UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

CoreSafe: Desarrollo Backend para la Plataforma del Programa de Registro Conductual (PRC) en Indemin.

Daniel Carrasco Pallaleo

daniel.carrascop@sansano.usm.cl

Óscar Carrasco Vera
Profesor Guía

Diego Cáceres Solís
Profesor Correferente

Resumen: En el contexto de la industria minera, la gestión del Programa de Registro Conductual (PRC) se realiza de forma manual, lo que provoca ineficiencias operativas, escasa trazabilidad de los datos, dificultades para el análisis de tendencias, lo que limita la capacidad de la organización para tomar acciones preventivas. Frente a esta problemática se propone CoreSafe, cuyo núcleo es un sistema Backend materializado en una API RESTful desarrollada con el framework Laravel. La solución se basa en una arquitectura monolítica con modularidad lógica, con mecanismos de autenticación basados en JWT, control de acceso con roles y permisos, y persistencia en base de datos relacional MySQL. La validación de la propuesta se realiza a través de pruebas manuales y funcionales de los endpoints. Como resultado, se espera digitalizar el PRC, mejorando la disponibilidad y confiabilidad de los datos, permitiendo una gestión de la seguridad basada en evidencia.

Palabras Clave: API REST, Laravel, Seguridad Basada en el Comportamiento, JWT.

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: CoreSafe: Desarrollo Backend para la Plataforma del Programa de Registro Conductual (PRC) en Indemin.

Nombre del candidato(a): Daniel Eduardo Carrasco Pallaleo

Carrera / Grado: Ingeniería en Informática

Campus: Viña del Mar **Departamento:** Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Óscar Francisco Carrasco Vera, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☑ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

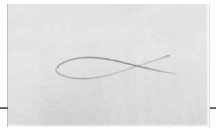
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 11-03-2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 11-03-2026 **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

1. Introducción.

1.1. Contexto y antecedentes.

El sector de la industria minera opera en un entorno de alta complejidad y con riesgos inherentes, donde la gestión de la seguridad y la salud de los trabajadores no es solo una cuestión ética y legal, sino que representa un pilar para la continuidad operativa y la sostenibilidad del negocio. En Chile, la Sociedad Nacional de Minería (SONAMI) registran [1] 61 fallecidos acumulados entre los años 2020 y 2024, evidenciando la necesidad de mejorar los mecanismos de control y gestión del riesgo.

En esta línea, los programas de Seguridad Basada en el Comportamiento (Behavior-Based Safety, BBS) se han consolidado como un estándar en la industria para la prevención de incidentes. Dichos programas se centran en la observación, el análisis y la modificación de las conductas de los trabajadores con el fin de reforzar prácticas seguras y corregir aquellas que podrían representar algún riesgo. Este enfoque es especialmente relevante en contextos donde los factores humanos y operacionales interactúan en entornos complejos.

La empresa Indemin, que es una organización dedicada a prestar servicios especializados en prevención de riesgos y seguridad operacional para la industria minera, ha desarrollado de manera interna el Programa de Registro Conductual (PRC), como adaptación de metodología BBS. Indemin ayuda en los procesos de distintas faenas y clientes del sector, implementando estrategias preventivas orientadas a reducir incidentes a través de la observación y el análisis conductual en terreno.

Actualmente, la ejecución del PRC es realizado como un proceso manual, donde supervisores y personal del área de prevención registran las observaciones usando planillas físicas o documentos digitales aislados, como hojas de cálculo. Seguidamente, esta información es transcrita de manera manual al final de las jornadas o del turno, lo que empieza a generar desfases entre la observación en terreno y la disponibilidad para su análisis, además de aumentar la probabilidad de errores de digitación, pérdida de registros y falta de estandarización de los datos.

La prevalencia de procesos manuales dificulta el seguimiento y la trazabilidad de la información, lo que repercute negativamente en la capacidad de la organización para tomar decisiones preventivas. La ausencia de una plataforma centralizada genera ineficiencias operativas y limita la capacidad de analizar tendencias, así como la detección de patrones de riesgo, impidiendo tomar acciones que se basan en evidencia sólida y oportuna.

1.2. Definición del problema.

El sistema actual de gestión del Programa de Registro Conductual (PRC) en Indemin, basado en registros manuales, evidencia una serie de deficiencias que limitan su eficacia y obstaculizan el potencial del programa para fomentar mejoras continuas en la seguridad de las operaciones. Estas limitaciones se pueden desglosar en los siguientes puntos clave:

- **Falta de Centralización de la Información**: Todos los datos que se registran en las observaciones se encuentran distribuidos en distintos formatos y ubicaciones que van desde los formularios físicos archivados, hasta hojas de cálculo en nubes o equipos locales. Esta dispersión impide consolidar una fuente única y confiable de información, dificultando la obtención de una visión integrada y en tiempo real del estado de la seguridad conductual dentro de la organización.
- **Limitada Trazabilidad y dificultad para el Análisis de Datos**: El formato no estructurado y descentralizado de los datos hace que el seguimiento de las conductas observadas de un trabajador o área de trabajo sea muy complejo. La elaboración de reportes y estadísticas, al ser un proceso manual, puede consumir varias horas por semana, además de ser susceptibles a errores humanos, restringiendo la capacidad para identificar patrones de riesgo y evaluar efectividad de las intervenciones de prevención.
- **Falta de validación e integridad de los datos registrados**: El uso de planillas físicas o documentos digitales sin una estructura relacional dificulta la aplicación de mecanismos formales de validación y control de consistencia. Esto genera registros incompletos, duplicados o asociados de forma incorrecta a trabajadores, áreas o faenas, demostrando una ausencia de integridad referencial que merma la confiabilidad de la información en la gestión preventiva.
- **Ineficiencia Operativa y Costos ocultos**: El equipo de prevención de riesgos y sus supervisores invierten una cantidad significativa de tiempo en tareas administrativas de escaso valor agregado, tales como la transcripción de las observaciones, la consolidación de planillas y la creación manual de informes. Este tiempo puede representar entre una y dos horas adicionales por turno, tiempo que podría ser reasignado a actividades de mayor impacto, como capacitaciones en terreno o en la implementación de mejoras en los procesos de prevención.
- **Demora en Toma de Decisiones**: La información crítica sobre riesgos latentes o tendencias conductuales no están disponible de manera oportuna para los niveles de gestión y gerencia. Esta latencia, que puede extenderse por varios días entre la observación en terreno y su consolidación final, reduce la capacidad de reacción de Indemin para implementar medidas correctivas y preventivas de forma oportuna.

1.3. Breve descripción sobre la propuesta de solución

Para abordar las deficiencias expuestas, se propone el desarrollo de CoreSafe, una plataforma digital diseñada para centralizar, estandarizar y gestionar las observaciones del Programa de Registro Conductual (PRC) de Indemin. El eje estructural de esta propuesta, y foco principal de la presente tesina, es el desarrollo de su sistema Backend, materializado en una API REST, construida con el framework Laravel y una base de datos relacional MySQL, incluyendo mecanismos de seguridad a través de autenticación basada en JSON Web Tokens (JWT).

Esta API constituirá el núcleo lógico y de procesamiento de la plataforma, siendo responsable de gestionar la lógica de negocio, asegurar la persistencia e integridad de los datos en una base de datos centralizada, y proporcionar un conjunto de servicios estandarizados para ser consumido por diversas aplicaciones cliente. Específicamente, el Backend organizará los módulos funcionales críticos como la gestión de usuarios, roles y permisos, la administración
de plantillas de los formularios, y el registro de las observaciones realizadas en terreno.

La arquitectura desacoplada es una decisión estratégica fundamental, ya que busca establecer una base tecnológica flexible, escalable y mantenible. La misma API será capaz de atender simultáneamente a múltiples clientes: una aplicación web (Vue.js) para administración, una aplicación móvil (Flutter) para el registro de las observaciones en terreno, un sistema de inteligencia de negocios (PowerBI) para alta gerencia, y como interfaz de integración con otros sistemas internos de Indemin.

Considerando que es un sistema orientado a la gestión preventiva, la solución añade control de seguridad que garantiza la confidencialidad, trazabilidad y acceso controlado a la información, con reglas de autorización basadas en roles y permisos.

Este enfoque, además de resolver la problemática del registro manual, también posiciona a la API como un activo tecnológico, con potencial para ser usada en futuras iniciativas de digitalización dentro de la organización.

Límites del Proyecto

En concordancia con las buenas prácticas de gestión de proyectos, y con el fin de mantener un alcance claro y viable, esta tesina se delimita de la siguiente manera:

- **Incluido en el desarrollo**:

- El diseño e implementación de la API REST utilizando el framework Laravel, incluyendo definición de endpoints, estructura de solicitudes/respuestas, validaciones y manejo de errores.
- La implementación del modelo de datos relacional para estructurar y almacenar la información del PRC.
- El desarrollo de un sistema de autenticación robusto basado en JSON Web Tokens (JWT) y un sistema de autorización granular por roles y permisos.
- La integración con servicios existentes de Indemin para la sincronización de datos críticos.

- **Fuera del alcance del desarrollo:**
  - El diseño y desarrollo de la aplicación Frontend web y la aplicación móvil, de quien su desarrollo está a cargo del desarrollador Frontend.
  - El diseño del modelo relacional de la base de datos y optimización de consultas, que su realización la hará el encargado de la Base de Datos.
  - El diseño de la arquitectura general de la plataforma CoreSafe, realizado por el encargado de arquitectura del Sistema.
  - El despliegue y configuración de infraestructura de producción (servidores, bases de datos, etc.).
  - La implementación de los dashboards de inteligencia de negocios, que su realización la hará la encargada de análisis de Datos.
  - La coordinación general y gestión operativa del proyecto CoreSafe, labor del Scrum Master.

1.4. Objetivos Generales y Específicos

Objetivo General

Desarrollar un sistema Backend seguro, modular y escalable a nivel horizontal, a través de una API RESTful implementada con el framework Laravel y una base de datos relacional MySQL, para la plataforma CoreSafe, permitiendo la gestión centralizada, sistemática y con un alto grado de confiabilidad de los datos del Programa de Registro conductual (PRC) de Indemin.

