UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Gestión y análisis de datos de actividad docente Herramienta PROInco

Informe de Proyecto de Grado presentado por

Donato Aguirre

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisor

Adriana Marotta

Montevideo, 2 de septiembre de 2025

Gestión y análisis de datos de actividad docente Herramienta PROInco por Donato Aguirre tiene licencia CC Atribución 4.0.

Agradecimientos

Agradezco profundamente a mi esposa Leticia por su apoyo incondicional y paciencia a lo largo de todo este proceso. A mi familia, por estar siempre presente con palabras de aliento y acompañamiento constante. Extiendo mi sincero agradecimiento a mi tutora, Adriana Marotta, por su guía, compromiso y valiosos aportes durante el desarrollo del proyecto. Finalmente, agradezco a Laura González por su colaboración y disposición, que resultaron fundamentales en distintas etapas del trabajo.

Resumen

El proyecto de grado surge a partir de necesidad en el Instituto de Computación (INCO), de una aplicación que simplifique la tarea de la gestión de los recursos docentes del instituto.

El equipo docente del INCO, conformado por más de 150 personas, lleva a cabo una amplia gama de actividades que se agrupan en cuatro categorías: enseñanza, investigación, gestión y extensión. La heterogeneidad y dinámica del cuerpo docente, que conforma distintos tipos de grupos de trabajo, y varían en cuanto a tipos de contratos, horas dedicadas, perfiles de puestos y grados académicos, entre otras, aumenta la complejidad de la gestión docente.

Este proyecto tiene como objetivo proporcionar una solución a la problemática de la recolección, centralización, y análisis de los datos de las actividades docentes. Para ello, se desarrolló una aplicación web para ser utilizada por los docentes involucrados en las tareas de dirección del instituto, llamada PROInco.

El sistema permite al administrador cargar los datos a centralizar desde las diversas fuentes que los originan, en forma periódica. Por ejemplo, la información sobre los cargos de los docentes puede cargarse desde un archivo proporcionado por el equipo de Recursos Humanos (RRHH) de la Facultad de Ingeniería (FING), mientras que las actividades de enseñanza del INCO se recogen de otro archivo proporcionado por la Comisión de Enseñanza. Para recolectar información sobre las actividades de gestión, el sistema permite al administrador enviar un correo electrónico a todos los docentes activos para que puedan acceder al sistema a través de un enlace temporal y declarar sus actividades.

El proyecto se desarrolló siguiendo una metodología ágil, incremental, con el fin de aportar valor y validar la solución desde etapas tempranas del desarrollo.

Los principales desafíos abordados incluyen el modelado de datos de la realidad, el relevamiento de los requerimientos para la recolección y carga de datos desde las diferentes fuentes, y la implementación de la solución utilizando exclusivamente tecnologías gratuitas o provistas por el instituto.

Se hizo especial énfasis en mantener altos estándares de mantenibilidad con el fin de facilitar la continuidad del proyecto por próximos equipos. Se prestó

particular atención a la usabilidad, entendiendo que la adopción exitosa del sistema depende en gran medida de su facilidad de uso.

A la fecha, el instituto utiliza exitosamente el sistema desarrollado y actualmente se está realizando la transferencia de todo lo desarrollado a un equipo del instituto designado por mantener la plataforma y expandir sus funcionalidades. Este proyecto ha proporcionado una solución efectiva a un problema significativo y ha trazado el camino para próximas mejoras y expansiones.

Palabras clave: Instituto de Computación, Gestión docente, Actividades de gestión, Recursos humanos, Metodología ágil, Bases de datos, PostgresQL, React, Next.js, Node.js, Adonis.js, Planillas, CSV

Siglas

1NF Primera Forma Normal. 31

2NF Segunda Forma Normal. 31

3NF Tercera Forma Normal. 31, 32

API Application Programming Interface. 19, 40, 84, 100

BCNF Forma Normal de Boyce-Codd. 31, 32

BI Business Intelligence. 69

CD Continuous Deployment. 72

CPU Central Processing Unit. 75

CSV Comma-Separated Values. 6, 10–15, 18, 31, 51, 59, 68, 85, 86

CTI Class Table Inheritance. 30

DDCs Docentes-Departamentos-Cargos. 14

DKIM DomainKeys Identified Mail. 82, 83

DMARC Domain-based Message Authentication, Reporting and Conformance. 82

DNS Domain Name System. 82

DT Dedicación Total. 11, 12, 15, 16, 24, 27, 45, 90

FING Facultad de Ingeniería. v, 6, 8, 22, 74, 87, 89

GDPR General Data Protection Regulation. 82

GUI Graphical User Interface. 93

HTTP Hypertext Transfer Protocol. 40, 41, 69, 71, 76, 79, 100, 104, 120

HTTPS Hypertext Transfer Protocol Secure. 39, 41

IaaS Infrastructure as a service. 73

IDE Integrated Development Environment. 68

INCO Instituto de Computación. v, 1, 2, 4, 5, 8, 18, 20, 22, 24, 74, 92, 93, 97, 107, 108

IP Internet Protocol. 82

JSON JavaScript Object Notation. 85, 102, 103

JWT JSON Web Token. 76

MER Modelo Entidad-Relación. 28

MR Modelo Relacional. 28, 117

MVC Model-View-Controller. 69

ORM Object-Relational Mapping. 28, 69

PaaS Platform as a Service. 73

PROInco Plataforma de Registro y Organización del Inco. 38

REST Representational State Transfer. 40

RRHH Recursos Humanos. v, 5, 6, 8, 10, 87, 89

SDK Software Development Kit. 100

SMTP Simple Mail Transfer Protocol. 84

SPF Sender Policy Framework. 82, 83

SSL Secure Sockets Layer. 41

STI Single Table Inheritance. 30

TLS Transport Layer Security. 41, 82, 93

UDP User Datagram Protocol. 103

UI Interfaz de Usuario. 93

URI Unidad de Recursos Informáticos. 88

Índice general

1. Introducción **1**
- 1.1. Contexto y motivación . . . 1
- 1.2. Objetivos del proyecto . . . 2
- 1.3. Estructura del documento . . . 4

2. Análisis del problema **5**
- 2.1. Contexto del problema . . . 5
- 2.2. Situación antes del proyecto . . . 6
- 2.3. Mapa de actores . . . 7
- 2.4. Problemas relevados . . . 8
- 2.5. Ingeniería de Requerimientos . . . 8
  - 2.5.1. Requerimientos funcionales . . . 9
  - 2.5.2. Requerimientos no funcionales . . . 17
- 2.6. Herramientas relacionadas . . . 20

3. Diseño de la herramienta **21**
- 3.1. Análisis y Diseño de la Base de Datos . . . 21
  - 3.1.1. Diseño conceptual . . . 21
  - 3.1.2. Diseño Lógico Relacional . . . 28
- 3.2. Arquitectura . . . 37
  - 3.2.1. Arquitectura a alto nivel . . . 38
  - 3.2.2. Decisiones principales . . . 38

4. Implementación **43**
- 4.1. Funcionalidades implementadas . . . 43
- 4.2. Lenguajes de programación y frameworks utilizados . . . 67
  - 4.2.1. JavaScript como principal lenguaje de programación . . . 67
  - 4.2.2. Tipado estático a través de TypeScript . . . 68
  - 4.2.3. Tecnologías utilizadas en la aplicación web . . . 68
  - 4.2.4. Tecnologías utilizadas en el servidor de aplicaciones . . . 68
- 4.3. Servidor de base de datos . . . 69
- 4.4. Procesamiento de trabajos asincrónicos . . . 69
  - 4.4.1. Redis como base de datos secundaria . . . 70

4.4.2. Implementación de flujos asincrónicos en la aplicación . . 70
4.5. Despliegue . . . 72
4.5.1. Docker . . . 72
4.5.2. Hosting en MiNube . . . 73
4.5.3. Hosting en MiNube . . . 73
4.5.4. PM2 como gestor de procesos . . . 75
4.6. Manejo de sesiones . . . 76
4.6.1. *Tokens* opacos . . . 76
4.6.2. Autenticación de administradores . . . 77
4.6.3. Autenticación de docentes . . . 80
4.7. Plataforma para el envío de emails . . . 81
4.7.1. Complejidades del envío de emails . . . 82
4.7.2. Elección del servicio externo . . . 83
4.7.3. Implementación de la integración con Mailjet . . . 84
4.8. Carga y procesamiento de archivos CSV . . . 85
4.8.1. Procesamiento de datos importados . . . 87
4.9. Visualización de Datos con Mode Analytics . . . 92

5. Calidad de la herramienta implementada 97
5.1. Verificación y validación . . . 97
5.1.1. Pruebas automáticas . . . 98
5.2. Gestión de la configuración . . . 98
5.3. Herramientas de análisis de código . . . 98
5.4. Integración Continua . . . 99
5.5. Manejo de errores de aplicación . . . 99
5.6. Recopilación y visualización de *Logs* . . . 102
5.7. Consistencia en la Interfaz de Usuario . . . 104
5.8. Documentación de la API . . . 104

6. Conclusiones y trabajo futuro 107
6.1. Trabajo futuro . . . 108

Referencias 109

A. Modelo Entidad Relación del Sistema 113

B. Modelo Relacional del Sistema 115

C. Adherencia a los Principios de "*12 Factor App*" 119

D. Detalles de implementación de la librería Bull 123

Capítulo 1

Introducción

En este capítulo se describe el origen y motivación del proyecto, se delinean los objetivos de la entrega y se explica la estructura del documento.

1.1. Contexto y motivación

Las actividades desarrolladas por los docentes en el Instituto de Computación son diversas y se organizan de diferentes maneras según el tipo de actividad. Cada docente participa en diferentes grupos de trabajo y bajo la coordinación de distintos colegas, lo que implica que un mismo docente puede estar involucrado en varias actividades diferentes, cada una con un supervisor distinto.

Existen cuatro tipos principales de actividades que los docentes llevan a cabo: enseñanza, investigación, gestión académica y extensión. Dentro de cada una de estas categorías, se desarrollan diversas actividades específicas. Por ejemplo, en el área de investigación, un docente podría estar involucrado en un proyecto, dirigiendo una tesis de maestría o haber realizado alguna publicación científica.

Históricamente, ha sido problemático para el INCO recolectar y administrar la información de las actividades de gestión, debido a su diversidad y descentralización. Varios actores de la dirección y Comisión de Instituto han expresado en repetidas ocasiones la necesidad imperiosa de centralizar la información sobre este tipo de actividades. A diferencia de por ejemplo las actividades de enseñanza, no hay una comisión en el instituto que gestione esta información.

Por otro lado, la heterogeneidad de los cargos docentes en el instituto es significativa, con distintos tipos de contratos, una amplia variedad de dedicaciones horarias, diferentes perfiles de cargo y grados académicos. Además, estas características suelen ser dinámicas, especialmente en lo que respecta a la carga

horaria de los docentes, la cual puede cambiar con frecuencia y, en algunos casos, varias veces en un corto período de tiempo.

El cuerpo docente del INCO está compuesto por más de ciento cincuenta personas. La gestión de recursos humanos en el INCO es una tarea particularmente desafiante, siendo el primer obstáculo disponer y manejar de manera eficiente la información completa sobre los docentes y sus actividades.

Mediante el desarrollo de una aplicación para la gestión de los datos de los docentes y las actividades que realizan, este proyecto aborda tres objetivos fundamentales: centralizar y almacenar de manera eficiente la información sobre las actividades de los docentes, facilitar la recuperación de datos necesarios para la operativa cotidiana, y proporcionar herramientas para el análisis de datos, apoyando así la toma de decisiones estratégicas.

Los principales desafíos de este proyecto incluyen: (i) diseñar un modelo de datos que represente de manera precisa la complejidad y magnitud de la realidad abordada; (ii) centralizar y mantener actualizada la información, requiriendo soluciones específicas para cada tipo de actividad docente; (iii) desarrollar una interfaz que facilite a los administradores la resolución de tareas operativas y toma de decisiones en base a la información del sistema; y (iv) implementar un mecanismo intuitivo y práctico que permita a los docentes reportar y mantener actualizadas sus actividades de gestión.

1.2. Objetivos del proyecto

A continuación, se detallan los principales objetivos académicos establecidos por el equipo al inicio del proyecto:

1. Desarrollar un sistema que satisfaga los requerimientos a relevar al inicio del proyecto, a través de entrevistas con docentes a cargo de la gestión de recursos humanos.
2. Diseñar y construir una base de datos de actividades docentes lo más completa posible.
3. Implementar y desplegar la aplicación en su entorno operativo real antes de la entrega final del proyecto. Este despliegue anticipado tiene como propósito permitir la validación práctica del software, asegurando que, para la entrega final, el sistema sea evaluado en términos de robustez y utilidad por los principales interesados, justificando su uso y mantenimiento a largo plazo.
4. Diseñar un software que, una vez construido, no necesite la intervención del equipo actual, permitiendo su operación y facilitando su mejora y expansión por equipos futuros.

5. Realizar la transferencia de conocimiento y accesos al equipo designado por el instituto para el mantenimiento de la plataforma.

