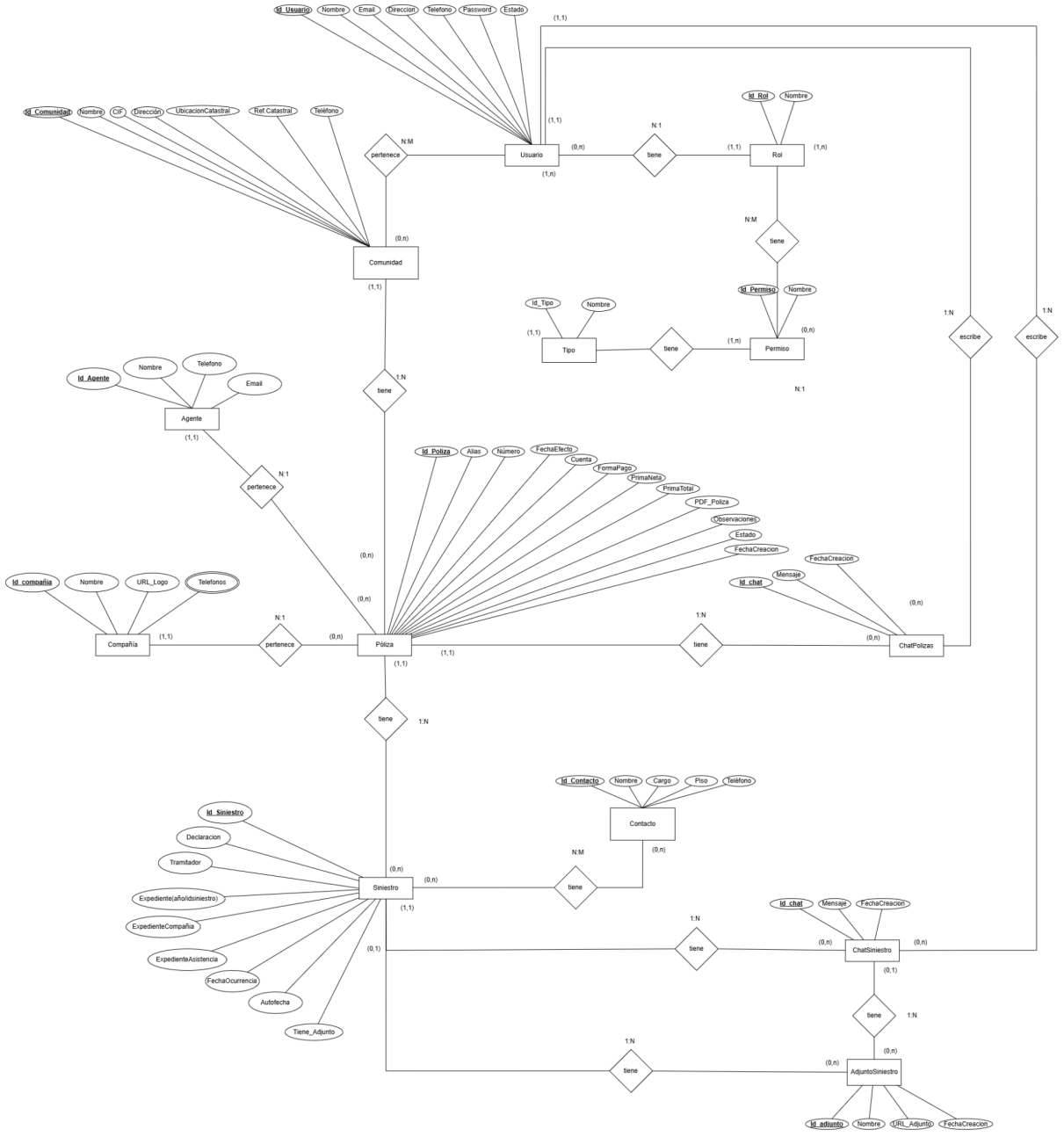
Ver Anexo III



Ver Anexo III

2.1.2 Modelo E/R(Entidad/Relación).

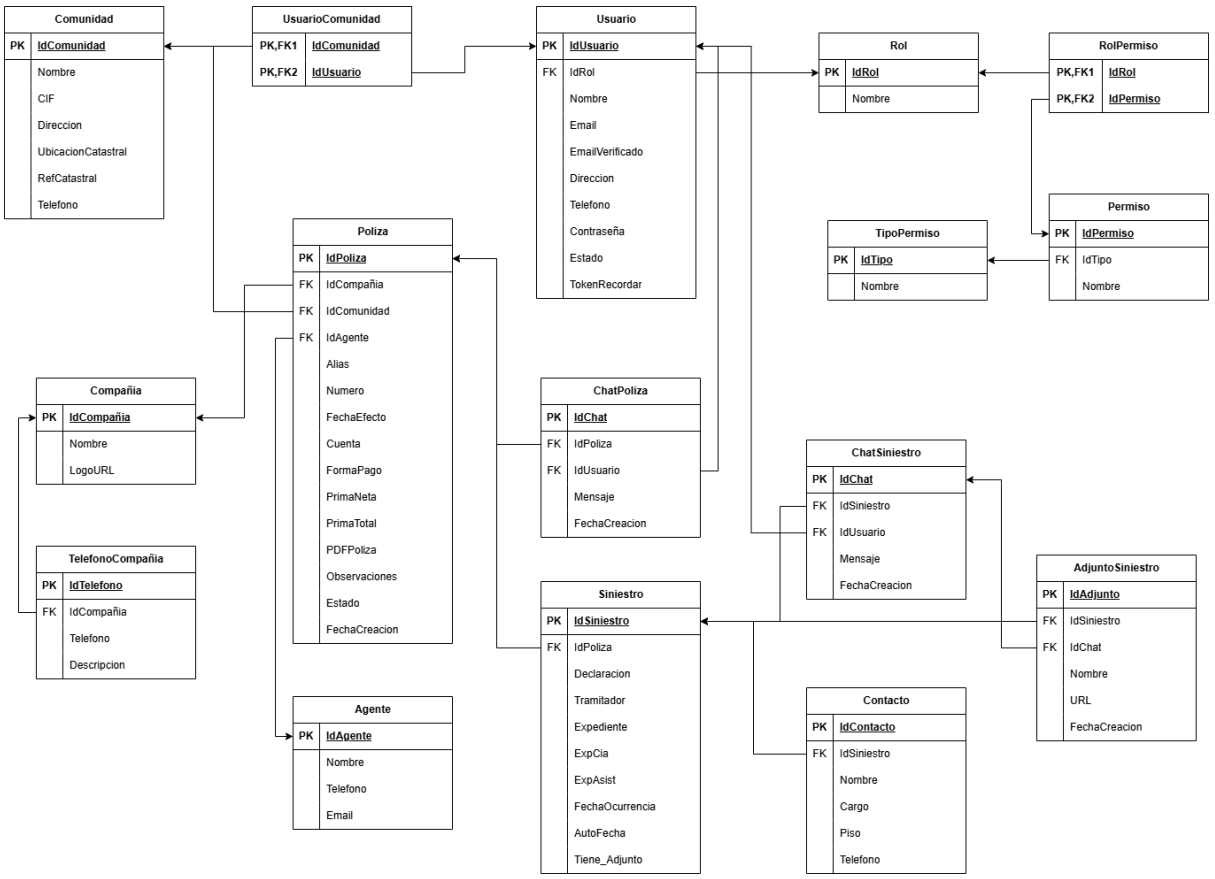
El modelo E/R permite representar gráficamente la estructura lógica de una base de datos antes de su implementación. Utiliza entidades, atributos y relaciones para describir cómo se conectan los distintos elementos del sistema. A continuación, se muestra el modelo entidad-relación diseñado para este proyecto.



Ver Anexo IV

2.1.3 Modelo Relacional.

El modelo relacional es la base teórica más utilizada para el diseño de bases de datos. Organiza la información en tablas que se relacionan entre sí mediante claves, facilitando el acceso, la integridad y la consistencia de los datos. A continuación, se presenta el modelo relacional propuesto para este sistema.



Ver Anexo V

2.1.4 Arquitectura del proyecto.

El proyecto sigue una arquitectura moderna que combina Laravel 12 como backend y React como frontend, utilizando Inertia.js como puente entre ambos.

Backend (Laravel)

```
seguros-app/
├── app/
│   ├── Http/
│   │   ├── Controllers/    # Controladores de la aplicación
│   │   └── Middleware/     # Middlewares personalizados
│   ├── Models/             # Modelos Eloquent
│   ├── Events/             # Eventos del sistema
│   ├── Policies/           # Políticas de autorización
```

```
│   └── Providers/      # Proveedores de servicios
├── database/
│   ├── migrations/     # Migraciones de base de datos
│   ├── factories/      # Factories para testing
│   └── seeders/        # Seeders para datos iniciales
└── routes/
    ├── web.php         # Rutas web principales
    ├── auth.php        # Rutas de autenticación
    ├── settings.php    # Rutas de configuración
    └── channels.php    # Canales de websocket
```

Laravel sigue un flujo **muy claro y estructurado**:
Ruta → Controlador → Modelo → Vista → Respuesta.