Objetivos Específicos

1. Diseñar e Implementar una base de datos relacional, a partir del modelo entidad-relación proporcionado, garantizando la integridad referencial, consistencia y la correcta interrelación de las distintas entidades clave del PRC.
2. Desarrollar un mecanismo de autenticación seguro y sin estado (stateless) utilizando el estándar JSON Web Tokens (JWT) para proteger el acceso a los recursos de la API.
3. Implementar y configurar un sistema de autorización basado en roles y permisos, para controlar de manera precisa las acciones que cada tipo de usuario puede ejecutar sobre los recursos del sistema.
4. Desarrollar y documentar un conjunto completo de endpoints de la API REST, para soportar las operaciones de creación, lectura, actualización y eliminación (CRUD) de las entidades del sistema, tal como Observaciones y Formularios.
5. Integrar módulos de comunicación necesarios para la sincronización de datos críticos entre la plataforma CoreSafe y los sistemas de Indemin.
6. Validar el funcionamiento del sistema con pruebas manuales con Postman y pruebas basadas en funcionalidades como Feature Tests utilizando PHPUnit, para asegurar el correcto comportamiento de los endpoints y la lógica de negocio implementada.

1.5. Justificación de proyecto

El presente proyecto reviste una alta relevancia técnica al aplicar las prácticas y arquitecturas de software modernas a un problema del mundo real: la gestión manual de registros conductuales. La adopción de una arquitectura basada en API REST favorece la modularidad, interoperabilidad y el desacoplamiento entre el servidor y los clientes, estableciendo las bases para un sistema flexible y escalable. Este diseño facilita la integración segura y estandarizada entre los componentes Frontend, la base de datos y otros servicios de Indemin, permitiendo además la gestión de múltiples solicitudes en tiempo real, en consonancia con las exigencias de un entorno operativo dinámico.

La selección de Laravel como framework de desarrollo se fundamenta en su ecosistema consolidado, su enfoque en la seguridad y su elevada productividad en entornos de desarrollo. Componentes como el ORM Eloquent y un sistema de enrutamiento aceleran la creación de APIs, mientras que sus mecanismos de autenticación contribuyen a la robustez del sistema. A su vez, el uso de estándares como HTTP, JSON y JWT ayuda a la mantenibilidad y escalabilidad del sistema, permitiendo a otros desarrolladores comprender, ampliar o integrar la solución sin depender de tecnologías propietarias o de otras arquitecturas. Aunque el proyecto utiliza autenticación basada en JWT, dicha decisión

no representa una limitación estructural, ya que el diseño modular permite adaptar el mecanismo de autenticación a las futuras reglas de negocio o las exigencias de Indemin.

Desde un punto de vista económico, la elección de tecnologías Open Source como Laravel, MySQL y PHPUnit se justifica por la reducción significativa de costos asociados a licencias de software propietario. La decisión permite a Indemin implementar una solución sin incurrir en gastos por licenciamiento, optimizando la inversión en tecnología. Además, el respaldo comunitario y documentación disponible reduce los costos de mantenimiento y capacitación asegurando independencia tecnológica.

Beneficios

La implementación del Backend CoreSafe aportará beneficios concretos y estratégicos para Indemin en distintos niveles:

- **Operacionales**: Se logrará la automatización integral del proceso conductual, eliminando la dependencia de soportes físicos o hojas de cálculo. Mejorando así la calidad, consistencia y disponibilidad de los datos. Esto permitirá liberar horas de trabajo del personal de prevención, para que puedan canalizar las tareas de mayor valor agregado.
- **Tácticos**: La centralización de los datos permitirá la generación de informes y análisis de tendencias de una manera más automatizada y en tiempo real, facilitando el acceso a supervisores y gerentes a dashboards con indicadores clave de desempeño (KPIs), identificando patrones de comportamiento de riesgo y apoyando la toma de decisiones informadas.
- **Estratégicos**: Se creará un activo de alto valor: una plataforma con datos históricos estructurados y centralizados sobre la seguridad conductual de la empresa. Estos datos podrán ser explorados con técnicas de análisis avanzado y modelos predictivos, ubicando a Indemin a la vanguardia en gestión de seguridad proactiva y basada en evidencia.

1.6. Metodología

Para el desarrollo de la plataforma CoreSafe se adoptó el enfoque ágil basado en Scrum, con el objetivo de gestionar el proyecto de manera incremental e iterativa, adaptándose a los requerimientos planteados por Indemin. Este enfoque facilitó la entrega progresiva de funcionalidades del Backend, asegurando la retroalimentación y alineación con las necesidades del cliente.

El proyecto se organizó en ciclos de trabajo de dos semanas (sprints). Al inicio de cada sprint se realizaba una reunión para definir las tareas más prioritarias, además de estimar el esfuerzo requerido y seleccionar los ítems del Product Backlog a desarrollar. Al terminar cada sprint, se realizaba una reunión con los representantes de Indemin para presentar los avances y así recopilar la retroalimentación técnica y funcional.

El Product Backlog fue gestionado mediante la herramienta de gestión de proyectos Jira, donde los requerimientos fueron registrados como historia de usuario bajo el formato: “Como [rol] quiero [acción] para [beneficio]”. Con dicho formato se mantuvo la trazabilidad entre los requerimientos funcionales y las tareas a implementar en el Backend.

En cuanto a los roles del equipo, se estableció una estructura alineada con Scrum: un integrante asumió el rol de Product Owner definiendo las prioridades y validando los entregables. Otro integrante desempeñó el rol de Scrum Master, facilitando la coordinación del equipo y el seguimiento del proyecto. Los demás integrantes asumieron sus roles más técnicos y específicos, siendo los encargados de la implementación de los distintos módulos del sistema.

Durante el desarrollo se realizaron reuniones internas de seguimiento, cuya frecuencia dependía de la disponibilidad del equipo con el objetivo de revisar el estado de las tareas, identificar bloqueos y coordinar el trabajo general. También, al término de los sprints se efectuaron reuniones de retrospección, para analizar las oportunidades de mejora y optimizar el flujo de desarrollo.

Desde el punto de vista técnico, la metodología ágil fue complementada con un flujo de trabajo basado en GitFlow. En este flujo cada tarea en el sprint se desarrolló en una rama independiente de tipo feature, que era validada localmente mediante pruebas manuales y revisión interna del equipo, la cual una vez validada se integraba a la rama de desarrollo (develop). Esto permitió mantener la estabilidad del código, facilitando revisiones y asegurando un control adecuado de versiones durante todas las fases del proyecto.

El enfoque iterativo permitió implementar la lógica de negocio de manera gradual, validando progresivamente los módulos más importantes como autenticación, autorización, gestión de los formularios y registro de observaciones, reduciendo riesgos e integrando de manera continua los componentes del sistema.

1.7. Organización del informe.

El presente documento se ha estructurado en cuatro capítulos, diseñados para guiar al lector desde la conceptualización del problema hasta la implementación de la solución y las conclusiones.

Capítulo 1 - Introducción: Este primer capítulo expone el marco contextual del proyecto, definiendo la problemática de la gestión manual de datos. Se presenta los objetivos generales y específicos, junto con la justificación técnica y la metodología de trabajo, además de describir la solución propuesta y establecer claridad en el alcance definido para el desarrollo de la tesina.

Capítulo 2 - Marco Teórico: Este capítulo se dedica a establecer el fundamento conceptual y técnico que sustenta el proyecto. Se tratan principios del desarrollo de software enfocados en el Backend, arquitectura REST y se realiza un análisis del estado del arte, justificando las tecnologías y herramientas utilizadas.

Capítulo 3 - Diseño e Implementación: Aquí se aborda el proceso de construcción del sistema Backend. Describiendo la arquitectura de software implementada, diseño de los componentes, estructura de la API, el modelo de datos y mecanismos de seguridad. Además de pruebas y validación funcional del sistema.

Capítulo 4 - Conclusiones: En este capítulo se presenta una reflexión sobre los resultados obtenidos, desafíos enfrentados, además de lecciones aprendidas durante el desarrollo de la plataforma. Se presentan los objetivos cumplidos y se plantean posibles opciones de mejoras para la plataforma CoreSafe.

2. Marco Teórico.

En esta sección del marco teórico se presentan los fundamentos conceptuales y técnicos que son la base del diseño y desarrollo del sistema CoreSafe. Se describe el rol del Backend dentro de la arquitectura, destacando las responsabilidades como: lógica de negocio, persistencia de datos, seguridad y comunicación entre componentes, así como la importancia en la operación general del sistema. Se abordan las arquitecturas más importantes como monolíticas, en capas, basadas en microservicios y serverless, junto con los lenguajes de programación, frameworks y patrones de diseño que se utilizan en el desarrollo Backend actual, justificando la elección en base a las necesidades del proyecto. Además, se agregan conceptos clave como APIs RESTful, mecanismos de autenticación y autorización y gestión de bases de datos, permitiendo entender el marco conceptual y las buenas prácticas.

2.1. Fundamentos del desarrollo Backend.

Dentro de la ingeniería de software encontramos la disciplina del desarrollo Backend, que es la encargada de construir y mantener la lógica del lado del servidor de una aplicación. El Backend es el responsable de procesar las solicitudes de los clientes, implementar la lógica de negocio, interactuar con la base de datos para almacenar y gestionar información, administrar la autenticación y autorización de usuarios, y asegurar el rendimiento y la escalabilidad del sistema. En arquitecturas desacopladas, como la adoptada por CoreSafe, el Backend es el núcleo funcional que centraliza la lógica y expone los servicios a los clientes.

Las APIs REST se basan en el protocolo HTTP (HyperText Transfer Protocol). Este protocolo funciona bajo un modelo de comunicación cliente-servidor basado en el ciclo Request/Response. Los principios arquitectónicos que sostienen a REST fueron formalizados por Roy Fielding en su tesis doctoral [2], donde se definen las restricciones que caracterizan este estilo de arquitectura. El modelo funciona con el cliente enviando la solicitud (request) al servidor, el cual contiene información como la URL del recurso, el método HTTP, encabezados y, de manera opcional una sección con datos. Posteriormente, el servidor procesa la solicitud, ejecuta la lógica y devuelve una respuesta (response) que incluye un código de estado HTTP, encabezados y un cuerpo de datos.

Una característica del protocolo HTTP es su naturaleza stateless (sin estado), lo que implica que cada petición es independiente y no mantiene información de las solicitudes anteriores. Esta propiedad incide directamente en el diseño de sistemas Backend más actuales, donde los mecanismos de autenticación, como JSON Web Tokens (JWT), permiten validar las solicitudes sin almacenar sesiones en el servidor, ayudando la escalabilidad horizontal.

HTTP define métodos que establecen la semántica de las operaciones sobre los recursos:

- GET: Obtener información.
- POST: Crear un nuevo recurso.
- PUT: Actualizar un recurso.
- DELETE: Eliminar un recurso.