Se acordaron los siguientes resultados esperados para la fecha de culminación del proyecto:

1. Análisis y documentación del contexto y los desafíos inherentes a la gestión de los datos de actividades docentes.
2. Modelado conceptual de la base de datos a implementar.
3. Diseño de la aplicación a implementar y de las estrategias de recolección de datos.
4. Implementación de la aplicación de gestión de datos.
5. Previsión de un sistema orientado al análisis de datos para la toma de decisiones.
6. Documentación exhaustiva del sistema para usuarios y futuros desarrolladores.

1.3. Estructura del documento

A continuación el contenido de cada sección de este informe se resume.

1. **Introducción**: En esta sección se presentó una visión general del proyecto y se establecieron los objetivos.
2. **Análisis del problema**: En esta sección se examinan los procesos operativos del INCO y el papel de sus actores clave, mencionando factores y restricciones que fue necesario tener en cuenta. Finalmente, se detallan los requerimientos funcionales y no funcionales relevados y se presenta la solución implementada.
3. **Diseño de la herramienta**: Esta sección aborda los principales desafíos de la arquitectura y justifican el modelado relacional relevado y su representación en tablas.
4. **Implementación**: Esta sección describe cómo se construyó la plataforma y cómo se lograron los atributos de calidad a través de decisiones tecnológicas.
5. **Calidad de la herramienta implementada**: Esta sección define los objetivos de calidad tanto para el producto como para el proceso y las actividades realizadas para garantizarlos. Además, se documenta la gestión de configuración y se presentan las métricas de calidad obtenidas al culminar el desarrollo.
6. **Conclusiones y trabajo futuro**: Esta sección contrasta los resultados del proyecto con los objetivos iniciales, proporcionando conclusiones, lecciones aprendidas y trabajos a futuro.

Capítulo 2

Análisis del problema

En este capítulo se describe el contexto en el que se desarrolla el proyecto, detallando los problemas identificados y se delinean los objetivos del sistema.

2.1. Contexto del problema

El INCO de la Facultad de Ingeniería es responsable, en la Universidad de la República, de la formación académica, la investigación, la extensión y el asesoramiento al medio público y privado del Uruguay en el área de la informática. (INCO, 2024)

El INCO es el instituto más numeroso en la Facultad de Ingeniería, contando con un plantel de aproximadamente ciento cincuenta docentes. El plantel docente presenta un alto grado de heterogeneidad en lo que refiere a grados académicos, dedicación horaria, perfil y responsabilidades. Adicionalmente, estas características suelen cambiar frecuentemente, especialmente la carga horaria de los docentes.

La dirección del INCO está a cargo de la Comisión de Instituto y del director del instituto. El director a su vez se apoya en un equipo de pro-directores entre los que se encuentran el Pro-director de RRHH, que se encarga de los aspectos relacionados a la gestión de recursos humanos del instituto, y el Pro-director de Enseñanza quien junto a la Comisión de Enseñanza, se encarga de coordinar las actividades de enseñanza de grado que se desarrollan en el instituto.

Muchas de las actividades que realiza el instituto se organizan en torno a Grupos Académicos, normalmente encabezados por un "Responsable".

Existen cuatro tipos principales de actividades que los docentes llevan a cabo: enseñanza, investigación, gestión académica y extensión. Dentro de cada una

de estas categorías, se desarrollan diversas actividades específicas, de manera simultánea, bajo la dirección de uno o más colegas.

Mientras la Comisión de Enseñanza lleva un registro de la asignación de docentes a actividades de enseñanza, el Pro-Director de RRHH gestiona los cargos y dedicaciones. Además, los responsables de los grupos del instituto centralizan las actividades de investigación. Sin embargo, no hay un registro formal para las actividades de gestión de los docentes. Esta falta de visibilidad sobre las actividades de gestión, que demandan una cantidad significativa de tiempo del cuerpo docente, especialmente de aquellos con mayor grado, genera una distribución desigual de responsabilidades. En la práctica, algunos docentes asumen una carga excesiva mientras que otros quedan subasignados. Por otro lado, aunque se lleva un registro de actividades de enseñanza e investigación, estos datos están dispersos en distintas planillas con formatos variados y problemas de calidad, lo que complica su procesamiento y aprovechamiento efectivo.

2.2. Situación antes del proyecto

Para asignar tareas a los docentes, la dirección del Instituto (Director, Comisión de Instituto y Pro-Director) requiere información detallada y consolidada sobre las actividades realizadas por los mismos. Sin embargo, el proceso es altamente fragmentado y manual, dificultando la obtención de una visión completa y unificada sobre cada docente.

La información sobre actividades de enseñanza es proporcionada por la Comisión de Enseñanza, que mantiene una base de datos con las asignaciones de los docentes a asignaturas de grado. Esta información se exporta en tres archivos Comma-Separated Values (CSV): uno contiene datos básicos de los docentes, como sus nombres y cédulas; otro, los nombres y códigos de las unidades curriculares (asignaturas); y un tercero detalla las asignaciones de los docentes a las asignaturas, indicando el año, semestre y rol correspondiente.

Por otro lado, los datos administrativos, como el grado académico, la carga horaria, y el tipo de dedicación (total, compensada o especial), son gestionados por el equipo de RRHH de la FING (conocido como "Personal"). Esta información se gestiona en un sistema que contiene un historial extenso de eventos relacionados con los docentes, como toma de posesión de cargos, extensiones y reducciones horarias, y ceses. El sistema permite exportar los datos a un archivo *INCOHIST.dbf*, que luego debe ser convertido a formato CSV para su uso por un humano. Este sistema frecuentemente presenta inconsistencias y errores en los datos que nunca han sido corregidos, debiendo ser gestionadas manualmente por quienes utilizan esta información, en este caso, los integrantes de la dirección.

Adicionalmente, identificar las actividades de gestión, investigación y extensión en las que participa un docente es aún más complejo, ya que no existe un

registro formal centralizado. En estos casos, es necesario recurrir a consultas directas con los docentes, sus pares, o solicitar información a las Comisiones correspondientes.

En resumen, cada vez que se busca determinar las actividades realizadas por un docente o identificar a aquellos con disponibilidad para asumir nuevas responsabilidades, se debe recopilar y validar datos de múltiples archivos provenientes de diferentes fuentes, procesarlos y analizarlos manualmente.

2.3. Mapa de actores

La identificación de las partes interesadas es fundamental para comprender la realidad del entorno y garantizar que la solución propuesta sea factible, efectiva y aborde los problemas de manera óptima. En la Figura 2.1, se presenta un mapa de involucrados que describe las interacciones entre ellos previo a la implementación del sistema.

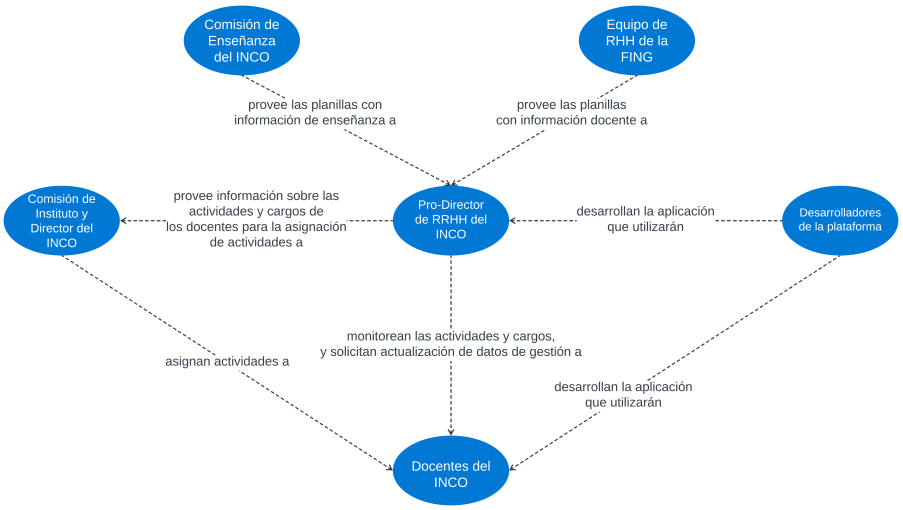


Figura 2.1: Mapa de actores

2.4. Problemas relevados

Tras analizar el contexto y la operativa, se identificaron los siguientes problemas a ser abordados:

1. No existe un registro formal de las actividades de gestión realizadas por los docentes del INCO, lo que dificulta identificar cuáles están sobreasignados y cuáles subasignados.
2. Responder a la pregunta "¿Cuáles son las responsabilidades actuales de un docente?" requiere consultar múltiples fuentes (planillas, correos electrónicos o directamente al docente), lo que implica una inversión significativa de tiempo.
3. No se dispone de un historial de las actividades realizadas por cada docente, que permita realizar análisis orientados a la toma de decisiones.
4. Debido a que la mayoría de los datos se almacenan en planillas o archivos de texto, estos son susceptibles a pérdida y cuentan con una baja calidad de datos (fechas inválidas, filas duplicadas, información errónea, etc.) como se analizará más adelante.

2.5. Ingeniería de Requerimientos

Para relevar los requerimientos y diseñar la herramienta, se realizaron múltiples entrevistas con las responsables de RRHH del INCO: Adriana Marotta, actual Pro-Directora, y Laura González, quien ocupó este cargo anteriormente.

Durante las sesiones con Laura, se mencionó la existencia de un modelo relacional previamente desarrollado que abarcaba parte de la realidad abordada en este proyecto. Se tuvo acceso a un modelo conceptual general, el cual no contemplaba todos los aspectos de la realidad que se buscaba representar. Dicho modelo se tomó como insumo para el desarrollo de nuestro modelo.

Algunos datos claves obtenidos durante entrevistas, que fueron considerados a la hora de determinar los requerimientos funcionales y no funcionales fueron los siguientes:

- Los sistemas utilizados por el equipo de RRHH de la FING no se encuentran accesibles para que el personal del INCO pueda hacer consultas o modificaciones. En consecuencia, la carga de datos deberá realizarse a través de archivos.
- Los docentes con mayor trayectoria suelen participar en numerosas actividades de gestión, muchas de ellas en repetidas ocasiones a lo largo del tiempo. Para simplificar este proceso, la plataforma debe facilitar la reutilización de información previamente ingresada, evitando la repetición

innecesaria. Por ejemplo, debe permitir buscar y seleccionar actividades de gestión o instituciones ya registradas, ya sea por el mismo docente u otros, agilizando la carga de datos.

Para validar el modelo de actividades de gestión, se distribuyó un formulario a través de *Google Forms* (Google, 2024b) a un grupo reducido de docentes. El objetivo era verificar el formato de los datos y optimizar el proceso de recolección antes de realizar la implementación.

Adriana Marotta seleccionó a cinco participantes para este piloto, de los cuales tres completaron el formulario. La elección se realizó considerando su interés en el tema y su participación activa en actividades de gestión, con el fin de garantizar información representativa.

Estrategia de recolección de actividades de gestión

Para la recolección de las actividades de gestión se evaluaron distintas alternativas y se optó por utilizar un formulario web accesible a través del correo electrónico de solicitud por las siguientes razones:

- Se desea facilitar al máximo el proceso para las docentes, incentivando así su participación y el aporte de información.
- Por el momento, se pretende evitar la gestión de un usuario individual para cada docente.
- No se quiere delegar al docente la responsabilidad de recordar acceder a la plataforma y mantener actualizada su información.

2.5.1. Requerimientos funcionales

A continuación se listan los requerimientos funcionales para cada actor de la plataforma.

Administradores

RF 1 - Inicio de sesión

La plataforma web debe permitir el acceso a los usuarios administradores mediante un proceso de inicio de sesión utilizando su correo electrónico y contraseña. El primer usuario administrador será creado previamente por el equipo de desarrollo a través de la consola.

RF 2 - Cierre de sesión

La plataforma web debe permitir a los usuarios con una sesión activa realizar el cierre de sesión, de forma que otro individuo que acceda al navegador tras el cierre de sesión, deba re-introducir las credenciales para obtener acceso a la información.

RF 3 - Creación de usuarios

Los administradores deben poder crear nuevos usuarios a través de la plataforma web, especificando su correo electrónico y asignando uno de los siguientes roles: ***read*** (lectura), ***write*** (escritura) o ***admin***.

La creación de nuevos administradores es esencial debido a la naturaleza rotativa de la Pro-Dirección, así como para permitir el acceso a otros actores relevantes, como el Director del Instituto.

Esta funcionalidad está restringida a usuarios con el rol de ***admin***.

RF 4 - Listado de usuarios

Los administradores deben poder ver un listado de todos los usuarios administradores de la plataforma, pudiendo visualizar el email, rol, fecha de creación y fecha de última actualización de cada usuario.

Esta funcionalidad está restringida a usuarios con el rol ***admin***.

RF 5 - Modificación de usuarios

Los administradores deben tener la capacidad de modificar el correo electrónico y el rol de los administradores. Los administradores no deben poder modificar su propio rol.

Esta funcionalidad está restringida a usuarios con el rol ***admin***.