Cuando un **usuario accede a una URL** en tu aplicación Laravel, la primera parada es el archivo de **rutas**. Las rutas definen qué acción tomar dependiendo de la URL y del **método** HTTP (GET, POST, etc.). Estas rutas se definen en archivos como **routes/web.php** para rutas web (normalmente HTML) o routes/api.php para APIs. Aquí, puedes definir si al acceder a una URL como /usuarios, Laravel debe ejecutar un controlador, retornar una vista, o responder con JSON.

Desde la ruta, generalmente se delega la **ejecución a un controlador**, una clase que organiza la **lógica**. Un controlador puede obtener datos desde la base de datos, validar peticiones, autorizar al usuario y decidir qué responder. Laravel promueve separar la lógica en **métodos como index(), store(), update()**, etc., donde cada uno se encarga de una acción específica. Por ejemplo, UsuarioController@index puede encargarse de listar todos los usuarios.

Los controladores suelen apoyarse en los modelos para acceder a la base de datos. Laravel utiliza **Eloquent como su ORM** (Object-Relational Mapping), lo cual permite trabajar con la base de datos como si manipulamos objetos PHP. Cada **modelo representa una tabla** y puede **contener métodos para relaciones** (como hasMany, belongsTo, etc.). Por ejemplo, el modelo Usuario puede representar la tabla usuarios, y puedes usar métodos como Usuario::all() para recuperar registros.

Una vez que el controlador tiene los datos, decide qué **respuesta enviar**. Si usas Laravel clásico, puedes devolver una **vista Blade** (return view('usuarios.index')) para renderizar HTML del lado del servidor. Sin embargo, si usas Inertia.js con React, el controlador devuelve una **página Inertia** (return Inertia::render(...)), que en realidad carga un componente React en el navegador y le **pasa los datos como props**. De esta forma, el frontend se maneja con React pero **sin necesidad de una API separada**.

Durante este flujo, Laravel puede **aplicar validaciones** (para asegurarse de que los datos son correctos), **políticas** (para controlar permisos de acceso a ciertas acciones), y **middlewares** (filtros que interceptan las peticiones antes o después del controlador). Por

ejemplo, puedes tener un middleware que asegure que el usuario esté autenticado antes de permitirle ver una ruta.

Finalmente, después de procesar la petición, Laravel **genera una respuesta** que puede ser una vista HTML (usando Blade o Inertia + React), un JSON (en caso de una API), una redirección a otra URL (por ejemplo, después de guardar un formulario), o una respuesta de error si algo falla.

Frontend (React + Inertia)

```
seguros-app/
├── resources/
│   ├── js/
│   │   ├── pages/          # Páginas de React (componentes principales)
│   │   ├── components/     # Componentes reutilizables
│   │   ├── layouts/        # Layouts de la aplicación
│   │   ├── hooks/          # Custom hooks de React
│   │   ├── types/          # Definiciones de TypeScript
│   │   ├── app.tsx         # Punto de entrada de React
│   │   └── ssr.tsx         # Configuración SSR
│   └── css/
│       └── app.css         # Estilos globales
```

Los componentes están organizados de la siguiente manera:

```
resources/js/
├── components/
│   ├── ui/             # Componentes de UI básicos
│   ├── usuarios/       # Componentes específicos de usuarios
│   ├── roles/          # Componentes de gestión de roles
│   └── siniestros/     # Componentes de siniestros
├── pages/
│   ├── dashboard/      # Páginas del panel de control
│   ├── usuarios/       # Páginas de gestión de usuarios
│   └── polizas/        # Páginas de gestión de pólizas
```

La **carpeta principal** para el código React se encuentra dentro de **resources/js/**. Ahí es donde viven todos los **componentes, páginas, layouts y la configuración de Inertia**. Las **páginas React completas** (por ejemplo, "CrearRol" o "EditarUsuario") se ubican en la **carpeta Pages/**. Estas páginas suelen coincidir con rutas definidas en Laravel, y cada una representa una vista completa. Por otro lado, los componentes reutilizables (como inputs personalizados, botones o modales) se agrupan dentro de **Components**/, y las estructuras generales de la interfaz, como menús o cabeceras, suelen estar en **Layouts/**.

La inicialización del frontend ocurre en app.tsx, donde se configura el cliente de Inertia, se monta React sobre un div dentro de la plantilla base Blade de Laravel, y se definen los

adaptadores para que Laravel y React se comuniquen. Al usar Inertia, Laravel no retorna HTML directamente, sino que retorna una respuesta JSON con los datos y el nombre del componente React que debe mostrarse. Inertia en el frontend recibe eso y renderiza la vista correspondiente sin recargar la página.