Estos métodos forman la base de las operaciones CRUD (Create, Read, Update, Delete) implementadas en APIs REST como la desarrollada en CoreSafe.

A lo largo de la historia, las aplicaciones web se desarrollaban siguiendo arquitecturas monolíticas fuertemente acopladas, en donde el Frontend y Backend compartían una misma base de código. Con el crecimiento de varios tipos de clientes como navegadores, aplicaciones móviles o dispositivos IoT, surge la necesidad de separar responsabilidades para fomentar las arquitecturas desacopladas.

El enfoque API-first posiciona la API como núcleo del sistema, implementando los contratos de comunicación antes que las interfaces visuales. Con este paradigma, el Backend se expone como un servicio independiente que es consumido por distintos clientes tecnológicos. CoreSafe adopta este paradigma al desarrollar una API REST como centro de la plataforma, teniendo interoperabilidad entre las aplicaciones web y móviles.

En el desarrollo Backend, las solicitudes no llegan directamente al controlador que ejecuta la lógica de negocio, sino que atraviesan una serie de capas denominadas middlewares, que siguen el patrón de diseño llamado canalización (pipeline). Por lo tanto, un middleware es un componente que intercepta una solicitud antes de que ingrese al controlador, permitiendo ejecutar tareas como la validación de autenticación, verificación de permisos, registro de actividad o de manejo de errores. Esta arquitectura favorece la seguridad del sistema.

En aplicaciones modernas el acceso a las bases de datos relacionales se realiza mediante herramientas como el Object-Relational Mapping (ORM). El ORM permite hacer un mapeo de las tablas de una base de datos a objetos del lenguaje de programación, facilitando la manipulación de los datos sin necesidad de escribir consultas SQL explícitas.

El patrón utilizado por Laravel es el Active Record a través de su ORM llamado Eloquent. En este patrón cada objeto representa una fila de la base de datos e incluye los métodos para la persistencia y recuperación de datos. En Laravel, este enfoque permite interactuar con entidades del sistema como si fueran objetos, manteniendo la coherencia entre el modelo relacional y la lógica de la aplicación.

La interoperabilidad entre los sistemas desacoplados necesita de un formato de intercambio de datos estandarizado y ligero. En este contexto de las APIs REST, el formato predominante es el JSON (JavaScript Object Notation), debido a su simplicidad y bajo peso en comparación con otros formatos como XML. JSON facilita la comunicación entre el Backend y clientes, como aplicaciones desarrolladas en Vue.js o Flutter, permitiendo representar estructuras de datos de forma más eficiente y con alta compatibilidad entre plataformas.

Las tendencias actuales de la industria indican una evolución en dirección a las arquitecturas basadas en microservicios [3], el uso de contenedores como Docker y enfoques serverless que permiten escalabilidad en entornos cloud. Estas prácticas buscan mejorar la eficiencia y disponibilidad en sistemas distribuidos.

Asimismo, la integración de metodologías ágiles y prácticas DevOps, favorece ciclos de entrega rápidos y confiables, fomentando la integración continua y un despliegue automatizado.

En resumen, el desarrollo Backend es el núcleo a nivel funcional de las aplicaciones modernas, porque es el responsable de aplicar las reglas de negocio, garantizar la comunicación segura entre componentes y de procesar los datos de manera eficiente y escalable.

2.2. Arquitectura de software.

Arquitectura de comunicación REST (Representational State Transfer)

Se debe distinguir entre el estilo de comunicación y la estructura interna del sistema. Dentro del proyecto REST es utilizado como estilo arquitectónico de comunicación, definiendo cómo los distintos clientes interactúan con el Backend a través del protocolo HTTP, mientras que la organización del código y el despliegue del sistema corresponden a una arquitectura monolítica con modularidad lógica. En resumen, REST define la forma en que se exponen y consumen los servicios, mientras que el enfoque de arquitectura monolítica con modularidad lógica entrega la estructura del Backend de CoreSafe.

Para el Backend de CoreSafe, se seleccionó la arquitectura REST, que es un conjunto de principios y restricciones de diseño para servicios distribuidos y escalables, permitiendo que la API sea predecible, interoperable y capaz de aprovechar las virtudes del protocolo HTTP. Los principios de REST son:

- Arquitectura cliente-servidor.
- Stateless.
- Cacheabilidad.
- Interfaz Uniforme.
- Identificación de Recursos.
- Manipulación de recursos.
- Mensajes autodescriptivos.
- Sistema de capas.
- Código On demand (es opcional, no se aplica).

En el contexto de CoreSafe, los principios anteriormente expuestos se implementan de la siguiente manera: el sistema adopta un modelo cliente-servidor, donde la parte del Backend expone los recursos y la lógica de negocio, mientras que los clientes consumen dichos recursos. Dicha comunicación es stateless, porque cada solicitud contiene toda la

información necesaria para ser procesada, utilizando tokens JWT para la autenticación. Los recursos se identifican mediante URLs y se manipulan con verbos HTTP (GET, POST, PUT, DELETE), retornando representaciones en formato JSON. Este flujo permite una comunicación de manera uniforme, desacoplada y escalable entre los distintos componentes del sistema.

Esta arquitectura monolítica con modularidad lógica entrega una estructura sólida, rendimiento y simplicidad de desarrollo, permitiendo un enfoque centrado en la entrega de valor funcional.

A continuación, se detalla el flujo general del sistema a través de un diagrama, destacando los componentes con sus interacciones.

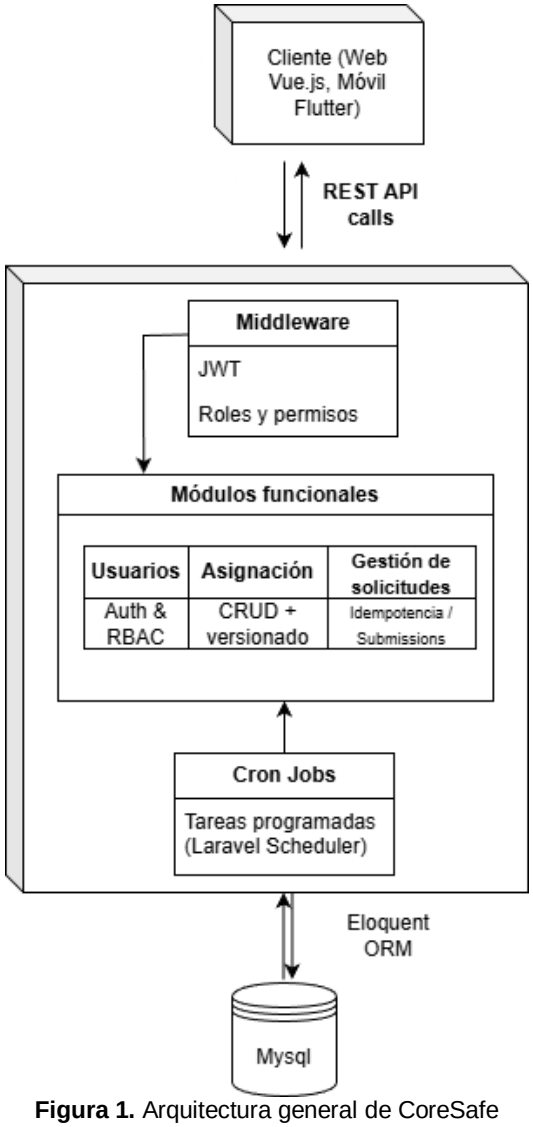


Figura 1. Arquitectura general de CoreSafe

Fuente: Elaboración propia

Como se aprecia en la Figura 1, la arquitectura del sistema se organiza en capas diferenciadas. En la parte superior se encuentran los clientes como la aplicación web desarrollada en Vue.js y la aplicación móvil en Flutter, los cuales interactúan con el Backend mediante llamadas a la API REST. Las solicitudes ingresan a la capa de Middleware, encargada de gestionar la autenticación con JWT y la autorización basada en roles y permisos (RBAC). Después, las peticiones son dirigidas a los módulos funcionales, donde se ejecuta la lógica de negocio del sistema, como la gestión de usuarios, asignaciones y formularios. Además, la arquitectura contempla Cron Jobs para la ejecución de tareas programadas, que en la implementación actual se encuentran configurados como procesos encolados que se ejecutan al iniciar sesión, por ejemplo, para tareas de sincronización y mantenimiento de datos. El acceso a la base de datos MySQL se realiza a través del ORM Eloquent.

Aunque se evaluaron otras arquitecturas como la de microservicios y enfoques serverless, se selecciona una API monolítica modular por la simplicidad de despliegue y mantenimiento. La modularidad permite separar responsabilidades y mantener una escalabilidad sin tanta complejidad de lo que implica arquitecturas distribuidas. Por lo tanto, responde estrictamente a las necesidades del proyecto, priorizando la estabilidad y rapidez en su implementación.

En relación con los patrones de diseño, se selecciona el modelo MVC (Modelo – Vista-Controlador) adaptado al desarrollo de APIs RESTful permitiendo separar la representación de los datos, la lógica de negocio y la gestión de solicitudes HTTP. Con esta adaptación, los controladores tienen un rol liviano, limitándose a la recepción de solicitudes HTTP, validación de datos y construcción de respuestas. Mediante la aplicación del patrón Service, se separa la responsabilidad del controlador con el fin de favorecer una arquitectura desacoplada. Con este enfoque se mejora la mantenibilidad del sistema, facilitando la implementación de pruebas de integración.

2.3. Tecnologías y frameworks utilizados

El desarrollo del Backend de CoreSafe se implementa con el lenguaje de programación PHP 8.2 en conjunto con el framework Laravel 12, apoyado por una base de datos MySQL, versiones que son estables y ampliamente usadas en entornos de producción, garantizando compatibilidad, seguridad y soporte a largo plazo. Esta elección de Laravel como framework se debió principalmente por la experiencia previa del equipo de desarrollo. Dicha experiencia permitió disminuir la curva de aprendizaje, pero la decisión del framework responde principalmente a las capacidades técnicas para el desarrollo de la API REST segura, escalable y mantenible, en concordancia con los requerimientos del proyecto.

El motor de base de datos seleccionado es MySQL, el cual se perfila como una buena alternativa para proyectos de media y gran escala. Además, es importante mencionar que está soportado por Laravel con el ORM Eloquent, y su estabilidad y rendimiento ya está comprobada en entornos productivos.