RF 6 - Eliminación de usuarios

Los administradores deben tener la capacidad de eliminar administradores. No obstante, un administrador no puede eliminar su propio usuario.

Esta funcionalidad está restringida a usuarios con el rol ***admin***.

RF 7 - Carga de docentes

Los administradores deben poder cargar docentes al sistema desde un archivo CSV. El archivo debe contener las columnas *CORREO* (email), *CEDULA*, *NOMBRE*, *TELEFONO*, *INGDOC* (fecha de ingreso) y *FECHANAC* (fecha de nacimiento).

Esta funcionalidad permite importar rápidamente en masa a los nuevos docentes que son contratados por el instituto a través de las planillas provistas por el equipo de RRHH.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 8 - Carga de email institucional de docentes

Los administradores deben poder cargar el email institucional de los docentes al sistema desde un archivo CSV. El archivo debe contener las columnas "CEDULA" y "EMAIL".

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 9 - Creación de docentes

Los administradores deben poder crear nuevos docentes a través de la plataforma web. Al registrar un docente, se deben especificar los nombres, apellidos, cédula, fecha de ingreso y fecha de nacimiento. Opcionalmente, se pueden incluir el email personal, email institucional, número de oficina, número de teléfono y su máximo nivel de formación.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 10 - Listado de docentes

Los administradores deben poder ver un listado de docentes en la plataforma web. El listado debe incluir cédula, nombres, apellidos, Dedicación Total (DT), grado, tipo de cargo (efectivo, interino, etc.) y carga horaria (semanal) de cada docente.

RF 11 - Búsqueda de docentes

Los administradores deben poder buscar docentes por nombre, apellido y cédula en la plataforma web. La búsqueda debe ser insensible a mayúsculas y minúsculas y debe soportar coincidencias parciales.

RF 12 - Ordenado de la lista de docentes

Los administradores deben poder ordenar la lista de docentes en la plataforma web por apellidos, presencia de DT, grado, tipo de cargo y carga horaria. La lista debe soportar la ordenación por múltiples campos, permitiendo seleccionar la dirección (ascendente o descendente) para cada uno de ellos. Por defecto la lista se ordena por apellidos de manera ascendente.

RF 13 - Filtrado de la lista de docentes

Los administradores deben poder filtrar la lista de docentes en la plataforma web por grado, DT y estado (***activo*** o ***inactivo***).

RF 14 - Descarga de la lista de docentes en formato CSV

Los administradores deben poder descargar la lista de docentes en formato CSV desde la plataforma web. El archivo CSV debe incluir la cédula, apellidos, nombres, email, DT, grado, tipo de cargo y carga horaria semanal de cada docente. De haberse aplicados, la descarga debe respetar los filtros y el orden seleccionados. Esta funcionalidad permite al administrador identificar grupos de docentes que cumplen ciertos criterios. Por ejemplo,

obtener la lista de docentes de grado 3 con DT y revisar la carga horaria de cada uno.

RF 15 - Modificación de docentes

Los administradores deben poder modificar la información de los docentes existentes en la plataforma web.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 16 - Eliminación de docentes

Los administradores deben poder eliminar docentes existentes en la plataforma web para poder corregir un error luego de una inserción manual o por importación. Sin embargo, un docente sólo puede ser eliminado si no está vinculado a ninguna actividad o cargo, porque esto significa que es un docente real y se debe preservar todo histórico.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 17 - Carga de asignaturas

Los administradores deben poder cargar asignaturas al sistema desde un archivo CSV. El archivo debe contener las columnas *CODIGO* (de bedelías), *NOMBRE*, y *TIPO* (obligatoria o electiva)

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 18 - Creación de asignaturas

Los administradores deben poder crear nuevas asignaturas a través de la plataforma web. Al crear una asignatura, es necesario especificar el nombre, el código de bedelías, si es obligatoria u optativa, y si es de postgrado. Opcionalmente, se le puede asignar un acrónimo.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 19 - Listado de asignaturas

Los administradores deben poder ver un listado de asignaturas en la plataforma web. El listado debe incluir código de bedelías, acrónimo, nombre y tipo (electiva u obligatoria) de cada asignatura.

RF 20 - Modificación de asignaturas

Los administradores deben poder modificar la información de las asignaturas existentes en la plataforma web. El código de bedelías solo puede ser modificado si el mismo no está siendo usado como identificador de la asignatura para una asignación.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 21 - Eliminación de asignaturas

Los administradores deben poder eliminar asignaturas existentes en la plataforma web. Sin embargo, una asignatura sólo puede ser eliminada si no está vinculada a ninguna actividad docente.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 22 - Carga de actividades de enseñanza

Los administradores deben poder cargar actividades de enseñanza en el sistema desde un archivo CSV. El archivo debe incluir las columnas *CI* (cédula), *ASIGNATURA*, *ANIO*, *SEMESTRE* (par o impar), y *ROL* (***responsable***, ***subresponsable*** o ***integrante***). Antes de cargar las actividades de enseñanza, es necesario cargar los docentes y las asignaturas.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 23 - Listado de actividades

Los administradores deben poder ver un listado de actividades para un docente particular en la plataforma web. El listado debe incluir el nombre de la actividad, horas mensuales (dedicadas a la misma por el docente), estado (en curso o finalizada), fecha de inicio, fecha de fin, tipo (gestión, enseñanza, investigación, extensión) y el rol del docente en cada actividad.

RF 24 - Filtrado de la lista de actividades

Los administradores deben poder filtrar la lista de actividades de un docente en la plataforma web por tipo, ya sea ***gestión***, ***enseñanza***, ***investigación*** o ***extensión***.

RF 25 - Modificación de actividades de enseñanza

Los administradores deben poder modificar horas mensuales y rol de las actividades de enseñanza de un docente en la plataforma web.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 26 - Modificación de actividades de gestión

Los administradores deben poder modificar horas semanales, rol, fecha de inicio y fecha de fin de las actividades de gestión de un docente en la plataforma web.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 27 - Eliminación de actividades

Los administradores deben poder eliminar actividades de un docente en la plataforma web.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 28 - Creación de cargos

Los administradores deben poder crear nuevos cargos a través de la plataforma web. Al crear un cargo, es necesario especificar el grado (1 a 5), y el tipo (***contrato***, ***efectivo***, ***honorario***, ***interino)***. Adicionalmente durante la carga de Docentes-Departamentos-Cargos (DDCs), si el cargo no existe debe ser creado automáticamente.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 29 - Listado de cargos

Los administradores deben poder ver un listado de cargos en la plataforma web. El listado debe incluir el tipo de cargo (***contrato***, ***efectivo***, ***honorario***, ***interino***) y el grado (1, 2, 3, 4 o 5) asociado a cada cargo.

RF 30 - Eliminación de cargos

Los administradores deben poder eliminar cargos existentes en la plataforma web. Sin embargo, un cargo sólo puede ser eliminado si no está vinculado a ningún docente.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 31 - Carga de cargos docentes (DDCs)

Los administradores deben poder cargar los cargos desempeñados por los docentes en el sistema desde un archivo CSV. El archivo debe incluir las columnas *NOMBRE*, *CEDULA*, *GRADO*, *HORAS* (dedicación semanal), *PROVISION* (nuevo cargo, renovación, cese, etc.), *DENOMINACION* (descripción del cargo), *POSESION* (fecha de posesión del cargo) y *CESE* (fecha de cese del cargo).

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 32 - Creación de cargo docente

Los administradores deben poder crear cargos para un docente particular a través de la plataforma web. Al crear el mismo, es necesario especificar el cargo, la fecha de inicio y fin, la última renovación y la carga horaria. Opcionalmente, se puede agregar una descripción.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 33 - Listado de cargos de un docente

Los administradores deben poder ver un listado de cargos para un docente particular en la plataforma web. El listado debe incluir el departamento, tipo, grado, carga horaria semanal, fecha de inicio, fecha de fin y última renovación de cada cargo.

RF 34 - Modificación de cargo docente

Los administradores deben poder modificar la información del cargo de un docente en la plataforma web. Las fechas solo pueden modificarse de manera que el resultado no se solape con otra instancia para el mismo docente.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 35 - Eliminación de cargo docente

Los administradores deben poder eliminar instancias de la relación entre docentes y cargos existentes en la plataforma web.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 36 - Carga de DT

Los administradores deben poder cargar DTs asociadas a docentes en el sistema desde un archivo CSV. El archivo debe incluir las columnas *NOMBRE*, *CEDULA*, *PROVISION* (nueva DT, renovación, cese, etc.), *POSESION* (fecha de posesión de la DT) y *CESE* (fecha de cese de la DT).

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 37 - Creación de DT

Los administradores deben poder crear nuevas DTs para un docente particular a través de la plataforma web. Al crear una DT, es necesario especificar la fecha de inicio, fecha de fin, y la última renovación.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 38 - Listado de dedicaciones totales (DTs)

Los administradores deben poder ver un listado de DTs para un docente particular en la plataforma web. El listado debe incluir fecha de inicio, fecha de fin y última renovación de cada DT.

RF 39 - Modificación de DT

Los administradores deben poder modificar la información de DTs existentes en la plataforma web. Las fechas solo pueden modificarse de manera que el resultado no se solape con otra DT para el mismo docente.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 40 - Eliminación de DT

Los administradores deben poder eliminar DTs existentes en la plataforma web.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 41 - Solicitud de actualización de actividades de gestión a través de email

Los administradores deben poder solicitar a los docentes que actualicen la información sobre sus actividades de gestión en la plataforma web. La solicitud se realiza mediante el envío automático de emails, de manera individual o en masa, esto es, a todos los docentes.

Esta funcionalidad está restringida a usuarios con el rol ***write*** o ***admin***.

RF 42 - Gráfica de distribución de grados a través del tiempo

Los administradores deben poder ver un gráfico de barras apiladas que muestre la cantidad de docentes de cada grado por año.

RF 43 - Gráfica de distribución de horas por grado y tipo

Los administradores deben poder ver un gráfico de barras apiladas que muestre las horas dedicadas a actividades de gestión y enseñanza por grado, para el año seleccionado.

RF 44 - Gráfica de distribución de horas por DT y tipo

Los administradores deben poder ver un gráfico de barras apiladas que muestre las horas dedicadas a actividades de gestión y enseñanza por DT (con y sin DT), para el año seleccionado.

RF 45 - Reportes

Los administradores deben tener acceso a una plataforma de *reporting* ya configurada, que demuestre el potencial de los datos recolectados y ofrezca herramientas de visualización y análisis. Además, se debe proporcionar un reporte con información histórica sobre las actividades de gestión.

Docentes

RF 46 - Resumen de actividades de gestión

Los docentes deben recibir solicitudes de actualización de la información sobre sus actividades de gestión vía email. El email debe incluir un listado de las actividades de gestión en curso o futuras registradas para el docente, así como un enlace de acceso a la plataforma.

RF 47 - Acceso a la plataforma web

Los docentes deben poder acceder a la plataforma web a través de un enlace incluido en el email de solicitud de actualización de actividades docentes. Este acceso debe permitirles únicamente consultar, crear y modificar sus actividades de gestión y el mismo debe expirar luego de tres meses.

RF 48 - Cierre de sesión

La plataforma web debe permitir a los docentes con una sesión activa realizar el cierre de sesión.

RF 49 - Creación de actividad de gestión del docente

Los docentes deben poder crear nuevas actividades de gestión a través de la plataforma web. Al crear una actividad de gestión, es necesario especificar la actividad (seleccionando una existente o creando una nueva), el rol, horas mensuales, tipo de participación (***titular*** o ***suplente***), si fue designado por votación o no, fecha de inicio y fecha de fin.

RF 50 - Listado de actividades de gestión del docente

Los docentes deben poder ver un listado de sus actividades en la plataforma web. El listado debe incluir el nombre de la actividad, horas mensuales dedicadas por el docente, estado (***en curso*** o ***finalizada***), fecha de inicio, fecha de fin, tipo (***gestión***, ***enseñanza***, ***investigación*** o ***extensión***) y el rol del docente en cada actividad.

RF 51 - Modificación de actividad de gestión del docente

Los docentes deben poder modificar horas mensuales, fecha de inicio y fecha de fin de sus actividades de gestión en la plataforma web.

RF 52 - Eliminación de actividad de gestión del docente

Los docentes deben poder eliminar sus actividades de gestión en la plataforma web.

- Búsqueda por nombre o institución y alta de Actividad de Gestión
- Búsqueda (por nombre) y alta de institución

2.5.2. Requerimientos no funcionales

A continuación se presentan los requerimientos no funcionales relevados.

RNF 1 - Utilización de software gratuito

El sistema debe poder desarrollarse y mantenerse en operación de manera completamente gratuita o utilizando recursos provistos por el instituto.

RNF 2 - Aplicación web

El sistema debe ser accesible principalmente desde un cliente web en una computadora de escritorio o *laptop*, dado que los usuarios lo utilizarán en el contexto de sus actividades profesionales, donde suelen trabajar con una computadora.