En las páginas de React, los datos enviados desde Laravel llegan como **props**, y suelen **manejarse con el hook useForm** del paquete @inertiajs/react. Este hook simplifica mucho el manejo de formularios, ya que **proporciona helpers** para el envío (post, put, etc.), el seguimiento de errores de validación, y el estado de procesamiento. También se puede usar **el hook usePage para acceder a datos globales** compartidos desde Laravel, como el usuario autenticado, flashes de sesión, o configuraciones globales.

La navegación entre páginas React se hace con el componente **Link** de Inertia, que reemplaza los enlaces tradicionales por navegación SPA sin recarga. Además, para gestionar el título del navegador o metadatos de la página, se usa el componente **Head**. Todo esto hace que la experiencia de usuario sea muy fluida y rápida, ya que no hay recargas completas, sino intercambios dinámicos entre Laravel y React.

2.1.5 Ficheros de configuración.

2.1.5.1 Archivo .env

A continuación se mencionan las variables utilizadas en este archivo y la utilidad de las mismas.

Configuración general de la aplicación

Variable	Definición
APP_NAME	Nombre de la aplicación.
APP_ENV	Entorno de ejecución (local, producción, etc.).
APP_KEY	Clave usada para cifrado (debe generarse para seguridad).
APP_DEBUG	Muestra errores detallados si está en true.
APP_URL	URL base de la aplicación.
APP_LOCALE	Idioma principal de la app.
APP_FALLBACK_LOCALE	Idioma alternativo si no hay traducción.
APP_FAKER_LOCALE	Localización usada por Faker para generar datos falsos.

APP_TIMEZONE	Zona horaria predeterminada.

Configuración del sistema

Variable	Definición
PHP_CLI_SERVER_WORKERS	Número de procesos que maneja el servidor CLI de PHP.
BCRYPT_ROUNDS	Nivel de complejidad del cifrado Bcrypt (más alto = más seguro, pero más lento).
LOG_CHANNEL	Canal de log principal.
LOG_STACK	Canal que agrupa múltiples canales de log.
LOG_DEPRECATIONS_CHANNEL	Canal de log para advertencias de funciones obsoletas.
LOG_LEVEL	Nivel de detalle del log (debug, info, error, etc.).

Base de Datos

Variable	Definición
DB_CONNECTION	Motor de base de datos (ej. mariadb, mysql, etc.).
DB_HOST	Dirección del servidor de base de datos.
DB_PORT	Puerto usado por la base de datos.
DB_DATABASE	Nombre de la base de datos.
DB_USERNAME	Usuario de la base de datos.
DB_PASSWORD	Contraseña del usuario de la base de datos.

Sesiones

Variable	Definición
SESSION_DRIVER	Motor de sesiones (en este caso, database).
SESSION_LIFETIME	Duración en minutos de la sesión.

SESSION_ENCRYPT	Indica si los datos de sesión están cifrados.
SESSION_PATH	Ruta donde es válida la cookie de sesión.
SESSION_DOMAIN	Dominio para la cookie de sesión (null usa el actual).

Broadcast y colas

Variable	Definición
BROADCAST_DRIVER	Sistema de difusión en tiempo real (log, reverb, etc.).
FILESYSTEM_DISK	Disco por defecto para el sistema de archivos.
QUEUE_CONNECTION	Sistema para gestionar colas de trabajo.
CACHE_STORE	Motor de almacenamiento en caché.

Correo

Variable	Definición
MAIL_MAILER	Protocolo de envío de correos (smtp, sendmail, etc.).
MAIL_HOST	Servidor SMTP usado para enviar correos.
MAIL_PORT	Puerto del servidor SMTP.
MAIL_USERNAME	Usuario de la cuenta de correo (tu email).
MAIL_PASSWORD	Contraseña del correo.
MAIL_ENCRYPTION	Encriptación usada (ej. tls).
MAIL_FROM_ADDRESS	Dirección de correo desde la cual se envían los mensajes.
MAIL_FROM_NAME	Nombre que se muestra como remitente (usa el nombre de la app).

Reverb (broadcast en tiempo real)

Variable	Definición
REVERB_APP_ID	ID de la aplicación Reverb.

REVERB_APP_KEY	Clave pública para conexión.
REVERB_APP_SECRET	Clave secreta para autenticación.
REVERB_HOST	Dirección del servidor Reverb.
REVERB_PORT	Puerto del servidor Reverb.
REVERB_SCHEME	Protocolo (http o https).

Variable	**Definición (usadas en frontend con Vite)**
VITE_APP_NAME	Nombre de la app para uso en frontend.
VITE_REVERB_APP_KEY	Clave Reverb accesible en frontend.
VITE_REVERB_HOST	Host Reverb accesible en frontend.
VITE_REVERB_PORT	Puerto Reverb accesible en frontend.
VITE_REVERB_SCHEME	Protocolo para conexión desde el frontend.