Laravel entrega un conjunto de herramientas orientadas al desarrollo de APIs RESTful, las que resultan un pilar fundamental para el Backend de CoreSafe. En esta línea, API Resources permite transformar los modelos en respuestas JSON estandarizadas, desacoplando la estructura de los datos de su representación externa. De la misma forma, el uso de middleware facilita la implementación de mecanismos de seguridad como la autenticación basada en JWT, en las rutas del sistema. Finalmente, la inyección de dependencias permite implementar el patrón Service Container de manera limpia y desacoplada, entregando mantenibilidad y evolución de código.

La solución se complementa con librerías del ecosistema Laravel, gestionadas a través de Composer. Para la autenticación se utilizó el paquete php-open-source-saver/jwt-auth, que permite implementar un esquema de autenticación stateless basado en JSON Web Tokens. Para la gestión de roles y permisos se empleó la librería Laravel permissions, permitiendo un control de acceso acorde a los distintos perfiles dentro del sistema. Docker fue utilizado tanto en desarrollo como en producción como contenedor de la aplicación para homologar entornos y evitar inconsistencias de versiones.

La seguridad del Backend de CoreSafe se implementa con:

- **Autenticación JWT**: La autenticación se implementa bajo un esquema Stateless (sin estado). A diferencia de las sesiones basadas en cookies en donde el servidor consulta la sesión en memoria o base de datos en cada petición, con JWT las credenciales viajan con el cliente. Por lo tanto, el flujo de seguridad implementado se escribe como:
    - Emisión: Si el usuario ingresa sus credenciales de forma correcta, el Backend genera un token (codificado con un algoritmo de hashing). Dicho Token tiene la identificación del usuario y la fecha de expiración.
    - Transporte: El cliente (frontend) almacena ese token y lo envía en la cabecera HTTP de cada solicitud en formato estándar de “Authorization: Bearer token_generado”.
    - Validación: El middleware del backend intercepta la solicitud antes de que llegue al controlador. Verifica la firma del token. Si el token fue alterado o ha expirado, la solicitud se rechaza con un error (401 Unauthorized), sin tener que consultar la base de datos para verificar la sesión.

- **Control de acceso basado en roles y permisos**: Para gestión de la autorización se implementa un modelo basado en Roles y permisos de la librería laravel permissions. Dicha arquitectura permite desacoplar los permisos de los usuarios, mejorando su administración dentro del sistema. En este punto de la implementación encontramos tres capas:

  - Roles: define cuál es el perfil del usuario dentro del sistema (Administrador, Analista, Observador).

  - Permisos: Son acciones atómicas (“Crear observación”, “Eliminar plantilla”).

  - Middlewares: Es la protección de las rutas API, que cumplen un papel de ser puertas lógicas.

  Para entender, antes de ejecutar una función llamada “*destroy”* de un controlador, el sistema verifica que, si el usuario tiene permiso para “eliminar registro”, si falla el framework lanza una excepción 403 deteniendo la ejecución, para que ningún usuario pueda ejecutar acciones con la manipulación de URLs si no tiene un rol asignado.

- **Prevención**: La seguridad defensiva de CoreSafe está basada en capas nativas de Laravel para mitigar las vulnerabilidades más frecuentes, como:

  - Inyección SQL: Para evitar usar consultas SQL concatenadas manualmente. Con Laravel se utiliza su ORM llamado Eloquent y el Query Builder que a su vez hacen el uso de construcción de parámetros PDO (PHP data Objects). Esto significa que la estructura de la consulta SQL se envía a la base de datos separada de los datos usando marcadores de posición.

  - Protección CSRF: Aunque al estar usando JWT se mitigan gran parte de este riesgo en particular, por defecto Laravel tiene incluido un middleware llamado VerifyCsrfToken para estas interacciones. El sistema genera un token único por sesión de usuario que se regenera de forma automática. Cada formulario de modificación de datos (que utilizan los verbos http como POST, PUT, DELETE), deben incluir este token. Si un sitio externo intenta forzar una acción en nombre del usuario desde otro dominio, la petición falla, porque no tiene dicho token generado por el servidor donde está alojado CoreSafe.

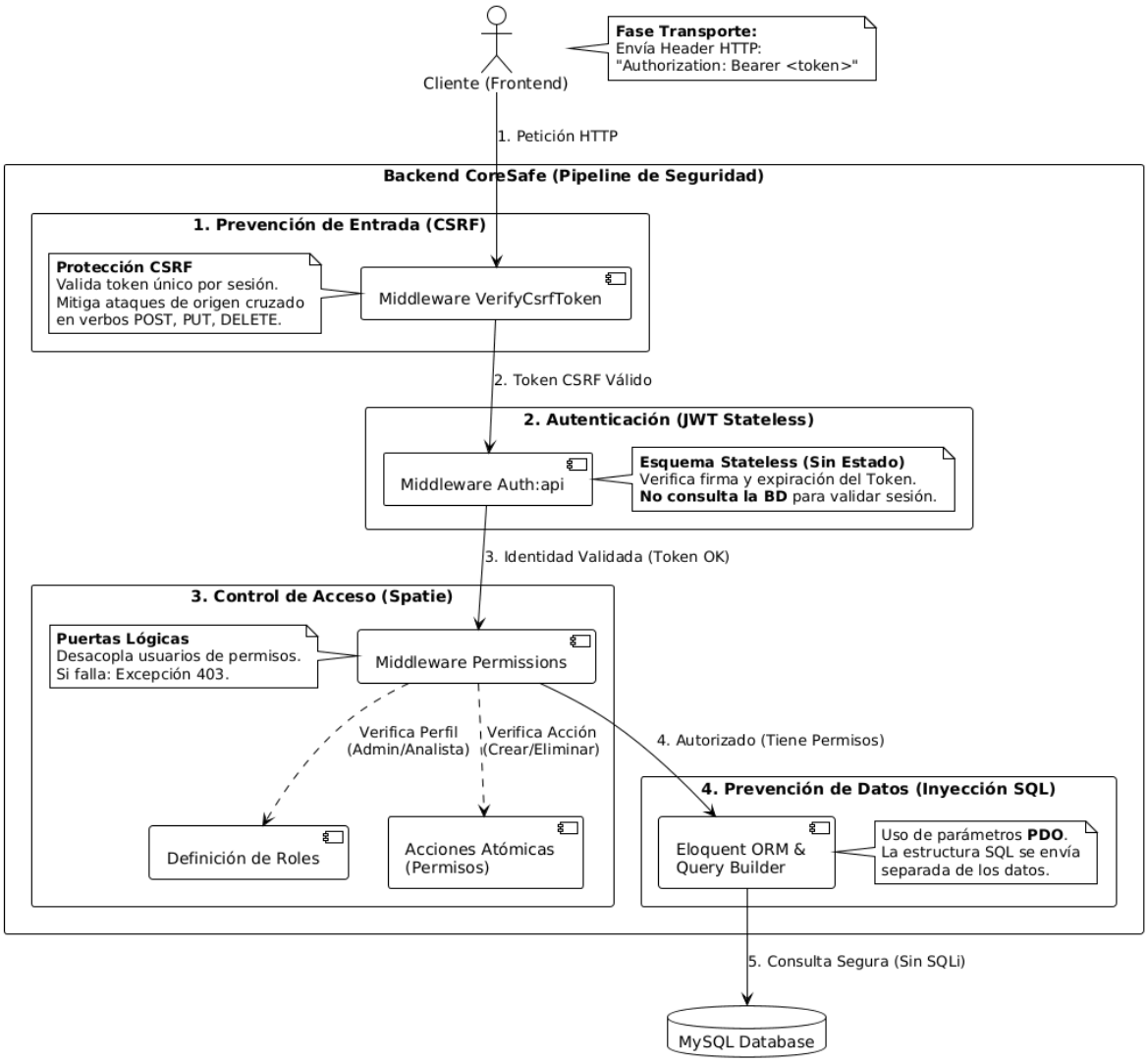


Figura 2. Capas de seguridad.

Fuente: Elaboración propia

La comunicación entre el Backend y aplicaciones clientes se hace a través de API REST, siguiendo la estandarización de los verbos HTTP. Además, se implementa la integración con Indemin para tener un login delegado, permitiendo tener una autenticación externa de manera segura y centralizada.

Se aplicaron feature test en el módulo Plantillas de Formularios con PHPUnit como herramienta de testing. De esta manera validamos el correcto funcionamiento de los endpoints relacionados a esa lógica de negocio además de asegurar la integridad de los datos en esas operaciones.

El Backend está desplegado en un servidor de Indemin mediante una imagen Docker, para tener portabilidad y consistencia en entornos de producción. Además, es válido mencionar

que la arquitectura monolítica con modularidad lógica en comparación de una de microservicios es por mantener simplicidad en la gestión, pero sin descuidar la escalabilidad ni la mantenibilidad.

Pese a evaluar alternativas como SpringBoot, FastAPI y Django, se selecciona una arquitectura monolítica con modularidad lógica en Laravel porque ayuda a la escalabilidad, mantenibilidad y rendimiento.

Tabla 1. Cuadro comparativo de tecnologías.

Categoría	Opción Seleccionada	Alternativas	Criterio de decisión
Lenguaje	PHP.	Python, Java.	Compatibilidad con APIs REST y soporte del framework
Framework	Laravel.	FastAPI, Django, Spring Boot.	Rapidez de desarrollo, soporte y comunidad
Base de datos	MySQL.	PostgreSQL, MongoDB	Soporte ORM, estabilidad.
Autenticación	Jwt + Roles y Permisos.	Sesiones nativas.	Granularidad de permisos
Comunicación	API REST.	WebSockets.	Interoperabilidad
Testing	PHPUnit.	PEST, JUnit.	Integración nativa con el framework.
Despliegue	Servidor Indemin + Docker.	AWS.	Portabilidad y simplicidad.
Arquitectura	Monolítica con modularidad lógica.	Microservicios.	Simplicidad operativa

Fuente: Elaboración propia

3. Diseño e Implementación

A continuación, se detalla el proceso de construcción del Backend CoreSafe. Se explican las decisiones de arquitectura, estructura de componentes, las buenas prácticas al escribir código y cómo se logró que el sistema se comunique con servicios externos, logrando una mirada más técnica de la solución.