RNF 3 - Datos almacenados en Uruguay

Los servidores de base de datos se deben encontrar dentro del país, para garantizar la seguridad de los mismos y la velocidad de acceso.

RNF 4 - Postgres como Base de datos relacional

La base de datos a utilizar debe ser relacional y de código libre, en particular PostgreSQL.

RNF 5 - Interoperabilidad y futura integración con otros sistemas

Dado que el sistema almacenará datos únicos como las actividades de gestión, debe estar diseñado para facilitar futuras integraciones con otros sistemas que INCO pueda desarrollar o adoptar. Esto implica que la arquitectura debe ser lo suficientemente flexible para permitir modificaciones que faciliten la exposición de datos a través de nuevas APIs u otros mecanismos de integración.

RNF 6 - Encriptación de los datos

Los datos sensibles como contraseñas de los usuarios deben ser encriptadas.

RNF 7 - Usuarios concurrentes

El sistema debe ser capaz de gestionar al menos doscientos docentes activos de manera simultánea, lo que se estima generará una carga de aproximadamente trescientas *requests* por minuto en los *endpoints* destinados a la consulta y actualización de información sobre las actividades docentes.

RNF 8 - Procesamiento de planillas

El sistema debe permitir el procesamiento de planillas CSV con hasta diez mil filas sin ocasionar errores.

RNF 9 - Monitoreabilidad

El sistema debe alertar automáticamente al equipo desarrollador en el momento que ocurran defectos críticos en la aplicación, como respuestas con código 500 o una excepción no manejada.

RNF 10 - No *spamming*

Las funcionalidades que envíen correos electrónicos a muchos docentes en una sola acción deben tener medidas para asegurar que no se envíe más de un correo en un lapso de cuarenta y ocho horas al mismo docente, incluso si ocurre una falla en el mecanismo de reintento o si el administrador presiona dos veces seguidas el botón de enviar solicitudes de actualización.

RNF 11 - Tecnologías actuales

La plataforma debe estar desarrollada con tecnologías de código abierto que sean activamente mantenidas para promover la sostenibilidad a lar-

go plazo del sistema, permitiendo aprovechar una comunidad activa para soporte y mejoras, y evitando la dependencia de tecnologías obsoletas o descontinuadas que puedan comprometer el funcionamiento o la seguridad de la plataforma.

RNF 12 - Lenguajes fuertemente tipados

Toda la plataforma debe estar codificada utilizando lenguajes fuertemente tipados para contribuir a una mejor mantenibilidad y prevención de errores.

RNF 13 - Mantenibilidad del código

Tanto la aplicación web como la *backend* Application Programming Interface (API) deben tener verificaciones automatizadas que garanticen el seguimiento de los estándares de calidad del código y buenas prácticas definidos.

RNF 14 - Integración continua

Tanto la aplicación web como la *backend* API deben verificar que las pruebas automatizadas estén pasando cada vez que se integra código a la rama principal.

RNF 15 - Documentación de la API

Todos los *endpoints* de la *backend* API deben estar apropiadamente documentados en el formato OpenAPI (SmartBear Software, 2024b) para su posterior mantenimiento por futuros desarrolladores y desarrolladores que trabajen en el sistema desarrollado o busquen crear una integración con otro sistema.

RNF 16 - Autenticación *Passwordless* para los Docentes

Ya que solicitar a los docentes que recuerden un usuario y contraseña para un servicio que, en promedio, utilizarán solo una vez por semestre afecta negativamente la experiencia de usuario, la autenticación debe implementarse mediante un método alternativo que sea más práctico y sencillo.

RNF 17 - Usabilidad para el docente

Dado que se requiere de la buena voluntad de los docentes para completar la información de actividades de gestión, el sistema debe estar diseñado para ofrecer una experiencia altamente intuitiva y amigable. El objetivo es minimizar la fricción durante el proceso de ingreso de datos, aumentando así la probabilidad de que los docentes participen voluntariamente.

RNF 18 - Cuota de correos electrónicos

El sistema debe garantizar la entrega efectiva de al menos doscientos correos electrónicos por día y cuatrocientos totales en un mes. Esto se basa en la estimación de que la base de datos contendrá entre ciento cincuenta y doscientos docentes activos en todo momento, a quienes se les enviará

la solicitud de actualización de información hasta un máximo de dos veces al mes.

RNF 19 - Confiabilidad de los correos electrónicos

El sistema debe garantizar que los correos electrónicos enviados lleguen a la bandeja de entrada de los destinatarios y no sean marcados como *spam*. Este objetivo se considera alcanzado siempre que no intervengan factores externos como filtros personalizados configurados por los destinatarios, catalogación previa de correos del remitente como *spam*, o configuraciones específicas del cliente de correo.

2.6. Herramientas relacionadas

Si bien existen plataformas como Float (Pixel Paddock Ltd., 2024), cuyos planes inician en U$S 10 por usuario al mes, Monday (monday.com Ltd., 2024) con planes desde U$S 9 por usuario al mes, o Asana (Asana Inc., 2024) con planes desde U$S 10.99 por usuario al mes, las mismas no se adaptan completamente a las particularidades del instituto.

La propuesta de valor de Float es mejorar la eficiencia operativa. Proporciona herramientas para la planificación de proyectos, la asignación de recursos, el registro de horas trabajadas y el seguimiento del presupuesto, que no son relevantes para el instituto. Aunque incluye algunas funcionalidades interesantes, como la generación de informes y alertas de disponibilidad (sub-asignación y sobre-asignación), las mismas están dirigidas a organizaciones con fines de lucro, por lo que no se adaptan a la realidad del INCO.

Asana y Monday, por su parte, se enfocan más en la gestión de proyectos y tareas. Si bien ambas ofrecen herramientas de gestión de recursos humanos, las mismas son genéricas y no permiten registrar datos importantes, como el historial de cargos de un docente.

En particular, todas las plataformas analizadas carecen de una funcionalidad que permita a cada docente cargar o actualizar su información, lo cual es clave. Tampoco son capaces de representar los diferentes tipos de actividades ni de capturar información adyacente, como los cargos y DTs.

Dado que existen requerimientos funcionales que no están cubiertos por las plataformas existentes y que el RNF 1 - Utilización de software gratuito exige que el costo operativo sea nulo, resulta necesario desarrollar una solución a medida.

Capítulo 3

Diseño de la herramienta

Este capítulo presenta el diseño de la herramienta desarrollada, enfocándose en el análisis de la realidad del problema. Se describen las entidades identificadas y sus relaciones para cumplir con los requerimientos establecidos, así como el proceso de transformación del modelo entidad-relación al esquema relacional implementado en PostgreSQL, incluyendo las técnicas de normalización, restricciones de integridad e indexación aplicadas.

3.1. Análisis y Diseño de la Base de Datos

En esta sección se describen y justifican las entidades y relaciones identificadas durante la fase de análisis para permitir el cumplimiento de los requerimientos del sistema.

3.1.1. Diseño conceptual

En el Anexo A, se presenta el modelo entidad relación completo del sistema. A continuación, a lo largo de esta sección, se describe y justifica el modelo de a partes para facilitar la compresión.

En primer lugar, se determinó que el sistema debe autorizar el acceso a la información únicamente a usuarios previamente identificados, por lo que se modeló la entidad “usuario” quien cuenta con un correo electrónico y una contraseña. Dado que algunos usuarios pueden ser además docentes, una instancia de usuario puede estar relacionada con una instancia de docente, como se puede apreciar en la Figura 3.1.

Para implementar distintos niveles de acceso a la plataforma se identificaron un conjunto de roles. Cada instancia de usuario tiene un rol. Del rol interesa

registrar su nombre, denominados ***admin***, ***read*** y ***write***.

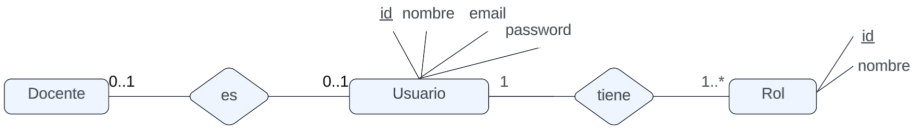


Figura 3.1: Entidad Usuario y Rol

La FING se organiza en ***institutos*** que a su vez comprenden diversos ***departamentos***, siendo el INCO uno de ellos y el contexto específico de este proyecto. La Figura 3.2 ilustra los conceptos vinculados a docentes y sus cargos.

El ***docente*** es la entidad clave en nuestra realidad. En particular interesa registrar la cédula, ya que se utiliza para identificar al docente a la hora de importar datos de las diferentes planillas que maneja el sistema. También es de suma importancia su correo electrónico institucional ya que a través del mismo se realizan los pedidos de actualización de datos.

Los docentes tienen un ***cargo*** asociado que determina el tipo de actividades que se espera que realicen y su dedicación. Del cargo interesa registrar, el tipo (interino, efectivo, contrato u honorario), y el grado (1 a 5) el cual establece la jerarquía.

En cada departamento, un docente ejerce un único cargo. De esta relación se registra la carga horaria y las fechas de inicio, renovación y finalización, considerando que los cargos poseen carácter temporal. Cuando un docente experimenta un ascenso de grado o una modificación en el tipo de cargo (por ejemplo, la transición de interino a efectivo), se procede a cerrar el registro vigente mediante la actualización de su fecha de finalización y se establece un nuevo registro para el cargo modificado.

Las instancias de la relación ***ejerce*** pueden estar relacionadas con ***reducciones horarias***, las cuales implican una disminución en la carga horaria general del docente. De estas reducciones se registra la fecha de inicio y fin, así como las horas correspondientes a la carga horaria durante el período de reducción. A diferencia de las reducciones, las ***extensiones horarias*** se modelan de manera diferente ya que están vinculadas directamente a actividades.

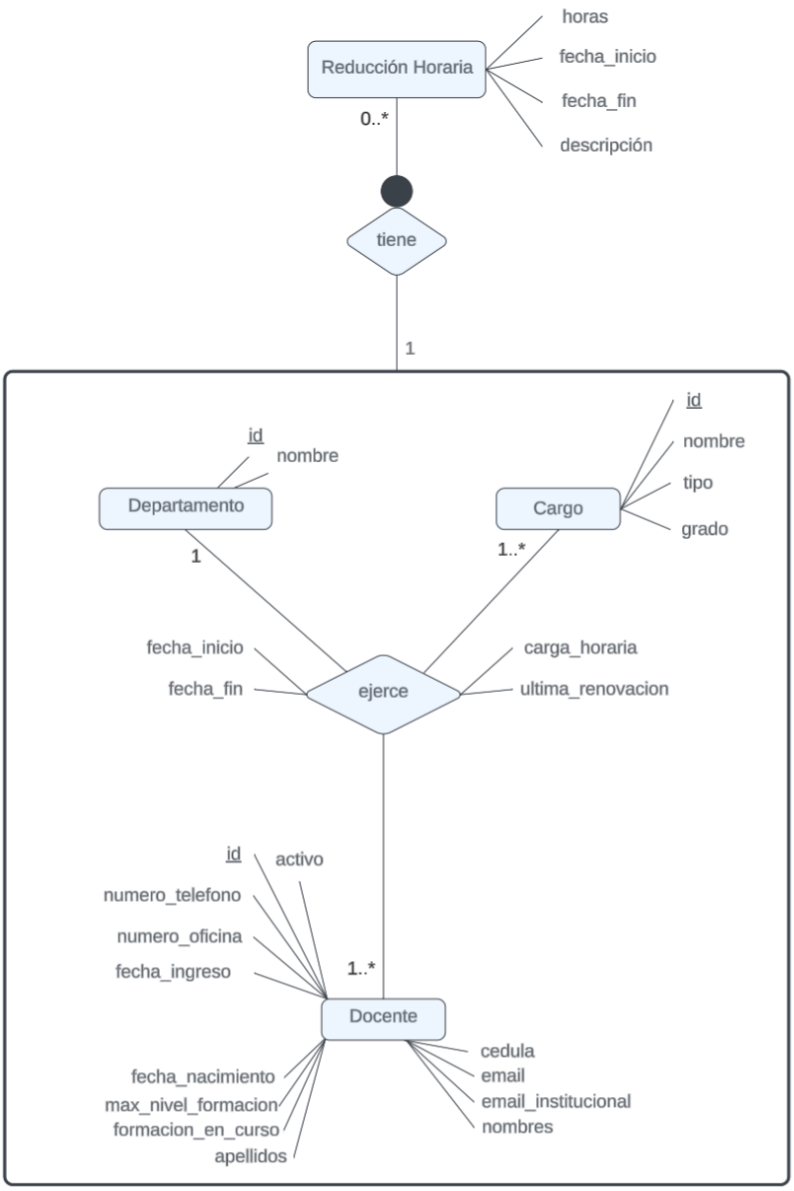


Figura 3.2: Entidades Docente, Departamento, Cargo y Reducción Horaria

Los docentes se relacionan con cuatro entidades que desarrollaremos a continua-

ción, ***DT***, ***licencias***, ***grupos académicos*** y ***actividades***.