2.1.5.2 Carpeta config

La carpeta **config** en un proyecto de Laravel contiene archivos de configuración que definen parámetros clave para el funcionamiento de la aplicación. Estos archivos permiten separar la lógica del código de los ajustes específicos del entorno, facilitando la mantenibilidad y escalabilidad del sistema.

Archivo	**Explicación**
app.php	Configura aspectos generales de la aplicación: nombre, entorno (local, producción), zona horaria, proveedores y alias.
auth.php	Define la configuración de autenticación, como los "guards", proveedores de usuarios y tiempos de expiración de sesiones o tokens.
broadcasting.php	Configura el broadcasting de eventos en tiempo real (WebSockets), incluyendo drivers como Pusher, Redis o log.
cache.php	Establece cómo y dónde se almacenará la caché de la aplicación (archivo, base de datos, Redis, etc.).

database.php	Define las conexiones a bases de datos (MySQL, SQLite, PostgreSQL, etc.) y parámetros relacionados como host, puerto y credenciales.
filesystems.php	Configura los sistemas de archivos utilizados para almacenar archivos, como local, public, Amazon S3, etc.
inertia.php	Configura opciones específicas para el uso de Inertia.js, que permite construir aplicaciones SPA con Laravel y frameworks frontend.
logging.php	Controla cómo y dónde se guardan los logs de la aplicación: canal por defecto, niveles de log, destinos (archivo, Slack, etc.).
mail.php	Configura el envío de correos electrónicos, como el driver (SMTP, Mailgun, etc.), host, puerto, remitente por defecto.
queue.php	Define la configuración de colas de trabajos (jobs), incluyendo el sistema usado (sync, database, Redis) y tiempos de retención.
reverb.php	Configuración para Laravel Reverb, el servidor WebSocket nativo de Laravel para eventos en tiempo real.
services.php	Guarda credenciales y configuraciones para servicios externos como Stripe, AWS, Mailgun, etc.
session.php	Controla el almacenamiento y duración de sesiones de usuario: driver (archivo, base de datos, Redis), tiempo de vida, etc.

Estos archivos recogen los valores de configuración del .env generalmente por lo que no es necesario modificarlos.

2.1.6 Características técnicas del sistema.

El sistema está desarrollado para ser accesible desde cualquier navegador, que centraliza la gestión de pólizas, siniestros, comunidades, agentes, usuarios y comunicaciones en tiempo real mediante chat. Su arquitectura es modular y escalable, permitiendo adaptarse a diferentes entornos de despliegue (local, servidor dedicado o en la nube).

Requisitos de instalación

Software

- **Servidor Web**: Apache 2.4+ o Nginx 1.18+.
- **Lenguaje Backend**: PHP 8.1+ (Framework Laravel).
- **Base de Datos**: MySQL 8.0+ o PostgreSQL 13+ (cualquier base de datos relacional)
- **Sistema Operativo**: Linux (Ubuntu 20.04+ recomendado), aunque también compatible con Windows Server
- **Node.js**: v18+ (para frontend, assets y WebSockets)
- **Composer**: para gestión de dependencias PHP
- **Broadcasting / WebSockets**: Laravel Reverb
- **Correo**: Configuración SMTP para envío automático de notificaciones (Google, Microsoft…)

Hardware mínimo (para servidor en producción)

- **CPU**: 4 núcleos (2.4 GHz o superior)
- **RAM**: 8 GB mínimo (16 GB recomendado para alto volumen de usuarios y siniestros)
- **Almacenamiento**: 100 GB SSD (escalable según la cantidad de documentación gestionada)
- **Conectividad**: Ancho de banda estable con buena latencia si se usa en la nube o accedido remotamente

Usuario final

- **Navegador compatible**: Google Chrome, Mozilla Firefox, Microsoft Edge (últimas versiones)
- **Dispositivo**: PC, portátil, tablet o móvil con acceso a internet

3. Fase de pruebas.

En la fase de pruebas se testean las validaciones de cada formulario, se chequean los accesos con cada permiso y rol (middlewares y policies), las rutas junto con cada acción del programa…

3.1 Creación de una batería de pruebas.

En el Anexo VI podemos ver los casos de prueba realizados para determinar el correcto funcionamiento del programa y poder corregir errores que se pasaron por alto. Se han testeado pruebas funcionales, de validación así como de seguridad.

Cada prueba está asociada a un **grupo** que corresponde una **sección del programa** y tiene un **actor o varios** para saber con que usuario realizar la prueba que se describe, en caso de tener varios, se realizará la prueba con cada tipo actor.

El **resultado esperado** puede ser positivo o negativo, este dato es el que tenemos que comprobar al realizar la prueba, en caso de obtener un mismo resultado la prueba tendrá el estado de Correcto, si no concuerda con el resultado esperado el estado será Por arreglar y en caso de no haber realizado la prueba será Pendiente.