3.1. Diseño de Componentes

El diseño de los componentes de CoreSafe garantiza modularidad, independencia y escalabilidad para cada funcionalidad del sistema. Esta modularidad está implementada a nivel lógico, mediante la separación de responsabilidades y dominios funcionales, utilizando la estructura del framework Laravel. Bajo los principios de una arquitectura monolítica con modularidad lógica, cada módulo representa un requerimiento del negocio, el cual se comunica con otros componentes de la plataforma a través de interfaces definidas por la API REST. Algunos módulos pueden representar partes específicas del sistema como la autenticación, autorización, servicios externos o administración de lógica de negocio, todos ellos expuestos mediante la API REST, la cual centraliza las peticiones de cada módulo para la correcta comunicación entre los sistemas frontend y el sistema Backend. Además, el uso de herramientas consolidadas como Laravel, MySQL, Vue.js y Flutter garantiza fiabilidad, seguridad y rendimiento.

3.1.1. Arquitectura y estructura de los componentes.

La arquitectura del Backend se diseña a partir del diseño Modelo – Vista – Controlador propio del framework Laravel [4], pero enfocado para el desarrollo de una API RESTful. Esta elección se basa en la separación de responsabilidades de manera clara, para poder construir un software mantenible, escalable y flexible ante cambios durante el desarrollo.

La justificación de la arquitectura monolítica modular la cual nos permite organizar todo el Backend en módulos funcionales, en donde cada uno tiene controladores, modelos y rutas responde principalmente a:

- **La naturaleza del problema**: porque se necesita centralizar la lógica de negocio en una única API.
- **La adaptabilidad**: permitiendo la modificación de los módulos sin afectar otras partes del sistema.

- **La escalabilidad**: facilitando la integración de clientes nuevos sin modificar la arquitectura base.

Los componentes del Backend se pueden representar como:

- **Rutas**: Direccionan las solicitudes HTTP a los controladores que corresponden y definen los endpoints.

- **Controladores**: Gestionan las solicitudes y respuestas HTTP, utilizando clases Request para validar datos de entrada y clases Resources para la construcción de las respuestas. Los controladores delegan la lógica de negocio a la capa de servicios, para que su función se limite a la orquestación del flujo de la petición.

- **Modelos**: Representan las entidades de la base de datos (formularios, plantillas, usuarios) además de gestionar la persistencia a través de Eloquent ORM.

- **Middlewares**: Interceptan solicitudes para aplicar autenticación (JWT), autorización (Roles y Permisos) y validaciones.
- **Servicios**: Encapsulan la lógica de negocio del sistema, actuando como intermediarios entre los controladores y los modelos. Estos servicios, que están implementados principalmente en el módulo de Plantillas, ejecutan las reglas de negocio y coordinan el acceso a los datos, evitando que los controladores manejen lógica compleja.

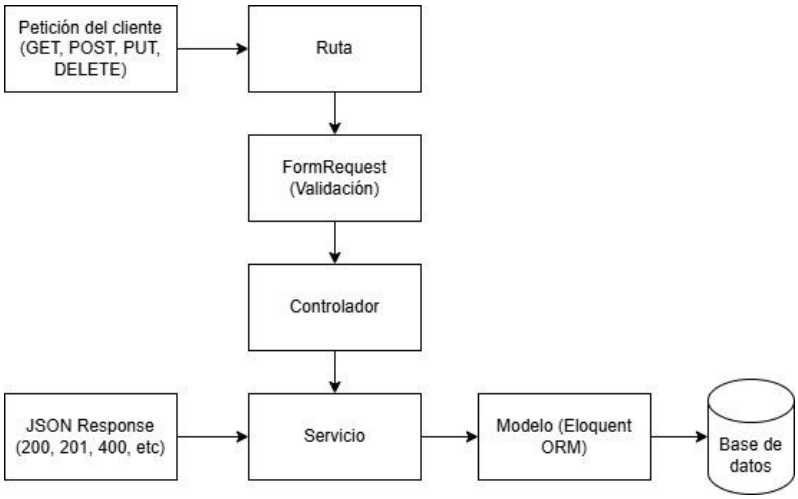


Figura 3. Diagrama del Modelo Vista Controlador API CoreSafe.

Fuente: Elaboración propia

La API está organizada en los siguientes módulos, cada uno se agrupa con sus recursos y endpoints relacionados:

1. **Módulo de Plantillas de Formularios**: Lógica de negocio enfocado a gestionar las plantillas que se utilizarán para crear formularios.

2. **Módulo de Formularios**: Lógica de negocio encargada de gestionar los formularios, los formularios que los observadores en terreno

3. **Módulo de Usuarios**: Lógica de negocio propios de los sistemas para mantener autenticación, roles y permisos.

4. **Módulo de Programas de Registro**: Lógica de negocio encargada de gestionar los programas de registros conductuales, normalmente tienen asociados las observaciones y los formularios.

5. **Módulo de Observaciones**: Lógica de negocio para gestionar las observaciones conductuales.

6. **Módulos de Tablas Catálogo**: Llamamos tablas catálogos a tablas que no dependen de otras tablas para existir, y que no cambian de forma frecuenta a través del tiempo.

Cada módulo está construido como un conjunto de rutas, controladores y modelos, aplicando los principios de encapsulamiento y bajo acoplamiento.

La justificación de la elección de Laravel como framework está fundamentada por:

- La experiencia previa del equipo: lo que mejora la capacidad de resolución de problemas del equipo.

- Madurez en la industria de software: por la flexibilidad de adaptar autenticación como JWT, la validación en varios niveles, testing (PHPUnit o PEST).

- Compatibilidad con el estándar RESTful: permite estructurar endpoints en base a verbos HTTP, facilitando la comunicación con clientes externos como aplicaciones webs o móviles.

La API permite la comunicación de los clientes web y móvil con una única base de datos, permitiendo la gestión de solicitudes de forma centralizada para garantizar la consistencia de los datos.

3.1.2. Uso de buenas prácticas y patrones de diseño.

Parte de las buenas prácticas que complementan la arquitectura del sistema se implementa el patrón Service para aislar la lógica de negocio y facilitar la implementación de feature tests.

El Service Pattern [5] o patrón Service es una práctica de arquitectura en donde se fomenta la separación de responsabilidades:

- Controlador: Recibe una petición, valida los datos, invoca un objeto servicio y devuelve una respuesta.
- Servicio: Encapsula toda la lógica de negocio como procesos, cálculos, reglas o integraciones.

Está basado en el principio de responsabilidad única de SOLID (principios para escribir un código limpio, mantenible y escalable), los cuales ayudan a desarrolladores a diseñar softwares flexibles y fácilmente escalables aplicable a cualquier lenguaje, en este caso PHP.

Ventajas:

- Código reutilizable, por centralizar la lógica de negocio.
- Facilidad para las pruebas basadas en funcionalidades, por la lógica desacoplada de los controladores.
- Mejora la mantenibilidad, porque permite modificar la lógica de negocio sin afectar al controlador mismo.
- Mejora la claridad de código, por el principio de responsabilidad única.

```php
public function store(StoreTemplateFormRequest $request)
{
    Gate::authorize('template-forms.store', 'api');
    DB::beginTransaction();

    try {
        $form = TemplateForm::create(collect($request)->except(['form_sections'])->all());

        foreach ($request->form_sections ?? [] as $sectionData) {
            $section = $form->templateFormSections()->create([
                'name' => $sectionData['name']
            ]);

            foreach ($sectionData['form_questions'] ?? [] as $questionData) {
                $section->templateFormQuestions()->create([
                    'question' => $questionData['question'],
                ]);
            }
        }

        DB::commit();

        return new TemplateFormResource($form->load('templateFormSections.templateFormQuestions'));

    } catch (\Exception $e) {
        DB::rollBack();
        return response()->json(['error' => $e->getMessage()], 500);
    }
}
```

Código 1. Clase controlador normal

Fuente: Elaboración propia

```php
public function store(
    StoreTemplateFormRequest $request,
    TemplateFormService $templateFormService
) {
    Gate::authorize('template-forms.store', 'api');

    $form = $templateFormService->store($request->validated());

    return new TemplateFormResource($form->load('templateFormSections.templateFormQuestions'));
}
```

Código 2. Controlador delegando lógica.

Fuente: Elaboración propia

```php
class TemplateFormService
{
    public function index(): LengthAwarePaginator
    {...
    }

    /**...
    public function store(array $data): TemplateForm
    {...
    }

    public function show(TemplateForm $templateForm): TemplateForm
    {...
    }

    public function update($data, TemplateForm $templateForm): TemplateForm
    {...
    }

    public function destroy(TemplateForm $templateForm): void
    {...
    }
}
```

Código 3. Clase del Service Patern

Fuente: Elaboración propia

Además, mencionar practicas adicionales que se aplicaron al desarrollo de CoreSafe como:

- El uso de Git para el control de versión con ramas main, develop, feature.
- Aplicación de pruebas basadas en funcionalidad para validar enddpoints y lógica de negocio.
- Validación de datos a nivel de request y modelo.
- Uso de middlwares para autenticación y autorización.
- Documentación de endpoints con estándares RESTful

3.1.3. Integración con el sistema general.

El Backend basa su integración con los clientes Vue.js y Flutter a través del protocolo HTTP. Siguiendo el formato estandarizado JSON.

Flujo de datos para la funcionalidad crear una plantilla de formulario

1. Se envía una solicitud desde el Frontend.
2. Interactúa con la ruta.
3. Se ejecuta el método o función del controlador.
4. Resuelve lógica de negocio.
5. El sistema responde con una respuesta JSON con código HTTP (200, 201, 400, 401).

El flujo descrito demuestra como los componentes del Backend trabajan de forma conjunta con el fin de cumplir una regla de negocio de CoreSafe.

```
{
  "name": "Checklist de Seguridad",
  "description": "Formulario para inspección de condiciones en obra",
  "form_sections": [
    {
      "name": "Área de trabajo",
      "description": "Evaluación del entorno físico",
      "form_questions": [
        {
          "question": "¿El área está libre de obstáculos?"
        },
        {
          "question": "¿Hay señalización visible?"
        }
      ]
    },
    {
      "name": "Equipos de protección",
      "description": "Verificación del uso de EPP",
      "form_questions": [
        {
          "question": "¿Se utiliza casco de seguridad?"
        },
        {
          "question": "¿El trabajador porta arnés?"
        }
      ]
    }
  ]
}
```

Código 4. Ejemplo de estructura JSON de petición post para crear una Plantilla de Formulario.

Fuente: Elaboración propia

3.2. Detalles de la codificación y desarrollo.

3.2.1. Metodología de trabajo y control de versiones

El desarrollo del proyecto CoreSafe fue realizado bajo metodología ágil, organizando las actividades con un total de 8 sprints cada uno con una duración de dos semanas. El equipo de trabajo estuvo integrado por seis personas, con responsabilidades distribuidas en áreas como Backend, Frontend, arquitectura, base de datos, análisis de datos y gestión de proyecto. En el caso del Backend, el desarrollo fue realizado por una persona, con aportes parciales de otros integrantes según la funcionalidad implementada.