La ***dedicación total*** es una remuneración adicional que tienen algunos docentes para recompensar que se dedican exclusivamente a la docencia (no trabajan en otros lugares) y tienen una carga horaria alta (30 horas semanales o más). La DT se otorga por un periodo dado y en caso de extenderse debe ser renovada. Por lo tanto interesa registrar la fecha de inicio, renovación y fin. Un docente puede tener varias dedicaciones totales en caso de existir interrupciones, pero las mismas no pueden superponerse. Es importante destacar que la DT, junto con el cargo, es un insumo importante a la hora de asignar actividades de gestión. Ver Figura 3.3.

Las ***licencias*** representan las ausencias que toma el docente, de las mismas interesa registrar las fechas de solicitud, resolución, inicio y fin, el tipo (reglamentaria, enseñanza, especial, sin goce de sueldo, sabático) y el estado (pendiente, aprobada, rechazada).

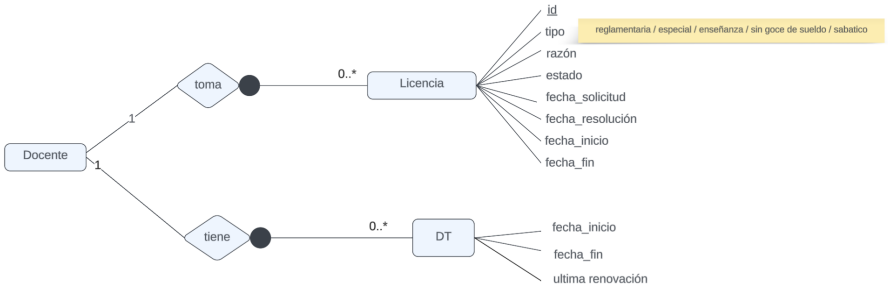


Figura 3.3: Entidades Licencia y DT

El INCO está conformado por ***grupos de investigación***, cada uno integrado por uno o más docentes. De estos grupos se registra el nombre que los identifica y su acrónimo correspondiente. En la relación entre docente y grupo académico, se documenta si el docente ejerce la responsabilidad de liderar dicho grupo. Ver Figura 3.4

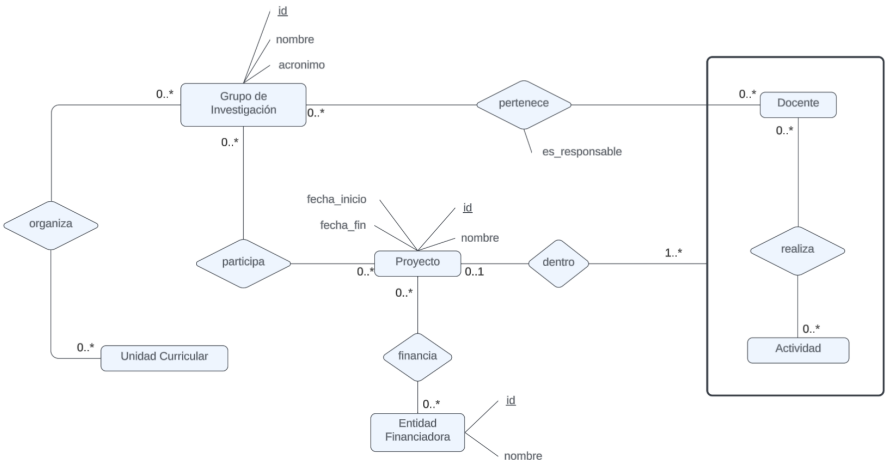


Figura 3.4: Entidades Grupos Académicos y Proyectos

Finalmente, otra entidad clave son las ***actividades***, las mismas representan las tareas o responsabilidades que el docente realiza en su día. En nuestro modelo las actividades se definen mediante una especialización, dado que existen cuatro tipos de actividades: enseñanza, gestión, investigación y extensión. De la relación ***realiza*** interesa registrar la fecha de inicio y fin, las horas semanales dedicadas por el docente a la misma y el rol del docente (***director***, ***integrante***, ***responsable*** o ***subresponsable***), como se ilustra en la Figura 3.5.

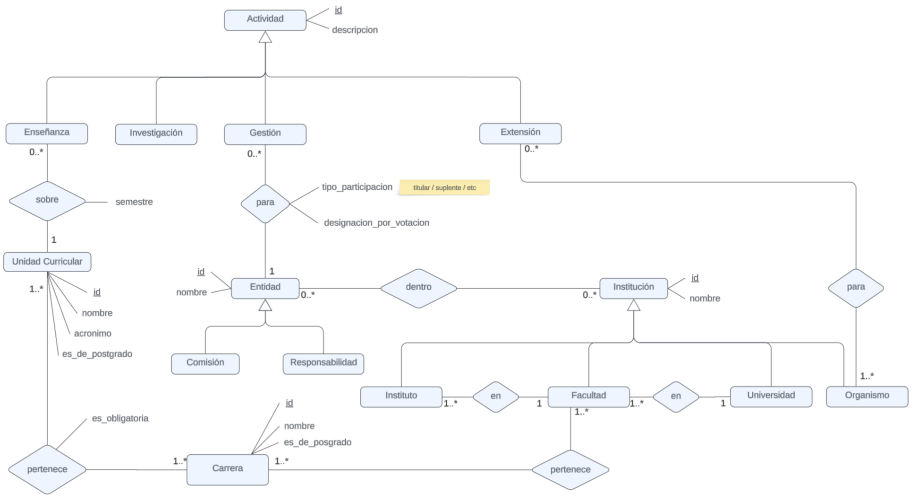


Figura 3.5: Entidad Actividades

La dirección del instituto registra las ***actividades de investigación*** y las ***actividades de extensión*** en una planilla Excel que se presenta como reporte al decanato de la facultad. Entre las tareas de investigación se incluyen la ***participación en proyectos***, ***seminarios*** y ***producción científica***. Las actividades de extensión generalmente se encuentran vinculadas a un ***organismo*** específico.

Los organismos constituyen especializaciones de la entidad ***institución***, de la cual se distinguen cuatro tipos: ***universidad***, ***facultad***, ***instituto*** y ***organismo***.

Las actividades de gestión se vinculan con la ***entidad*** donde se ejecutan, ya sea una ***comisión*** o una ***responsabilidad***. De esta relación se registra el tipo de participación del docente (***titular*** o ***suplente***) y si su designación se realizó mediante votación.

Una ***actividad de gestión*** relacionada con ***comisión*** se refiere a que el docente es integrante de una comisión. Si la actividad está relacionada con ***responsabilidad***, significa que esa actividad de gestión es una responsabilidad individual, como por ejemplo, “Director de Instituto”, “Director de Carrera”, “Pro-director de Enseñanza”, etc.

Las ***actividades de enseñanza*** corresponden al dictado de unidades curriculares. Estas actividades se asocian naturalmente con una unidad curricular específica, documentándose el semestre en el cual se desarrollan.

De las ***unidades curriculares*** se registra el nombre, acrónimo y si pertenecen al nivel de postgrado. Estas unidades son ***organizadas*** por uno o más grupos académicos.

La relación ***realiza*** (entre docente y actividad) se relaciona con dos entidades adicionales: ***dedicación*** y ***extensión horaria***, como se ilustra en la Figura 3.6.

Las ***extensiones horarias*** se otorgan por actividades de distintos tipos y eventualmente el docente regresa a la carga horaria original una vez concluida la actividad. Las extensiones cambian frecuentemente y por ende la dedicación de los docentes también es muy cambiante. En un mismo semestre se pueden tener varías extensiones horarias. De la misma interesa registrar horas, fecha de inicio y fin, y opcionalmente una descripción.

La ***dedicación*** se modela mediante una especialización dado que existen dos tipos (especial y compensada).

El tipo de dedicación se determina según la fuente de financiamiento, que puede ser ***presupuestal*** (recursos del presupuesto universitario asignado por el gobierno) o ***extra presupuestal*** (fondos provenientes, por ejemplo, de convenios establecidos con empresas).

La ***dedicación compensada*** es financiada con fondos presupuestales, es fija y significa un aumento del 45 % del sueldo. Normalmente se otorga por poco tiempo. Los docentes con DT no pueden cobrar dedicación compensada.

La ***dedicación especial*** es financiada con fondos extrapresupuestales, es variable (máximo 70 %) y por tanto es pertinente registrar cuál es el porcentaje.

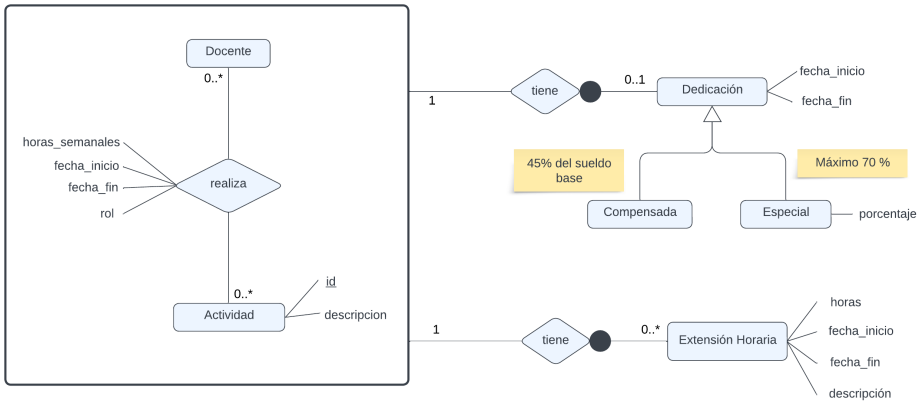


Figura 3.6: Entidades Dedicación y Extensión Horaria

Las actividades de extensión pueden ser realizadas a cualquier clase de institución (empresas, organismos o facultades). Las actividades de gestión se realizan sobre la universidad u otros organismos.

Múltiples docentes pueden ocupar el mismo tipo de cargo, por lo que la entidad ***cargos*** almacena exclusivamente la información básica. Las fechas de ocupación

del cargo se definen en la relación establecida entre ***docentes*** y ***departamentos***.

3.1.2. Diseño Lógico Relacional

En esta sección se describen las decisiones clave tomadas durante el proceso de transformación del Modelo Entidad-Relación (MER) al esquema relacional implementado en PostgreSQL. Se justifican ciertos patrones utilizados y tipos de datos utilizados, la convención de nomenclatura, y las técnicas empleadas para garantizar la integridad de los datos y el rendimiento general del sistema.

En el Anexo B, se presenta el modelo relacional completo del sistema.

Para el paso del MER al Modelo Relacional (MR), se aplicaron los principios de normalización estándar, aplicando reglas comunes para transformar entidades y relaciones en tablas.

Modelado de relaciones

En el caso de las tablas de relaciones, se optó por nombrarlas con una convención que refleja las entidades que conectan, por ejemplo, la tabla **docentes__departamentos__cargos** conecta las tablas **docentes**, **departamentos** y **cargos**, como se ilustra en la Figura 3.7. Estas tablas intermediarias incluyen una clave primaria independiente (**id**), en lugar de emplear claves primarias compuestas. Esto facilita la integración con el *framework* utilizado, que maneja de forma más eficiente claves primarias simples. Aunque el *framework* admite claves compuestas, hacerlo requiere una configuración adicional considerable, lo cual podría afectar el rendimiento y la simplicidad del código. Además, el uso de claves primarias simples, mejora la compatibilidad con ciertas características avanzadas del Object-Relational Mapping (ORM), como la gestión de asociaciones y las operaciones de actualización y eliminación en cascada.

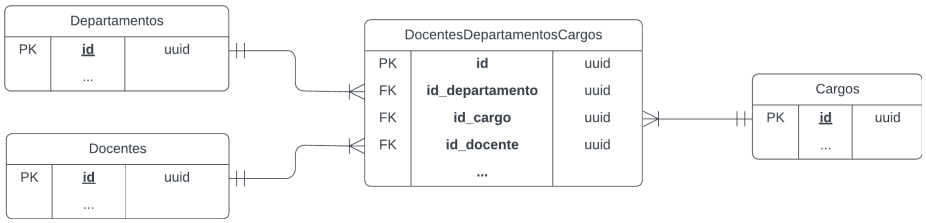


Figura 3.7: Modelado de la relación entre Docentes Departamentos y Cargos

Las relaciones de tipo *1* a *0..1* se implementaron mediante tablas separadas para evitar la presencia de valores nulos innecesarios y mantener las entidades con

una única responsabilidad para mejorar la mantenibilidad del código. Por ejemplo, dado que no todos los docentes tienen una compensación de dedicación en todas las actividades que realizan, combinar las tablas **actividades_docentes** y dedicaciones en una única tabla habría introducido múltiples columnas nulas.

Tipos de datos: Uso de `ENUM`

En el esquema relacional de PostgreSQL se hizo uso de los tipos `ENUM` para promover la consistencia y la integridad de este tipo de datos y permitir generar reportes agrupando por valor de forma confiable. Se presenta un ejemplo en la Figura 3.8, donde se representa a la tabla **actividades_docentes**, cuya columna **rol** solo permite los valores "***responsable***", "***subresponsable***" o "***integrante***". La base de datos no permitiría accidentalmente introducir el valor "***sub-responsable***" o cualquier otra variante que pudiese ser interpretada como un valor diferente cuando la intención semántica no varía.