ID	Tipo de Prueba	Grupo	Descripcion	Actor	Ejemplo	Observaciones	Resultado esperado	Estado
CP1	Funcional	Login	Login con usuario y contraseña correctos	Cualquiera			Positivo	Correcto
CP2	Funcional	Login	Login con usuario incorrecto	Cualquiera			Negativo	Correcto
CP3	Funcional	Login	Login con contraseña incorrecta	Cualquiera			Negativo	Correcto
CP4	Funcional	Login	Recuperar contraseña mediante correo existente	Cualquiera			Positivo	Correcto
CP5	Funcional	Login	Recuperar contraseña mediante correo inexistente	Cualquiera			Negativo	Correcto
CP6	Funcional	Login	Login con usuario innactivo	Cualquiera			Negativo	Correcto
CP7	Seguridad	Acceso	Acceso sin loguearse	Ninguno		Deberia redirigir al login	Negativo	Correcto
CP8	Validación	Usuarios	Campo de nombre con más de dos caracteres y menos de 255 carácteres	Superadministrador	Carlos	Este caso de prueba ha de repetirse en crear usuario y editar usuario	Positivo	Correcto
CP9	Validación	Usuarios	Campo de nombre con menos de dos caracteres o más de 255 caracteres	Superadministrador	C	Debería mostrar un error, este caso de prueba ha de repetirse en crear usuario y editar usuario	Negativo	Correcto
CP10	Validación	Usuarios	Campo de nombre vacio	Superadministrador		Debería mostrar un error, este caso de prueba ha de repetirse en crear usuario y editar usuario	Negativo	Correcto
CP11	Validación	Usuarios	Campo de email válido	Superadministrador	carlos.perez@example.com	Este caso de prueba ha de repetirse en crear usuario y editar usuario	Positivo	Correcto
CP12	Validación	Usuarios	Campo de email inválido, sin @, con espacios, con carácteres espaciales, sin dominio (.com), vacío, ya registrado	Superadministrador	carlos'.perez	Deberia mostrar un error, este caso de prueba ha de repetirse en crear usuario y editar usuario	Negativo	Correcto
CP13	Validación	Usuarios	Campo de contraseña con 8 caracteres mínimo, 1 mayúscula mínimo, 1 minúscula mínimo, 1 carácter especial mínimo, 1 número como mínimo	Superadministrador	Password-1	Este caso de prueba ha de repetirse en crear usuario y editar usuario	Positivo	Correcto

Ver Anexo VI

En la fase de pruebas **se encontraron ciertos errores** que se solucionaron con éxito:

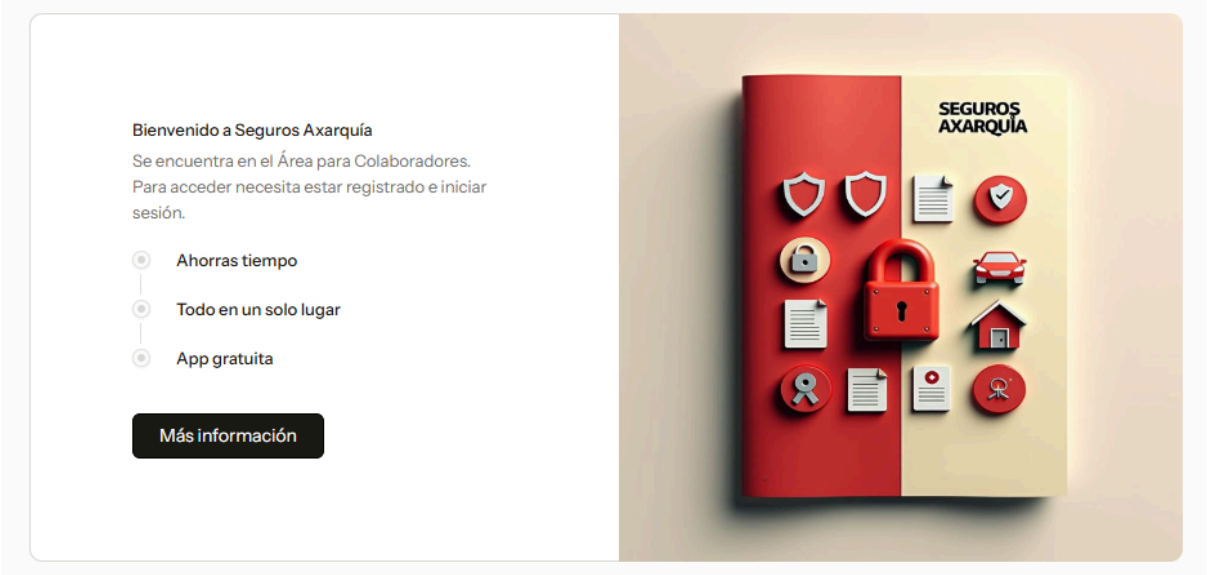
- Accesos a rutas inexistentes
- Acceso a empleados para el superadministrador
- Resultados de error diferentes, en algunos 403 y otros con redirección y notificación de error, en vez de la misma lógica en todo.
- Mostrar en crear/editar usuario en el campo ‘Es empleado de otro usuario? ‘, solo administradores principales.
- Mostrar en crear/editar empleados los roles menos el de superadministrador.
- Acceso de usuario inactivo.
- Funcionamiento correcto en caso de ser un subusuario administrador perteneciente a un administrador principal.
- En comunidades añadir el campo de propietario en la vista de superusuario para mostrar solo usuarios asociados a ese propietario.