El control de versiones utilizado fue Git, con el repositorio alojado en GitHub, siguiendo el flujo de trabajo GitFlow. Este flujo toma en cuenta una rama main destinada para versiones

estables de producción, una rama develop para desarrollo, y ramas feature y bugfix para la implementación de funcionalidades y correcciones específicas.

Antes de la integración del código, las funcionalidades eran discutidas con el equipo del proyecto. Debido a restricciones de tiempo del proyecto y de los integrantes del equipo, la integración del código fue realizada mediante merges directos a la rama develop, manteniendo la trazabilidad de los cambios a través del historial de Git.

Sin embargo, como práctica recomendada el flujo de Pull Request se consideró para futuras iteraciones del proyecto. Dicho flujo, propone que cada funcionalidad desarrollada en una rama feature debe ser integrada mediante un Pull Request hacia la rama develop, permitiendo su revisión por al menos una persona antes el merge, para tener un mejor control de calidad de código.

3.2.2. Entorno de desarrollo y gestión de dependencias

Para el desarrollo del Backend se utilizó el entorno de desarrollo basado en XAMPP, con versiones específicas de PHP y MySQL definidas para el proyecto. Docker fue utilizado solo para homologar el entorno en producción.

La gestión de dependencias fue realizada con Composer, que es el gestor de paquetes del ecosistema PHP. En el archivo composer.json se definieron las librerías necesarias para el funcionamiento del sistema, incluido el framework y dependencias a utilizar tales como autenticación, autorización, y otras funcionalidades asociadas al Backend. Este método de gestión de dependencias favoreció la mantenibilidad el proyecto.

3.2.3. Estándares y calidad de código

Durante el desarrollo del Backend se siguieron estándares de codificación que están alineados con la guía de código de PHP (PSR – 12 por sus siglas), para garantizar legibilidad, consistencia y mantenibilidad al código desarrollado. Se utilizó nomenclatura camelCase para variables y métodos, y PascalCase para clases, siguiendo las buenas prácticas promovidas por el framework Laravel.

También se promovió la separación de responsabilidades aplicando el patrón Service a un módulo representativo, evitando concentración de lógica de negocio en los controladores. De esta manera se logró tener una mejor organización de código reduciendo la probabilidad de errores por inconsistencias en el código.

3.2.4. Estructura de directorios del proyecto

El Backend fue desarrollado usando una estructura propuesta por el framework Laravel, pero adaptada para dar soporte a una arquitectura monolítica con modularidad lógica, donde la lógica de negocio para el módulo más representativo, así como la integración con servicios externos está organizada en capas de servicios la cual fue alojada en el directorio app/Services.

Los controladores se ubican en app/Http/Controllers, con un rol de gestión de solicitudes HTTP. Los modelos de dominio se mantienen en app/Models, utilizando el ORM Eloquent para la interacción con la base de datos.

La organización del código se caracteriza por tener una estructura clara, escalable y alineada con los principios de separación de responsabilidades, facilitando la evolución del proyecto sin recurrir a una arquitectura más compleja.

3.3. Pruebas y Validación

Para verificar la calidad del software y el cumplimiento de los requerimientos propuestos para el Backend de CoreSafe, se estableció una estrategia de pruebas enfocada en validar tanto el funcionamiento de los endpoints como la robustez del sistema, la integridad de los datos y el cumplimiento de los lineamientos de seguridad definidos en la arquitectura RESTful.

La estrategia de validación se centró en pruebas del tipo funcional e integración, simulando escenarios reales de uso del sistema desde la perspectiva de los clientes de la API. Este enfoque permitió verificar el comportamiento del flujo de las solicitudes HTTP, desde la autenticación y la autorización del usuario, hasta llegar al procesamiento de la lógica de negocio y la generación de respuestas en formato JSON, priorizando la estabilidad del sistema y, en consecuencia, la correcta aplicación de las reglas de negocio.

Para garantizar la calidad del software y que se cumplan los requerimientos en base a la arquitectura RESTful, se decidió aplicar:

- Pruebas manuales con el programa Postman [6]: Para cada endpoint se verifica que esté interactuando con la API, haciendo validación de los códigos de estados y estructuras JSON que deben retornar según la lógica de negocio que se definió.
- Pruebas basadas en Feature Tests: En el proyecto se implementa una rama derivada de “develop” una rama llamada “test” en donde por ejemplo a la lógica de negocio relacionado con las plantillas de los formularios se aplica el patrón de diseño Service para aislar la lógica de los controladores, posteriormente a este controlador se le aplican test basados en funcionalidad o los llamados “feature tests” con la suit de PHPUnit.

Las pruebas manuales fueron realizadas a través de una colección Postman, organizadas en módulos funcionales. La colección validó de forma semántica los endpoints definidos, usando variables de entorno para poder diferenciar entre configuraciones locales y de producción. Esta práctica facilitó la reutilización de pruebas, trazabilidad de resultados y además sirvió como apoyo para la documentación de la API.

Por restricciones de tiempo y alcance en este proyecto Backend no se pudieron realizar pruebas unitarias, ni pruebas de rendimiento y carga, pero quedarán como opciones de mejoras para futuras iteraciones.

3.3.1. Estrategias de testing aplicadas a los componentes.

Los Feature Tests o pruebas basadas en funcionalidades, corresponden a pruebas que validan el comportamiento del sistema desde la perspectiva del cliente (o consumidor) de la API, simulando situaciones lo más cercanas al entorno real. Estas pruebas permiten validar el cumplimiento de los requerimientos funcionales, los flujos de usuario del sistema y la correcta comunicación entre los componentes del Backend.

En el contexto de CoreSafe, los Feature Tests se utilizaron para validar el flujo de las solicitudes HTTP, considerando los casos exitosos y también los escenarios negativos. Las validaciones incluyen la verificación de códigos de estado HTTP (201, 401, 403, 422), la estructura y las respuestas en formato JSON, el funcionamiento de las validaciones de los datos de entrada y la correcta implementación de las reglas de autorización basadas en roles y permisos.

Específicamente para el módulo de Plantillas de formularios, las pruebas fueron aplicadas directamente sobre los controladores, los cuales delegan la lógica de negocio sobre una capa de servicios. La aplicación del patrón Service para encapsular la lógica de negocio permite validar el sistema sin acoplar las pruebas a los detalles del controlador. De esta manera, los Feature Tests se centran en verificar que la funcionalidad cumpla con el comportamiento esperado, independientemente de cómo se implemente internamente la lógica de negocios.

Debido a las restricciones de tiempo y alcance, no se implementaron pruebas unitarias sobre componentes individuales, y esta estrategia de validación sobre el módulo de Plantillas actuará como base para los demás módulos para que el sistema sea aún más robusto. No obstante, el enfoque basado en pruebas funcionales permite validar de forma efectiva los flujos más importantes del sistema, asegurando la estabilidad del Backend y la correcta aplicación de las reglas de negocio en los módulos determinados.

En el siguiente código de ejemplo, que es la funcionalidad de crear plantillas de formularios, se muestra el controlador al cual se le aplican las pruebas, y los distintos casos que se deben validar para que la suite de pruebas cubra los requerimientos de la funcionalidad.

```php
#[Test]
public function se_crea_una_plantilla_de_formulario(): void
{
    // Arrange
    $tituloPlantilla = 'Plantilla de test';
    $data = [ ···
    ];

    $user = $this->create_user_admin();

    // Action
    $response = $this->actingAs($user, 'api')->postJson(route('template-forms.store'), $data);
    $response->assertJsonStructure([
        'data' => [
            'id',
            'name',
            'form_sections' => [
                '*' => [
                    'id',
                    'name',
                    'form_questions' => [
                        '*' => [
                            'id',
                            'question'
                        ]
                    ]
                ]
            ]
        ]
    ]);

    // Accerts
    $PlantillaCreada = TemplateForm::where('name', $tituloPlantilla)->first();
    $this->assertNotNull($PlantillaCreada, 'no hay plantilla');
    $response->assertStatus(201);
}
```

Código 5. Test de caso exitoso.

Fuente: Elaboración propia

```
PASS  Tests\Feature\Http\Controllers\TemplateFormControllerTest
✓ se listan las plantillas                                              1.66s
✓ se crea una plantilla de formulario                                   0.25s
✓ se muestra una plantilla con formato correcto                         0.22s
✓ se actualiza un template form                                         0.22s
✓ se elimina plantilla                                                  0.21s
✓ usuario sin permiso no puede crear plantilla                          0.15s
✓ usuario sin permiso no puede listar plantilla                         0.14s
✓ usuario sin permiso no puede ver plantilla                            0.13s
✓ usuario sin permiso no puede editar plantilla                         0.13s
✓ usuario sin permiso no puede eliminar plantilla                       0.12s
✓ usuario no autenticado no puede crear plantilla                       0.03s
✓ usuario no autenticado no puede listar plantilla                      0.03s
✓ usuario no autenticado no puede ver plantilla                         0.04s
✓ usuario no autenticado no puede actualizar plantilla                  0.03s
✓ usuario no autenticado no puede eliminar plantilla                    0.03s
✓ usuario ingresa datos erroneos en formulario crear plantilla          0.17s
✓ usuario ingresa datos erroneos en formulario actualizar plantilla     0.17s

Tests:    17 passed (139 assertions)
Duration: 4.03s
```

Figura 4. Tests sobre Plantilla de formularios.

Fuente: Elaboración propia

3.3.2. Resolución de bugs y optimización

La resolución de errores y la optimización del Backend de CoreSafe se integraron al flujo de trabajo, en concordancia con la metodología GitFlow utilizada en el proyecto. Este enfoque permitió tener un control más estructurado de los cambios, aportando positivamente a la detección, corrección y validación de fallos, sin afectar la estabilidad del Backend.

Al identificar un error, mediante pruebas manuales con Postman o pruebas automatizadas mediante Feature Tests, el problema se reproducía en el entorno de desarrollo sobre la rama develop. A continuación, se creaba una rama específica del tipo bugfix, destinada a la corrección del error identificado.

La corrección del error incluía la resolución del comportamiento defectuoso, con el objetivo de prevenir regresiones futuras. Cuando se validaba dicha solución, los cambios se integraban a la rama develop, a través de un Pull Request, asegurando la trazabilidad de las modificaciones realizadas.