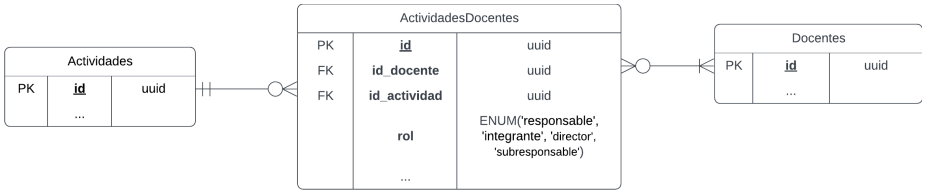


Figura 3.8: Uso de enumerados en la tabla actividades_docentes

Tipos de datos: Uso de `CITEXT`

En el esquema relacional de PostgreSQL se empleó el tipo de dato `CITEXT` en columnas de texto con restricción de unicidad. De esta forma las comparaciones en las consultas SQL son insensibles a variaciones en capitalización y no requieren al desarrollador realizar transformaciones explícitas como `LOWER()` o `UPPER()`. Al poder evitar estas transformaciones es posible optimizar las consultas a través del uso de índices simples. Un ejemplo es la columna email en la tabla usuarios, donde aunque la aplicación ya transforma los correos a minúsculas, se usa `CITEXT` para asegurar la integridad a nivel de base de datos, evitando duplicados por diferencias de capitalización. Otro ejemplo donde se aplica este enfoque es en el nombre de las entidades.

Class Table Inheritance (CTI)

Como se puede apreciar en la Figura 3.9, el diseño de la tabla actividades junto con las tablas específicas para los distintos subtipos de actividades **act_extension**, **act_gestion**, **act_enseñanza**, **act_investigacion**) siguen el patrón Class Table Inheritance (CTI). En este patrón, la tabla **actividades** actúa como una tabla general que agrupa en un único lugar los atributos comunes a todas las actividades, mientras que las tablas específicas almacenan los atributos adicionales que corresponden a cada tipo de actividad. La columna **actividades.tipo**, de tipo `ENUM`, indica a qué tabla específica se debe hacer referencia para obtener los detalles adicionales. Por ejemplo, si el tipo de actividad es "***extension***", los datos específicos estarán en la tabla **act_extension**.

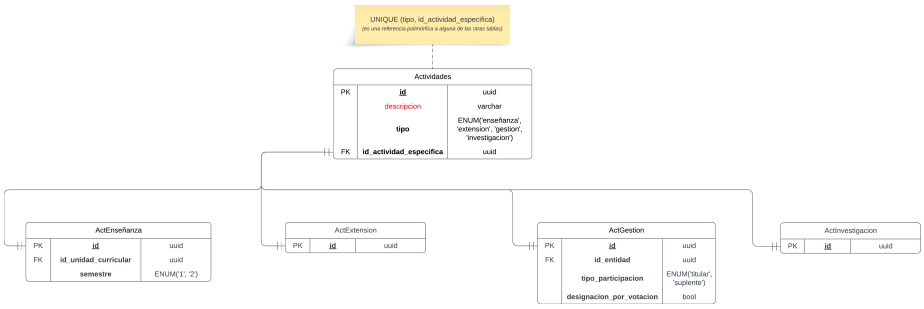


Figura 3.9: Modelo Relacional de actividades y sus subtipos

Alternativamente, se podría haber optado por el patrón Single Table Inheritance (STI), donde se utilizaría una única tabla actividades que incluyera todos los campos, dejando columnas opcionales para los distintos tipos de actividades. Sin embargo, este diseño habría generado una gran cantidad de valores nulos para las actividades que no utilizan ciertos campos y hubiese requerido una gran cantidad de validaciones extras para asegurar la integridad de otros aspectos.

Cabe mencionar que la utilización del patrón CTI conlleva una ligera complejidad adicional en las consultas, ya que es necesario realizar `JOIN` con las tablas específicas para obtener los detalles completos de cada actividad.

El patrón CTI también se aplica en el modelado de instituciones, con **instituciones** como tabla general y las específicas (**universidades**, **facultades**, **institutos**, etc.) almacenando los detalles adicionales según el tipo.

Normalización de la Base de Datos

El diseño de la base de datos se diseñó siguiendo los principios de normalización para garantizar la correcta organización de los datos, evitando redundancias y anomalías de actualización. La base se encuentra en la Forma Normal de Boyce-Codd (BCNF). Cabe mencionar que se excluyen tablas como **api_tokens**, **file_imports** y **file_import_errors**, ya que estas cumplen una función de apoyo a funcionalidades como la autenticación y la carga de archivos CSV, y han sido des normalizadas. La estrategia utilizada para realizar el pasaje del modelo conceptual al modelo relacional (Elmasri y Navathe, 2016), asegurada que el esquema resultando cumple con la Tercera Forma Normal (3NF).

Primera Forma Normal (1NF)

Para cumplir con la Primera Forma Normal (1NF), cada tabla debe tener una clave primaria y los valores de cada columna deben ser atómicos, es decir, no deben existir grupos repetidos o multivaluados.

Un ejemplo es la tabla **carreras_facultades**, que relaciona las tablas **carreras** y **facultades**. Cada registro en esta tabla contiene valores únicos y atómicos para los campos **id_carrera** y **id_facultad**, que forman una clave compuesta, asegurando que no existen valores repetidos o multivaluados. De esta manera, la estructura de la tabla garantiza que cada relación entre una carrera y una facultad se represente de manera indivisible.

En este esquema, todas las tablas cumplen con este principio por lo que la base se encuentra en 1NF.

Segunda Forma Normal (2NF)

La Segunda Forma Normal (2NF) exige que una base de datos se encuentre en 1NF y adicionalmente que todos los atributos no clave dependan completamente de la clave primaria, evitando dependencias parciales.

En las tablas con claves simples, como usuarios, los atributos no clave, como el correo electrónico y el nombre, dependen exclusivamente de la clave primaria. En las tablas con claves compuestas, como **carreras_facultades**, los atributos no clave, como **es_obligatoria**, dependen de la clave compuesta completa (**id_carrera**, **id_facultad**), lo que asegura que no hay dependencias parciales y que la tabla cumple con 2NF.

Este es el caso para todas las tablas por lo que la base se encuentra en 2NF.

Tercera Forma Normal (3NF)

La 3NF exige que todos los atributos no clave dependan exclusivamente de la clave primaria, sin dependencias transitivas entre atributos no clave.

Por ejemplo, en la tabla dedicaciones, el atributo **porcentaje_especial** depende directamente de la clave primaria, y no de otros atributos no clave como tipo. Esto asegura que cada atributo no clave se relacione únicamente con la clave primaria.

En este esquema, las tablas cumplen con este requisito por lo que la base se encuentra en 3NF.

Forma Normal de Boyce-Codd (BCNF)

En la BCNF, la base debe encontrarse en 3NF y a su vez cada dependencia funcional debe tener como determinante una súper-clave. En este esquema, las dependencias funcionales se gestionan de manera que los atributos no clave dependan únicamente de claves primarias o súper-claves.

En la tabla **actividades**, la columna **tipo** determina qué tabla especializada (**act_extension**, **act_gestion**, etc.) contiene los detalles adicionales de la actividad. A pesar de esta dependencia funcional, tipo no es una súper-clave, pero no afecta la integridad, ya que todas las dependencias se gestionan en función de la clave primaria de la tabla **actividades**. El mismo patrón de diseño se aplica en tablas como **dedicaciones** y **docentes_departamentos_cargos**, donde los atributos no clave dependen completamente de las claves primarias y no existen dependencias que violen BCNF.

Dado que este es el caso para todas las dependencias, la base se encuentra en BCNF.

Restricciones de integridad

Garantizar la integridad de los datos fue una de las prioridades en el diseño del esquema relacional, dado que este problema se destacó como uno de los más críticos en la situación previa a la implementación del sistema. Para abordar esta necesidad, las validaciones más críticas se implementan directamente a nivel de la base de datos. Esto asegura que se cumplan incluso si la lógica de validación de la aplicación contiene errores, o si los datos son manipulados fuera de la aplicación a través de consultas manuales o de sistemas externos.

En esta sección se presentan ejemplos de las restricciones de integridad implementadas a nivel de base de datos.

Restricciones de unicidad

Se aplicaron restricciones `UNIQUE` para asegurar que ciertos valores no se repitan dentro de una columna o combinación de columnas, evitando la duplicación de datos.

Ejemplo: En la tabla usuarios, la columna email es única:

```
CREATE TABLE "usuarios" (
  ...
  "email" citext UNIQUE NOT NULL
);
```

Ejemplo compuesto: En la tabla **docentes_departamentos_cargos**, se define una restricción única sobre la combinación de **id_docente**, **id_departamento** e **id_cargo**, asegurando que un docente no tenga duplicados del mismo cargo en el mismo departamento:

```
CREATE TABLE "docentes_departamentos_cargos" (
  ...
  CONSTRAINT index_unq_docente_departamento_cargo
    UNIQUE (id_docente, id_departamento, id_cargo)
);
```

Restricciones `FOREIGN KEY`

Las claves foráneas (`FOREIGN KEY`) garantizan la integridad referencial al asegurar que los valores en una columna estén presentes en otra tabla.

Ejemplo: La tabla **dedicaciones** tiene una referencia hacia **actividades_docentes** en la columna **id_actividad_docente**:

```
CREATE TABLE "dedicaciones" (
  ...
  CONSTRAINT "FK_dedicaciones.id_actividad_docente"
    FOREIGN KEY ("id_actividad_docente")
      REFERENCES "actividades_docentes"("id")
);
```

Restricciones `NOT NULL`

Se utilizó la restricción `NOT NULL` en columnas donde la presencia de un valor es obligatoria para el correcto funcionamiento del sistema.

Ejemplo: En la tabla **docentes**, columnas como **cedula**, **nombres** y **apellidos** no pueden ser nulas:

```
CREATE TABLE "docentes" (
  ...
  "cedula" varchar UNIQUE NOT NULL,
  "nombres" varchar NOT NULL,
  "apellidos" varchar NOT NULL
);
```

Restricciones `CHECK`

Las restricciones `CHECK` permiten validar condiciones específicas dentro de una tabla para garantizar la consistencia lógica de los datos.

Ejemplo `NOT NULL` condicional: En la tabla **dedicaciones**, se valida que la columna **porcentaje_especial** solo tenga un valor si el tipo de dedicación es "***especial***":

```
CREATE TABLE "dedicaciones" (
  ...
  CONSTRAINT porcentaje_especial_solo_si_tipo_especial
    CHECK (((tipo = 'especial') AND (porcentaje_especial IS NOT NULL))
        OR ((tipo <> 'especial') AND (porcentaje_especial IS NULL)))
);
```

Ejemplo de fechas: En la tabla **docentes_departamentos_cargos**, se asegura que la fecha de inicio sea anterior o igual a la fecha de finalización:

```
CREATE TABLE "docentes_departamentos_cargos" (
  ...
  CONSTRAINT fecha_inicio_before_fecha_fin CHECK (fecha_inicio <= fecha_fin)
);
```

Exclusiones mediante `EXCLUDE`

Las restricciones de exclusión (`EXCLUDE`) se aplican para evitar solapamientos en rangos de valores.

Ejemplo: En la tabla **actividades_docentes**, se evita que un docente esté asignado a la misma actividad en un período que se solape con otro, ya que en este caso los registros deberían haber sido previamente combinados en un única fila:

```
CREATE TABLE "actividades_docentes" (
  ...
  EXCLUDE USING gist (
    id_docente WITH =,
    id_actividad WITH =,
    daterange(fecha_inicio, coalesce(fecha_fin, 'infinity'), '[]') WITH &&
  )
);
```

Restricciones de tipo `ENUM`

Se utilizaron tipos `ENUM` para restringir los valores posibles en columnas específicas.

Ejemplo: En la tabla **actividades**, la columna tipo solo admite valores definidos en **tipo_actividad**:

```
CREATE TYPE tipo_actividad AS ENUM(
    'enseñanza',
    'extension',
    'gestion',
    'investigacion'
);

CREATE TABLE "actividades" (
  ...
  "tipo" tipo_actividad NOT NULL
);
```

Utilización de índices

Si bien en la actualidad la base de datos no cuenta con un volumen significativo de datos, se espera que con el paso de los años la cantidad de registros aumente. Por esta razón, resulta fundamental optimizar el rendimiento del sistema desde el diseño inicial, ya sea para el funcionamiento regular de la aplicación como para futuras consultas realizadas a través de herramientas de visualización de datos.

En una base de datos, un índice es una estructura que permite acelerar las consultas al proporcionar un acceso más eficiente a los registros, evitando la necesidad de realizar búsquedas secuenciales en toda la tabla. Sin embargo, la creación de índices debe evaluarse y justificarse en cada caso, ya que añaden un costo adicional en operaciones de escritura, como las inserciones y actualizaciones, así como en el espacio en disco necesario. Por este motivo, se evaluó cuidadosamente qué índices agregar, priorizando aquellos que realmente contribuyen a la optimización de consultas frecuentes y operaciones críticas.