4. Documentación del sistema.

4.1 Introducción a la aplicación.

La aplicación consiste en una plataforma web para la gestión de seguros que permita administrar pólizas, siniestros y mejorar la comunicación con los clientes a través de un sistema de mensajería en tiempo real.

El programa mejora la eficiencia en la administración de seguros, optimizando la gestión de información y permitiendo una mejor atención al cliente, al estar todo unificado en una sola herramienta online, expedientes por póliza y comunicación bidireccional.



4.2 Manual de Instalación (local)

4.2.1 Configuración Previa

Para el despliegue del proyecto en un entorno local (desarrollo o pruebas) necesitaremos lo siguiente:

Un servidor de base de datos, en mi caso uso XAMPP v.8.2.12
Composer v.2.8.4 para el sistema de Laravel
NodeJS v.22.12.0 para el sistema de React

4.2.2 Despliegue del Proyecto

4.2.2.1 Clona el repositorio:

git clone https://github.com/cvaclop1911/proyecto-daw.git

4.2.2.2 Copiar el archivo `.env.example` a `.env`

Configurar las variables base de nuestra aplicación como la base de datos, el nombre de la aplicación, el servidor de mail…

4.2.2.3 Instalación de las dependencias de PHP

composer global require laravel/installer
composer install

4.2.2.4 En caso de tener XAMPP puede que no nos deje instalar algunos paquetes.

Deberemos ir al archivo `php.ini`, buscar y descomentar la siguiente línea:
extension=zip

4.2.2.5 Crear una app-key para el archivo `.env`

php artisan key:generate

4.2.2.6 Crear la carpeta `node_modules` (en la raíz) y la carpeta `build` (dentro de public)

npm install
npm run dev

4.2.2.7 Ejecutar las migraciones y seeders

php artisan migrate --seed

4.2.2.8 Iniciar el servidor

npm run build
php artisan serve

4.2.2.9 Instalar broadcasting e iniciar reverb

php artisan install:broadcasting
php artisan reverb:start

4.3 Manual de instalación (servidor)

4.3.1 Configuración previa

- Servidor con Ubuntu 20.04+ (con acceso SSH)
- Dominio apuntando al servidor (opcional pero recomendable)
- Base de datos disponible
- Tener instalado: PHP, Composer, Node.js y npm, y Apache o Nginx

4.3.2 Despliegue de la aplicación

4.3.2.1 Clona el repositorio:

git clone https://github.com/cvaclop1911/proyecto-daw.git

4.3.2.2 Copiar el archivo `.env.example` a `.env`

Configurar las variables base de nuestra aplicación como la base de datos, el nombre de la aplicación, el servidor de mail…

4.3.2.3 Instalación de las dependencias de PHP

composer install

4.3.2.4 Crear una app-key para el archivo `.env`

php artisan key:generate

4.3.2.5 Crear la carpeta `node_modules` (en la raíz) y la carpeta `build` (dentro de public)

npm install
npm run build

4.3.2.6 Ejecutar las migraciones y seeders

php artisan migrate --seed

4.3.2.7 Configuración de Apache o Nginx

Tendremos que configurar un sites-available con nuestro dominio, carpeta, permisos que necesitamos…

4.3.2.8 Instalacion de Reverb y configuración

php artisan install:broadcasting
Después tendremos que seguir la configuración de producción de reverb para el servidor web: https://laravel.com/docs/12.x/reverb#production

4.4 Manual de usuario.

Ver en el Anexo VII

4.5 Manual de administrador.

Ver en el Anexo VIII

4.6 Manual de superadministrador.

Ver en el Anexo IX

5. Conclusiones finales.

5.1 Grado de cumplimiento de los objetivos fijados.

Los objetivos han sido más que satisfactorios. Se ha desarrollado una plataforma funcional que permite la gestión completa del ciclo de vida de una póliza, el registro y seguimiento de siniestros, y la comunicación directa mediante un sistema de chat en tiempo real entre los distintos usuarios implicados, lo cual era el objetivo principal.