Desde la mirada de la optimización, se identificaron problemas de rendimiento durante las pruebas, como el patrón N + 1 en consultas a base de datos, dentro del módulo de Plantillas de Formularios, y el módulo de Formularios (que son muy similares). Dicho problema fue mitigado usando el ORM de Laravel (Eloquent), reduciendo el número de consultas a la base de datos y mejorando el rendimiento general del sistema.

Dicho proceso permitió corregir errores funcionales y también mejorar progresivamente la eficiencia y mantenibilidad del Backend, logrando así un sistema más estable y preparado para futuras extensiones funcionales.

3.3.3. Métricas de cobertura

Dado el alcance que se estableció para el desarrollo del Backend de CoreSafe, las pruebas funcionales se centraron en el Módulo de Plantillas de Formularios, el cual fue seleccionado como módulo representativo por su complejidad funcional y por involucrar los mecanismos más transversales del sistema, tales como autenticación JWT, control de acceso basado en roles y permisos, validaciones de datos y operaciones CRUD.

El sistema cuenta con un total de 107 endpoints documentados y expuestos a través de la API REST. Las pruebas mediante Feature Tests fueron aplicadas sobre el conjunto de endpoints asociados al Módulo de Plantillas de Formularios, correspondientes a las siguientes rutas:

- GET /template-forms
- POST /template-forms
- PUT /template-forms/{id}
- GET /template-forms/{id}
- DELETE /template-forms/{id}

En total, se implementaron 17 Feature Tests, que abarcan tanto los escenarios de éxito, validaciones de estructura de datos, como errores de autorización (403 Forbidden) y también errores de autenticación (401 Unauthorized).

Tabla 2. Resumen de cantidad de pruebas

Elemento evaluado	Resultado
Total de endpoints	107
Endpoints con pruebas	5
Módulos cubiertos	Plantillas de Formularios
Tipo de pruebas aplicadas	Feature Tests

Fuente: Elaboración propia

Aunque no todos los módulos del sistema tienen Feature Tests, el enfoque adoptado permitió validar un módulo crítico y que además es representativo del comportamiento general del Backend. Esto se debe a que todos los endpoints comparten la misma infraestructura de autenticación, autorización y validación. Dichas pruebas entregan un alto grado de confianza sobre la robustez general del sistema.

3.3.4. Pruebas de seguridad y control de acceso

Parte de la validación del Backend es el diseño y ejecución de pruebas orientadas a verificar el correcto funcionamiento de los mecanismos de seguridad establecidos en la arquitectura, en específico la autenticación basada en JWT (JSON Web Tokens) y el control de acceso basado en roles y permisos (RBAC).

Las pruebas de seguridad fueron implementadas en los Feature Tests, centrándose en los escenarios negativos que buscan asegurar que el sistema responda de forma correcta ante intentos de accesos no autorizados. En este contexto, se validó que los endpoints protegidos no puedan ser consumidos por usuarios no autenticados, retornando respuestas con códigos HTTP (401 Unauthorized). También se verificó que los usuarios autenticados, pero sin permisos, reciban respuestas HTTP 403 (Forbidden).

Específicamente en el módulo de Plantillas de Formularios se implementaron casos de prueba en los que se verifica que un usuario con rol limitado no pueda ejecutar acciones administrativas como la creación, modificación o eliminación de plantillas. Estos casos garantizan la correcta aplicación del esquema de roles y permisos definidos en los requerimientos, asegurando la integridad y confidencialidad de las funcionalidades críticas del sistema.

El enfoque de pruebas de seguridad valida que la API cumple con los flujos funcionales esperados, además de responder de forma robusta ante accesos indebidos, reforzando la confiabilidad del Backend y en sintonía con los principios de seguridad propios de una aplicación RESTful.

3.4. Integración con otros componentes

Laravel, El proyecto contiene una comunicación con los servicios de Indemin nuestro patrocinador, en el cual consta de una lógica de Login delegado, es decir que para poder entrar a la plataforma se debe primero validar con un pin de usuario que contiene el usuario registrado en Indemin. Además, tenemos un job corriendo el cual es el encargado de cargar los datos de la tabla intermedia users worksites que nos dice las faenas de Indemin que están asociadas a los usuarios.

```php
class IndeminClient
{
    protected string $apiUrl;
    protected array $headers;
    protected bool $verify;

    public function __construct()
    {…
    }

    /**
     * Cargamos los datos de las faenas del usuario con la API de Indemin.
     * Devuelve respuesta JSON con las faenas del usuario.
     */
    public function faenasGet(int $idUser): array
    {…
    }

    /**
     * Autentica a un usuario con la API de Indemin.
     * Devuelve el cuerpo de la respuesta JSON o lanza una ExternalServiceException en caso de error.
     */
    public function loginIndemin(string $usuario, string $pin): array
    {…
    }
}
```

Código 6. Clase Servicio de Indemin

Fuente: Elaboración propia

En el caso de la implementación de sincronizar las faenas de Indemin con cada usuario, se hace a través de la creación de un Job, que dicho proceso el worker es ejecutado en el servidor de forma asincrónica, específicamente en este job se aísla la lógica de sincronización de usuarios – faenas, para esto también es necesario la creación de una tabla en base de datos encargada de encolar los procesos de los jobs, también es necesario crear una migración nueva.

4. Conclusiones

El desarrollo del Backend de CoreSafe permitió consolidar el conocimiento sobre la arquitectura de software, diseño e implementación de APIs RESTful y la aplicación de buenas prácticas de desarrollo. A lo largo del proyecto se comprendió la importancia de separar responsabilidades mediante el patrón Service, mejorando la mantenibilidad, escalabilidad y claridad estructural del sistema.

Con respecto a los objetivos específicos planteados en el inicio de la presente tesina, se dio cumplimiento a través de la implementación de un modelo relacional que garantiza la integridad de los datos y la consistencia transaccional, además del desarrollo de una API REST que permite interoperabilidad entre clientes web y móvil. A su vez se implementó un sistema de seguridad basado en autenticación mediante JWT y autorización granular con roles y permisos (RBAC), consolidando la protección de los recursos de la organización. Los componentes mencionados fueron validados a través de pruebas manuales y técnicas descritas en el capítulo de implementación, certificando el cumplimiento de los requerimientos que se habían definido inicialmente.

Desde el punto de vista de la arquitectura, se seleccionó una arquitectura monolítica con modularidad lógica usando el framework Laravel, permitiendo rapidez en el desarrollo y simplicidad en la gestión del proyecto. Sin embargo, se reconoce que en contextos de alta demanda futura se podría evaluar una migración hacia arquitecturas basadas en microservicios o enfoques serverless, para buscar más desacoplamiento estructural. Esta reflexión nos enseña la necesidad de equilibrar decisiones de corto plazo con proyecciones de largo plazo.

Como desafíos técnicos importantes, se identifica la optimización de consultas a la base de datos, detectando y resolviendo el problema de N+1 asociado al uso del ORM. La corrección de este patrón permitió mejorar los tiempos de respuesta de la API, aspecto sumamente relevante en entornos industriales donde la conectividad es variable. Asimismo, la correcta definición de los módulos funcionales y la implementación de los mecanismos de autenticación y autorización requirieron un estudio profundo de la documentación técnica y de las buenas prácticas, fortaleciendo la capacidad de análisis y resolución de problemas complejos.

Como principal logro, CoreSafe transforma un proceso manual de registro de observaciones en un activo digital para Indemin, disminuyendo la latencia entre la identificación de un riesgo en terreno y la disponibilidad de la información para la toma de decisiones gerenciales. Con ello, se mejora la eficiencia operativa y se fortalece la cultura preventiva en Indemin.

Desde una perspectiva profesional, el proyecto evidencia la aplicación de principios de arquitectura limpia, separación de capas y control de acceso seguro en un entorno empresarial real. La incorporación de pruebas con Postman y PHPUnit (para el módulo representativo) permitió validar los flujos críticos del sistema, aportando fiabilidad en la gestión de formularios y observaciones, y asegurando la calidad de los entregables.

Para finalizar, como opciones de mejora futura se proponen la incorporación de pruebas de rendimiento y de carga, integración de módulos de análisis predictivo basados en datos históricos. Las mejoras permitirían ampliar el valor del sistema, consolidándolo como una plataforma robusta y adaptable ante los nuevos requerimientos industriales.

5. Agradecimientos.

Agradezco a mi familia en especial a mi madre que con la fortaleza y orgullo de nuestra sangre araucana me alienta a mantenerme firme en cada desafío, a mi padre trabajador de toda la vida dando todo su esfuerzo para sus hijos, a mi abuelita por estar apoyándome en

todo mi camino tanto personal como profesional y a mis gatitas Arween y Pantufla por acompañarme en los tiempos de estudio en mi etapa universitaria.

6. Referencias

[1] SONAMI, «sonami,» 2026. [En línea]. Available: https://www.sonami.cl/v2/seguridad-en-mineria/accidentes-fatales/.

[2] R. Fielding, «Rest Thesis,» 2000. [En línea]. Available: https://es.scribd.com/document/712345721/Roy-Fielding-Rest-Thesis.

[3] IBM, «Tipos de Arquitecturas de Software,» 2020. [En línea]. Available: https://www.ibm.com/think/topics/application-architecture-types.

[4] «Laravel Documentation,» 2025. [En línea]. Available: https://laravel.com/docs/12.x/routing.

[5] O. C. Kingsley, «Uso del patrón de capa de servicio en PHP para código limpio y escalable,» [En línea]. Available: https://dev.to/otutukingsley/using-the-service-layer-pattern-in-php-for-clean-and-scalable-code-15fb.

[6] «Postman Documentation,» 2025. [En línea]. Available: https://learning.postman.com/docs/introduction/overview/.

7. Anexos

Tabla 3. Documentación de Endpoints.