Todos los índices utilizados son del tipo `B-tree`, el por defecto de PostgreSQL. Los índices `B-tree` organizan los datos de manera jerárquica, siendo eficientes para consultas de igualdad (`=`), comparación (`<`, `>`) y ordenamiento (`ORDER BY`).

Índices en claves foráneas

Se implementaron índices en todas las columnas que funcionan como claves foráneas para optimizar las búsquedas y las operaciones de unión (`JOIN`) entre tablas. Es por eso que estos índices también mejoran el rendimiento de las operaciones de inserción y actualización de claves foráneas, ya que suelen permitir que la validación de presencia en la tabla relacionada se realice más rápidamente.

En PostgreSQL, la creación de un índice sobre una clave foránea no es automática, por lo que debe realizarse explícitamente.

Ejemplo: En la tabla **docentes_departamentos_cargos**, se indexó la columna **id_docente**:

```
CREATE INDEX IF NOT EXISTS index_docentes_departamentos_cargos_on_id_docente
ON "docentes_departamentos_cargos" (id_docente);
```

Índices generados por restricciones `UNIQUE`

En PostgreSQL, al definir una columna con la restricción `UNIQUE`, se crea automáticamente un índice único que garantiza la integridad de los datos al evitar valores duplicados. Además, este índice optimiza las búsquedas sobre la columna, evitando la necesidad de crear un índice adicional manualmente.

Por ejemplo, en la tabla **docentes**, la columna **cedula** tiene una restricción de unicidad:

```
CREATE TABLE docentes (
  cedula varchar UNIQUE NOT NULL
);
```

En este caso, PostgreSQL genera internamente un índice único en **cedula**.

```
CREATE UNIQUE INDEX docentes_cedula_key
ON docentes(cedula text_ops);
```

Índices en columnas de uso frecuente

Se definieron índices en las columnas que se utilizan con frecuencia en filtros o condiciones `WHERE` para optimizar el rendimiento de las consultas.

Es importante destacar que esta estrategia se aplicó únicamente en tablas con un volumen considerable de filas. En tablas con pocos registros, como **institutos**, la base de datos suele preferir un SEQUENTIAL SCAN en lugar de un *INDEX SCAN*, lo que hace irrelevante la creación de índices en esos casos.

Por ejemplo, en la tabla **docentes**, se indexó la columna **created_at**:

```
CREATE INDEX IF NOT EXISTS idx_docentes_created_at
ON docentes (created_at);
```

Índices generados por restricciones `EXCLUDE`

En PostgreSQL, las restricciones `EXCLUDE` generan automáticamente un índice, generalmente `GIST`, para realizar las validaciones de manera eficiente. No es necesario crear un índice adicional, ya que PostgreSQL lo maneja implícitamente.

Por ejemplo, en la tabla **actividades_docentes**, se utiliza una restricción `EXCLUDE` para evitar que un docente esté asignado a la misma actividad en períodos que se solapen:

```
CREATE TABLE actividades_docentes (
    ...
    CONSTRAINT actividades_docentes_exclude_periodo
        EXCLUDE USING gist (
            id_docente WITH =,
            id_actividad WITH =,
            daterange(fecha_inicio, COALESCE(fecha_fin, 'infinity'), '[]')
     WITH &&
        )
);
```

Internamente, PostgreSQL define un índice `GIST` para soportar esta restricción eficientemente, como se muestra a continuación:

```
CREATE INDEX actividades_docentes_id_docente_id_actividad_daterange_excl
    ON actividades_docentes
    USING GIST (
        id_docente,
        id_actividad,
        daterange(fecha_inicio, COALESCE(fecha_fin, 'infinity'), '[]')
    );
```

3.2. Arquitectura

Esta sección describe la arquitectura de alto nivel y las decisiones tecnológicas fundamentales, incluyendo la adopción del estilo REST y la selección de bases de datos. Posteriormente se detallan aspectos específicos de la implementación como el manejo de sesiones, procesamiento de archivos CSV y despliegue. Finalmente se abordan los aspectos de calidad del software, tales como pruebas automáticas, integración continua y manejo de errores.

3.2.1. Arquitectura a alto nivel

En la Figura 3.10 se presenta un diagrama de alto nivel de los principales elementos de la arquitectura del sistema construido.

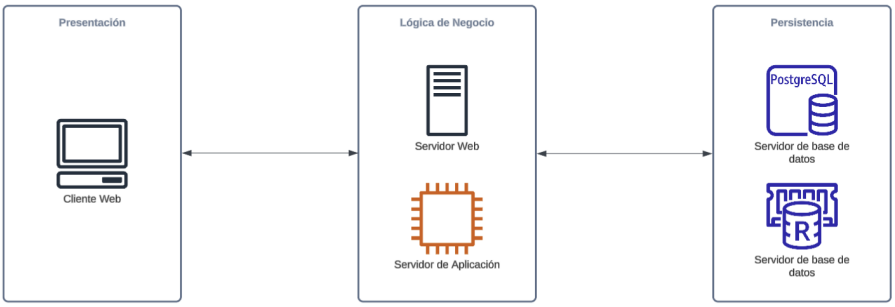


Figura 3.10: Diagrama de la arquitectura por capas

- **Cliente web**: Es la aplicación web a la que acceden los docentes y administradores de Plataforma de Registro y Organización del Inco (PROInco) desde sus navegadores.
- **Servidor de aplicaciones**: Es la aplicación que maneja la lógica de negocio, procesa las solicitudes provenientes del cliente web, interactuando con las bases de datos y otros servicios externos.
- **Base de Datos Principal**: Es donde se almacenan los datos estructurados que son consumidos por el servidor y las herramientas de análisis de datos.
- **Base de Datos en memoria**: Es utilizada por el servidor para encolar tareas que realiza de forma asíncrona, permitiendo una ejecución eficiente y no bloqueante de las solicitudes de la aplicación web.

3.2.2. Decisiones principales

Para el diseño de la arquitectura se consideraron tanto los requerimientos funcionales como los no funcionales previamente identificados. Principalmente, la mayoría de las decisiones se vieron limitadas por la necesidad de que el sistema no incurra en costos, lo que implica el uso exclusivo de tecnologías gratuitas o desarrollos propios. Además, se priorizó que el sistema sea fácilmente comprensible y mantenible por próximos estudiantes, y que su arquitectura permita una integración sencilla con otros sistemas en el futuro.

Además se aplicaron los principios "*12 Factor App*", con el fin de promover la robustez, escalabilidad, mantenibilidad y desplegabilidad del sistema. En el Anexo C se presentan cada uno de los principios junto con una breve explicación de su aplicación concreta en el proyecto.

En las siguientes secciones se presentan los aspectos de alto nivel del diseño del sistema, acompañados de sus justificaciones y las alternativas evaluadas.

Separación del *backend* y *frontend*

La plataforma está dividida en *backend* y *frontend*, como se puede apreciar la Figura 3.11. Se denomina *frontend* a la aplicación web mientras que el término *backend* abarca al servidor y las bases de datos, que se comunican mediante Hypertext Transfer Protocol Secure (HTTPS).

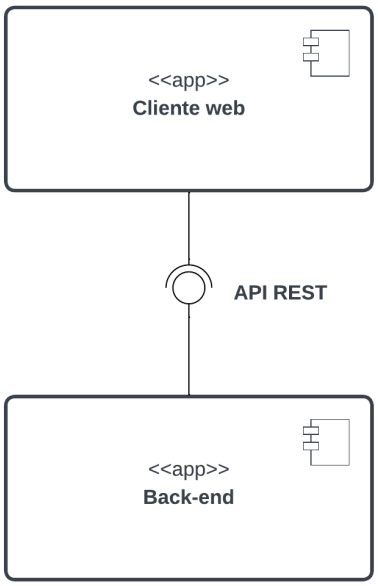


Figura 3.11: Diagrama de componentes y conectores

La decisión de separar al cliente web del *backend* se fundamenta a través de las siguientes consideraciones:

1. **Escalabilidad**: Al desacoplar la aplicación web del *backend*, cada componente puede escalar de manera independiente según las necesidades de carga. Esto permite, por ejemplo, aumentar los recursos del servidor *backend* para manejar más solicitudes de procesamiento sin la obligación de

invertir en escalar innecesariamente la infraestructura del *frontend* al mismo tiempo.

2. **Flexibilidad y Mantenibilidad**: La separación en módulos bien definidos facilita el mantenimiento y la evolución del sistema, ya que cada componente puede ser actualizado, refactorizado o reemplazado de manera independiente.
3. **Desarrollo Independiente**: La arquitectura desacoplada permite que los desarrolladores de *frontend* y *backend* puedan trabajar de manera independiente y en paralelo sin conflictos.
4. **Interoperabilidad**: La implementación de una API Representational State Transfer (REST) estandarizada permite que la lógica de negocio del *backend* sea accesible no solo por la aplicación web, sino también por otros futuros tipos de clientes, como una aplicación móvil.
5. **Reutilización de Servicios**: La API REST desarrollada en el *backend* puede ser reutilizada por diferentes interfaces de usuario o por servicios de terceros que beneficien de integrarse, maximizando el retorno de inversión del desarrollo del sistema.

API REST como estilo arquitectónico

REST es un estilo arquitectónico que se utiliza para diseñar redes de aplicaciones y servicios, basado en el protocolo Hypertext Transfer Protocol (HTTP). En una API REST, los recursos del sistema se exponen a través de URLs y se manipulan mediante operaciones estándar como `GET`, `POST`, `PUT` y `DELETE`, que corresponden a las operaciones conocidas como *CRUD* (*Create*, *Read*, *Update*, *Delete*). Cada operación es *stateless*, lo que significa que no se mantiene estado en el servidor entre solicitudes, permitiendo que cada petición sea independiente y contenga toda la información necesaria para su procesamiento.

Se optó por implementar una API REST en este proyecto por varias razones. En primer lugar, REST ofrece una estructura convencional, clara y predecible, lo que acelera el desarrollo y facilita el mantenimiento. Su simplicidad y estandarización la hacen ampliamente compatible con diversos clientes y tecnologías modernas, ampliando las posibilidades de integración y desarrollo.

Comunicación por HTTPS

HTTPS es una extensión segura del protocolo HTTP, que utiliza cifrado mediante Secure Sockets Layer (SSL)/Transport Layer Security (TLS) para proteger la integridad y la confidencialidad de los datos transmitidos entre el cliente y el servidor. Se implementó HTTPS en este proyecto para proteger la confidencialidad e integridad de los datos transmitidos, previniendo que la información intercambiada sea interceptada y leída o alterada por terceros.

Capítulo 4

Implementación

En este capítulo se documentan los aspectos relacionados a la implementación del sistema construido, y flujo de la información durante funcionalidades claves junto con las justificaciones para las decisiones tomadas y alternativas consideradas.

4.1. Funcionalidades implementadas

A continuación, se describen las principales funcionalidades del sistema implementado.

En primer lugar, los usuarios administradores acceden a la plataforma utilizando su correo electrónico y contraseña, como se muestra en la Figura 4.1.

Figura 4.1: Formulario de inicio de sesión

La pantalla principal es el *Dashboard*, donde se visualizan gráficos que analizan y resumen los datos de la base de datos. La primera gráfica muestra la distribución de grados a través de los años, como se aprecia en la Figura 4.2.

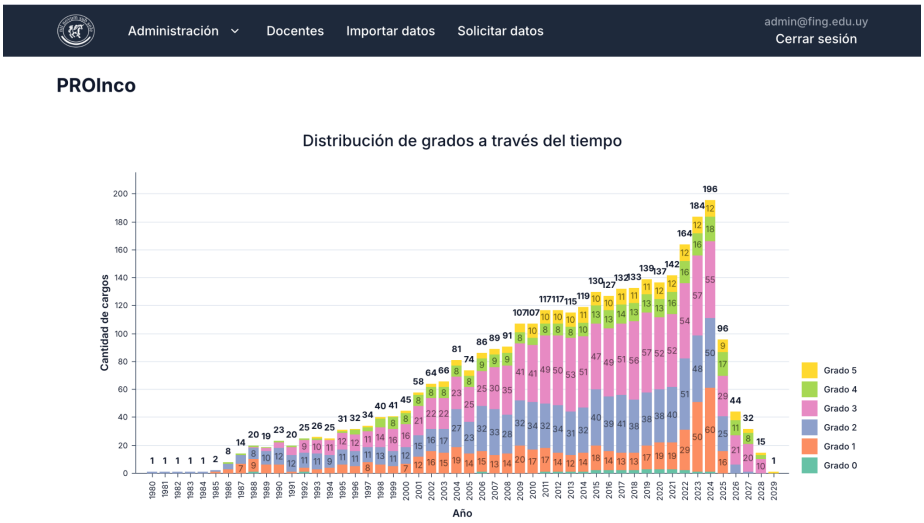


Figura 4.2: *Dashboard* de la plataforma (perspectiva del administrador)

La segunda gráfica muestra las horas dedicadas a actividades de gestión y enseñanza por DT (con y sin), para el año seleccionado. Esta visualización se presenta en la Figura 4.3.