En cuanto a los objetivos específicos:

- El sistema permite registrar, visualizar, modificar y asociar siniestros a pólizas específicas, con su documentación y estado de tramitación.
- Se ha integrado un módulo de chat basado en WebSockets (Laravel Reverb), permitiendo comunicación en tiempo real organizada por expediente.
- La aplicación incluye autenticación de usuarios, roles (superadministrador, agente, subusuarios), cifrado de contraseñas, protección CSRF y control de acceso a la información mediante middlewares y policies.
- El sistema ha sido desarrollado con Laravel 12 y React, usa Laravel Reverb para tiempo real, base de datos relacional, y está preparado para ser desplegado en entornos escalables.

Esto cumple todos los objetivos pactados en el anteproyecto.

Adicionalmente, se ha podido alcanzar otros objetivos no establecidos en el anteproyecto:

- El sistema permite la subida de archivos privados para las pólizas y siniestros, haciendo uso del storage de Laravel.
- Se incorporó la recuperación de contraseña en caso de pérdida, haciendo uso de envío por correo electrónico.
- Asimismo se puede enviar una carta de anulación de la póliza por correo electrónico al propietario de la póliza.

- Cuenta con notificaciones de confirmación de cada acción para el usuario y también un log de cada acción realizada para facilitarle al desarrollador encontrar errores futuros.

- Se ha podido adaptar el diseño con posibilidad de ver en modo claro y oscuro.

5.2 Propuesta de modificaciones o ampliaciones futuras del sistema implementado.

A pesar de cumplir todos los requisitos y objetivos definidos en el anteproyecto, aún se pueden hacer muchas mejoras o modificaciones:

- Permitir adjuntar archivos en cada chat y crear un módulo dentro de pólizas y siniestros llamado ‘Adjuntos’ donde podamos visualizar todos los adjuntos.

- Dentro de editar póliza y siniestro crear un diseño para los archivos que permite eliminar archivos concretos y añadir nuevos, actualmente si se modifica subes archivos y se borra los antiguos.

- Modificar el sistema de acceso a la póliza o siniestro, actualmente se asignan usuarios a una comunidad y se permite acceder a todas las pólizas o siniestros que pertenezcan a esa comunidad, pero se debe poder asignar usuarios a una póliza o siniestro en caso de tener un rol concreto.

- Incorporar sistema de herramientas de análisis y paneles de estadísticas que ayuden en la toma de decisiones estratégicas y operativas.

- Se prevé incorporar un nuevo módulo de proyectos en el que puedas generar presupuestos a medida en base a características base de las compañías y de las comunidades, según las coberturas y otra información.

- Asimismo el punto anterior se podría suplir con un agente de IA entrenado para el cual te genere un pdf con los presupuestos a medida.

- Sería interesante a través de websockets hacer un canal de notificaciones que muestre avisos como la llegada de un mensaje a un chat de póliza o siniestro, junto con notificaciones al correo electrónico de que ha habido un envío de mensaje a cierta póliza o siniestro para un mayor control de la actividad en la aplicación.

- Incorporar una vista de empleados para el superadministrador en la que pueda seleccionar un administrador principal y muestre los empleados de dicho administrador.

6. Bibliografía.

6.1 Libros, proyectos, manuales y bibliografía web utilizados.

Cursos para la formación de las nuevas tecnologías:

- Curso de Typescript (para el uso de typescript en react)
- Curso de Laravel Profesional (a pesar de estar enfocado en laravel 11, es un curso muy completo y bien enfocado)
- Curso de Websocket (curso para comprender la arquitectura y funcionamiento de un websocket)
- Curso de Reverb (ejemplo del uso de reverb en laravel 12)
- Fundamentos de React (react básico para crear componentes reactivos con useEffect y useState)
- Curso de Laravel (curso básico de laravel para el comienzo del desarrollo)

Recursos de Github:

- Laravel-React-Reverb
- React-Básico
- Proyecto básico de React

Documentación:

- Laravel Docs (documentación completa para laravel)
- React Learn
- Inertia Docs
- Tailwind Docs (consultar etiquetas)
- Shadcn Docs (componentes de react ya diseñados)
- Lucide Icons (iconos de libre acceso para react)

Recursos de IA:

- ChatGPT (consultas específicas de diseño o estructuración de código, ideas o alternativas para un buen uso de las herramientas de Laravel y React, generación de datos para pruebas, investigación de recursos…)
- Claude (fallos complejos de incompatibilidad de componentes)