Ruta Endpoint	Verbo HTTP	Descripción	Datos enviados	Datos recibidos
/auth/login	POST	Autenticación de usuario	Nombre y pin del usuario	Token de autorización
/auth/logout	POST	Terminar sesión de usuario	Token del usuario	Mensaje de fin de sesión

/auth/me	GET	Información de perfil del usuario	Token del usuario	Información del usuario autenticado
/permissions	GET	Listar todos los permisos	No se envían datos	Listado de permisos
/permissions	POST	Crear un permiso	Nombre del permiso	El nombre creado
/permissions/{permissionid}	PUT	Actualizar un permiso	El id del permiso a editar con el nombre nuevo	El permiso editado
/permissions/{permissionid}	GET	Obtener un permiso	El id del permiso	Datos de un permiso
/permissions/{permissionid}	DELETE	Eliminar un permiso	El id del permiso	Mensaje de confirmación
/roles	GET	Listar todos los roles	No se envían datos	Listado de roles
/roles	POST	Crear un rol	Nombre del rol	El rol creado
/roles/{roleid}	GET	Obtener un rol	El id del rol	El rol encontrado
/roles/{roleid}	PUT	Actualizar un rol	El id del rol con el nombre del rol actualizado	El rol actualizado
/roles/{roleid}	DELETE	Eliminar un rol	El id del rol	Mensaje de confirmación
/activities	GET	Listar todas las actividades	No se envían datos	Listado de actividades
/activities	POST	Crear una actividad	Datos de la actividad	La actividad creada
/activities/{activityid}	PUT	Actualizar una actividad	Id de la actividad, con los datos nuevos	La actividad actualizada
/activities/{activityid}	GET	Obtener una actividad	El id de la actividad	La actividad
/activities/{activityid}	DELETE	Eliminar una actividad	El id de la actividad	Mensaje de confirmación
/equipments	GET	Listar equipos	No se envían datos	Listado de equipos
/equipments	POST	Crear un equipo	Datos del equipo	El equipo creado
/equipments/{equipmentid}	PUT	Actualizar un equipo	El id del equipo con los datos nuevos	El equipo actualizado
/equipments/{equipmentid}	GET	Obtener un equipo	El id del equipo	El equipo encontrado
/equipments/{equipmentid}	DELETE	Eliminar un equipo	El id del equipo	Mensaje de confirmación
/worksites	GET	Listar faenas del usuarios	No se envían datos	Listado de faenas
/worksites	POST	Crear una faena	Datos de la faena	La faena creada
/worksites/{worksiteid}	GET	Obtener una faena	El id de la faena	La faena encontrada

/worksites/{worksiteid}	PUT	Actualizar una faena	El id de la faena con los datos nuevos	La faena actualizada
/worksites/{worksiteid}	DELETE	Eliminar una faena	El id de la faena	Mensaje de confirmación
/positions	GET	Listar cargos	No se envían datos	Listado de cargos
/positions	POST	Crear un cargo	Datos del cargo	El cargo creado
/positions/{positionid}	PUT	Actualizar un cargo	El id del cargo con los datos nuevos	El cargo actualizado
/positions/{positionid}	GET	Obtener un cargo	El id del cargo	El cargo encontrado
/positions/{positionid}	DELETE	Eliminar un cargo	El id del cargo	Mensaje de confirmación
/barriers	GET	Listar todas las barreras	No se envían datos	Listado de barreras
/barriers	POST	Crear una barrera	Datos de la barrera	La barrera creada
/barriers/{barrierid}	PUT	Actualizar una barrera	El id de la barrera con los datos nuevos	La barrera encontrada
/barriers/{barrierid}	GET	Obtener una barrera	El id de la barrera	La barrera encontrada
/barriers/{barrierid}	DELETE	Eliminar una barrera	El id de la barrera	Mensaje de confirmación
/companies	GET	Listar todas las empresas	No se envían datos	Listado de las empresas
/companies	POST	Crear una empresa	Datos de la empresa	La empresa creada
/companies/{companyid}	PUT	Actualizar una empresa	El id de la empresa con los datos nuevos	La empresa actualizada
/companies/{companyid}	GET	Obtener una empresa	El id de la empresa	La empresa encontrada
/companies/{companyid}	DELETE	Eliminar una empresa	El id de la empresa	Mensaje de confirmación
/managements	GET	Listar todas las gerencias	No se envían datos	Listado de las gerencias
/managements	POST	Crear una gerencia	Datos de la gerencia	La gerencia creada
/managements/{managementid}	PUT	Actualizar una gerencia	El id de la gerencia, con los datos nuevos	La gerencia actualizada
/managements/{managementid}	GET	Obtener una gerencia	El id de la gerencia	La gerencia encontrada
/managements/{managementid}	DELETE	Eliminar una gerencia	El id de la gerencia	Mensaje de confirmación
/departments	GET	Listar todos los departamentos	No se envían datos	Listado de los departamentos

/departments	POST	Crear un departamento	Datos del departamento	Departamento creado
/departments/{departmentid}	PUT	Actualizar un departamento	El id del departamento con los datos nuevos	El Departamento actualizado
/departments/{departmentid}	GET	Obtener un departamento	El id del departamento	El departamento encontrado
/departments/{departmentid}	DELETE	Eliminar un departamento	El id de un departamento	Mensaje de confirmación
/areas	GET	Listar todas las áreas	No se envían datos	Listado de áreas
/areas	POST	Crear un área	Datos del área	El área creada
/areas/{areaid}	PUT	Actualizar un área	El id del área con los datos nuevos	El área actualizada
/areas/{areaid}	GET	Obtener un área	El id del área	El área encontrada
/areas/{areaid}	DELETE	Eliminar un área	El id de un área	Mensaje de confirmación
/critical-risks	GET	Listar todos los riesgos críticos	No se envían datos	Listado de los riesgos críticos
/critical-risks	POST	Crear un riesgo crítico	Datos del riesgo crítico	El riesgo crítico creado
/critical-risks/{criticalriskid}	PUT	Actualizar un riesgo crítico	El id del riesgo crítico, con los datos nuevos	El riesgo crítico actualizado
/critical-risks/{criticalriskid}	GET	Obtener un riesgo crítico	El id del riesgo crítico	El riesgo crítico encontrado
/critical-risks/{criticalriskid}	DELETE	Eliminar un riesgo crítico	El id de un riesgo crítico	Mensaje de confirmación
/turns	GET	Listar todos los turnos	No se envían datos	Listado de turnos
/turns	POST	Crear un turno	Datos del turno	El turno creado
/turns/{turnid}	PUT	Actualizar un turno	El id del turno, con los datos nuevos	El turno actualizado
/turns/{turnid}	GET	Obtener un turno	El id del turno	El turno encontrado
/turns/{turnid}	DELETE	Eliminar un turno	El id de un turno	Mensaje de confirmación
/conducts	GET	Listar todas las conductas	No se envían datos	Listado de conductas
/conducts	POST	Crear una conducta	Datos de una conducta	La conducta creada
/conducts/{conductid}	PUT	Actualizar una conducta	Id de la conducta con los datos nuevos	La conducta actualizada
/conducts/{conductid}	GET	Obtener una conducta	El id de la conducta	La conducta encontrada
/conducts/{conductid}	DELETE	Eliminar una conducta	El id de una conducta	Mensaje de confirmación

/observed-tasks	GET	Listar todas las tareas observadas	No se envían datos	Listado de las tareas observadas
/observed-tasks	POST	Crear una tarea observada	Datos de una tarea observada	La tarea observada creada
/observed-tasks/{observed-taskid}	PUT	Actualizar una tarea observada	Id de la tarea observada más los datos nuevos	La tarea observada actualizada
/observed-tasks/{observed-taskid}	GET	Obtener una tarea observada	Id de la tarea observada	La tarea observada encontrada
/observed-tasks/{observed-taskid}	DELETE	Eliminar una tarea observada	El id de una tarea	Mensaje de confirmación
/program-details	GET	Listar detalles de programas	No se envían datos	Listado de detalle de programas
/program-details	POST	Crear un detalle de programa	Datos del detalle programa	El detalle de programa creado
/program-details/{programid}	PUT	Actualizar un detalle de programa	Id del detalle de programa mas los datos nuevos	El detalle de programa actualizado
/program-details/{programid}	GET	Obtener un detalle de programa	El id de detalle de programa	El detalle de programa encontrado
/program-details/{programid}	DELETE	Eliminar un detalle de programa	El id de un programa	Mensaje de confirmación
/program-registers	GET	Listar todos los programas de registro	No se envían datos	Listado de programa de registros
/program-registers	POST	Crear un programa de registro	Datos del programa de registro	El programa de registro creado
/program-registers/{programregisterid}	PUT	Actualizar un programa de registro	El id del programa de registro mas los datos nuevos	EL programa de registro actualizado
/program-registers/{programregisterid}	GET	Obtener un programa de registro	El id del programa de registro	El programa de registro encontrado
/program-registers/{programregisterid}	DELETE	Eliminar un programa de registro	El id de un programa de registro	Mensaje de confirmación
/program-registers/with-details	POST	Crear un programa de registro con detalle de programa	Id del programa de registro	El programa de registro con sus detalles
/forms	GET	Listar todos los formularios	No se envían datos	Listado de los formularios
/forms	POST	Crear un formulario	Datos del formularios con secciones y preguntas	El formulario creado

/forms/{formid}	PUT	Actualizar un formulario	Id del formulario con los datos nuevos	El formulario modificado
/forms/{formid}	GET	Obtener un formulario	Id del formulario	El formulario con sus secciones y preguntas
/forms/{formid}	DELETE	Eliminar un formulario	El id de un formulario	Mensaje de confirmación
/observations	GET	Obtener todas las observaciones	No se envían datos	Listado de observaciones
/observations	POST	Crear una observación	Datos de una observación	L observación creada
/observations/{observationid}	PUT	Actualizar una observación	El id de la observación con los datos nuevos	La observación actualizada
/observations/{observationid}	GET	Obtener una observación	El id de la observación	La observación encontrada
/observations/{observationid}	DELETE	Eliminar una observación	El id de una observación	Mensaje de confirmación
/observations/show-with-questions/{observationid}	GET	Obtener una observación con el formulario asociado	El id de la obervación	Una observación con el formulario
/observations/store-record-questions	POST	Responder un formulario	Las respuestas con las preguntas del formulario	Mensaje de confirmación
/observations/{observationid}/form-with-answers	GET	Obtener las respuestas de un formulario	El id de la observación	El formulario con sus respuestas
/template-forms	GET	Listas todas las plantillas de formulario	No se envían datos	Listado de las plantillas de formulario
/template-forms	POST	Crear una plantilla de formulario	Los datos de una plantilla de formulario	La plantilla de formulario con sus preguntas y secciones
/template-forms/{templateformid}	PUT	Actualizar una Plantilla de formulario	Id de la plantilla con los datos nuevos	La plantilla actualizada
/template-forms/{templateformid}	GET	Obtener una Plantilla de formulario	Id de la plantilla	La plantilla de formulario encontrada
/template-forms/{templateformid}	DELETE	Eliminar una Plantilla de formulario	El id de una plantilla de formulario	Mensaje de confirmación