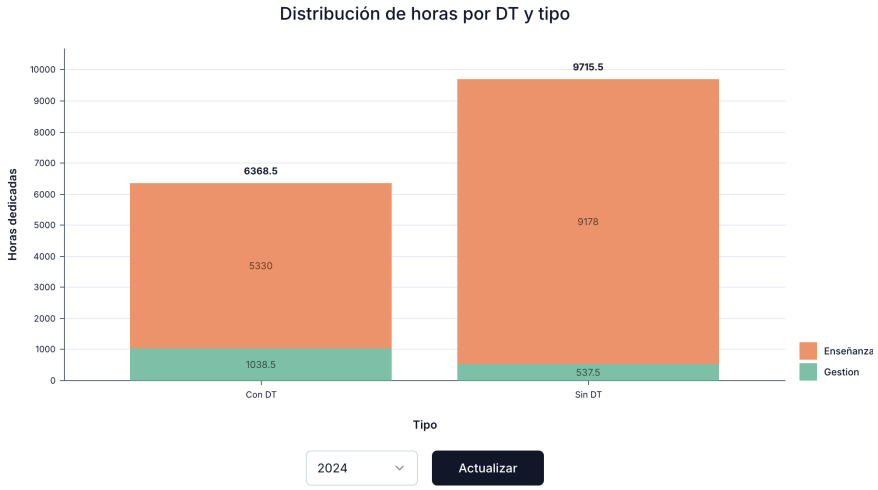


Figura 4.3: Gráfica de distribución de horas por DT y tipo

La tercera gráfica ilustra las horas dedicadas a actividades de gestión y enseñanza por grado, para el año seleccionado, según se observa en la Figura 4.4.

Figura 4.4: Gráfica de distribución de horas por grado y tipo

En el menú de Administración, el administrador puede acceder a las pestañas de "Asignaturas" (Unidades Curriculares), "Cargos" y "Usuarios", como se muestra en la Figura 4.5.

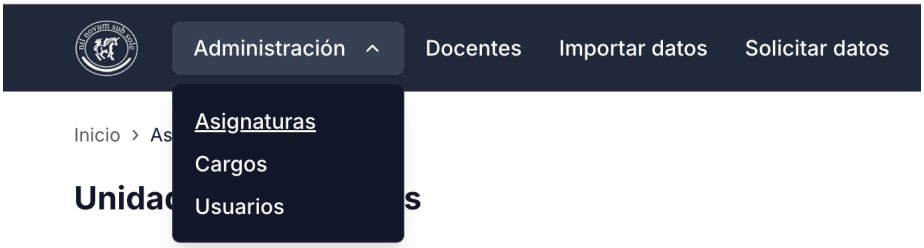


Figura 4.5: Menú de navegación

En la pestaña de asignaturas, el administrador puede ver la lista, filtrar y buscar asignaturas. El administrador con rol de escritura puede también editarlas, eliminarlas y agregar nuevas. La Figura 4.6 presenta el listado de asignaturas disponibles.

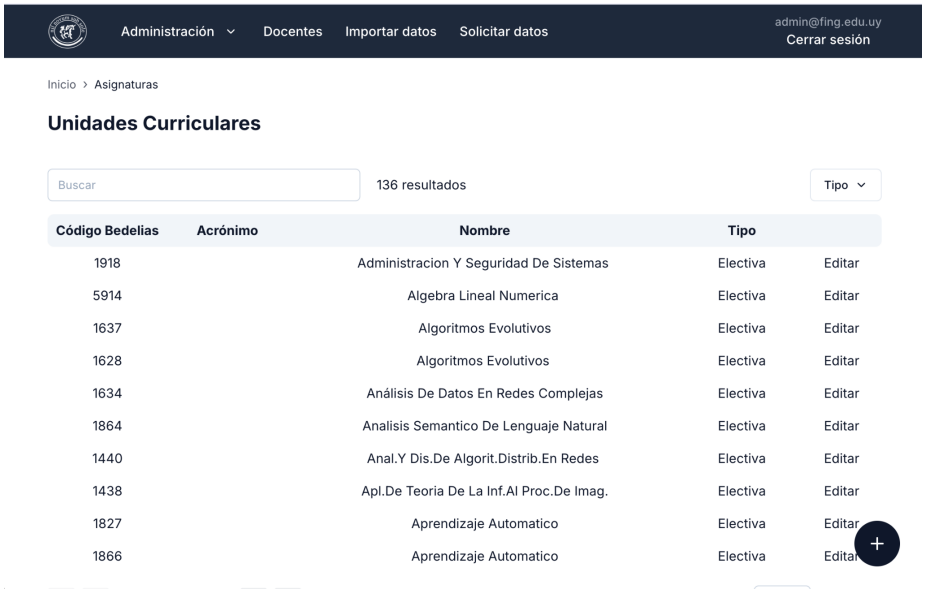


Figura 4.6: Listado de asignaturas

Para la edición de asignaturas existentes, el sistema proporciona un formulario específico que se ilustra en la Figura 4.7.

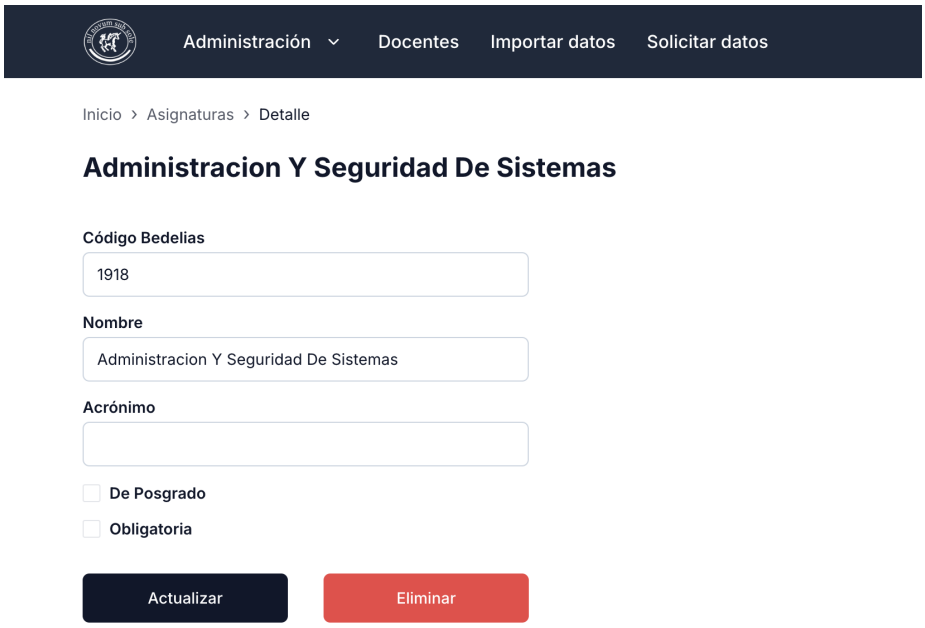


Figura 4.7: Formulario de edición de asignatura

Del mismo modo, cuando se requiere crear una nueva asignatura, el administrador puede utilizar el formulario de creación que se muestra en la Figura 4.8.

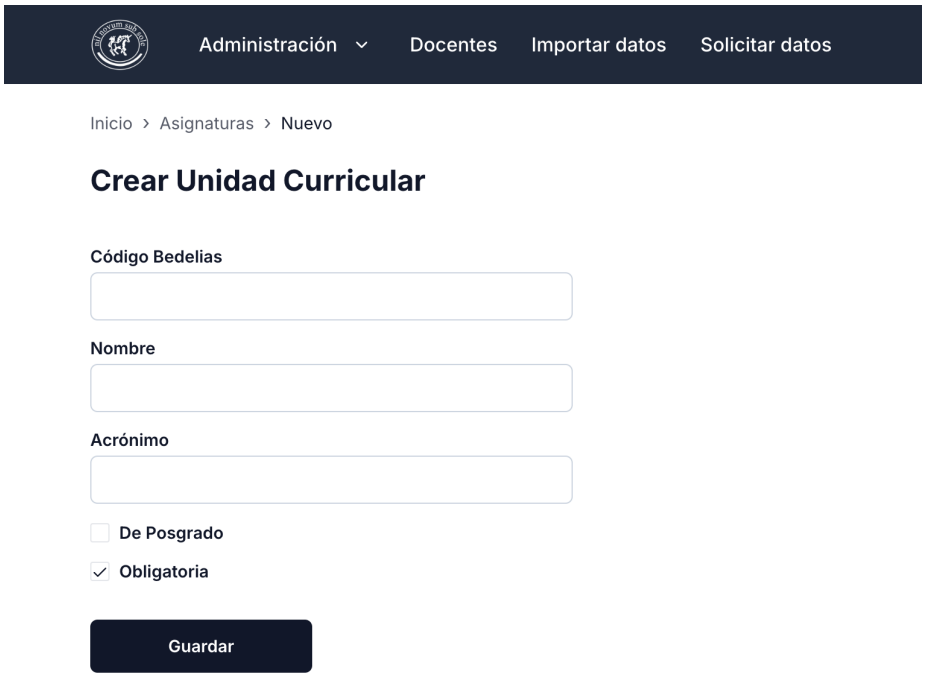


Figura 4.8: Formulario de creación de asignatura

En la pestaña “Cargos” se presenta la lista de cargos existentes en el sistema, los cuales se identifican por la combinación de tipo y grado. Estos cargos se generan automáticamente al importar datos desde las planillas, pero también es posible crear cargos nuevos desde el listado de cargos de un docente, como se mostrará más adelante. En esta pestaña se brinda al administrador con rol de escritura la opción de eliminar cargos, facilitando así la corrección de posibles errores provenientes de las planillas. La Figura 4.9 muestra la interfaz correspondiente.

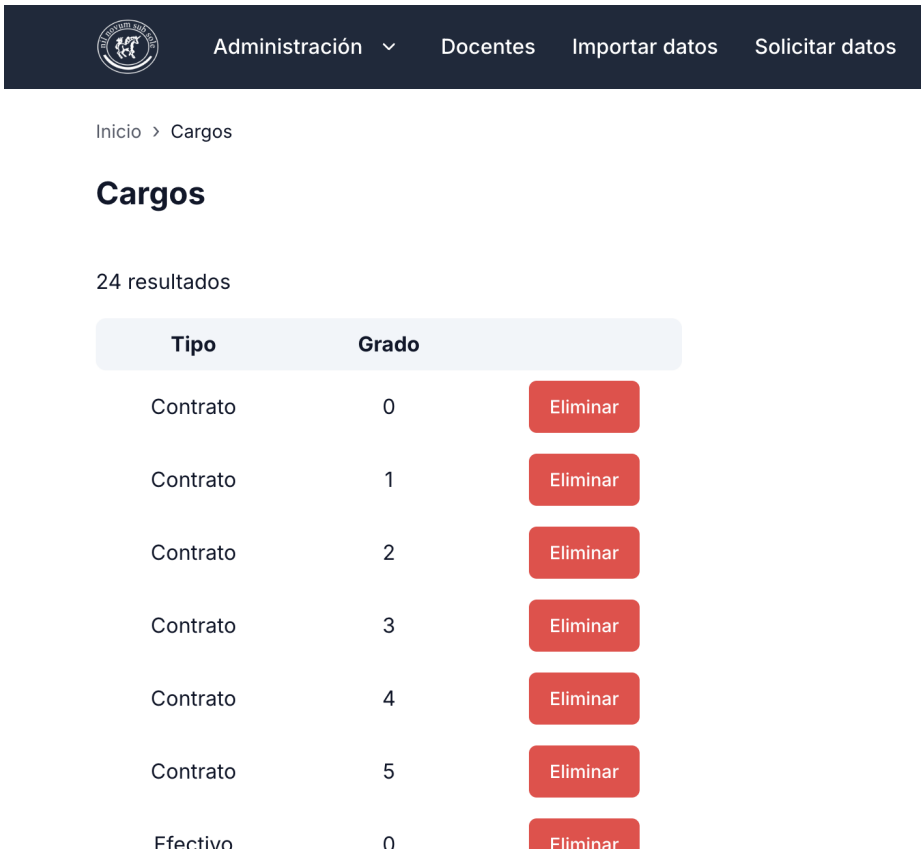


Figura 4.9: Listado de cargos

El administrador con rol de admin tiene la posibilidad de determinar quiénes tienen acceso a este sistema, a través de la pestaña “Usuarios”, como se presenta en la Figura 4.10.

Figura 4.10: Listado de usuarios

La pestaña más importante es la de “Docentes”, donde se visualiza la lista de docentes permitiendo la búsqueda, filtrados avanzados, ordenamiento por múltiples columnas, en todos los casos pudiendo exportar los resultados de la búsqueda en formato CSV. Esta funcionalidad se aprecia en la Figura 4.11.

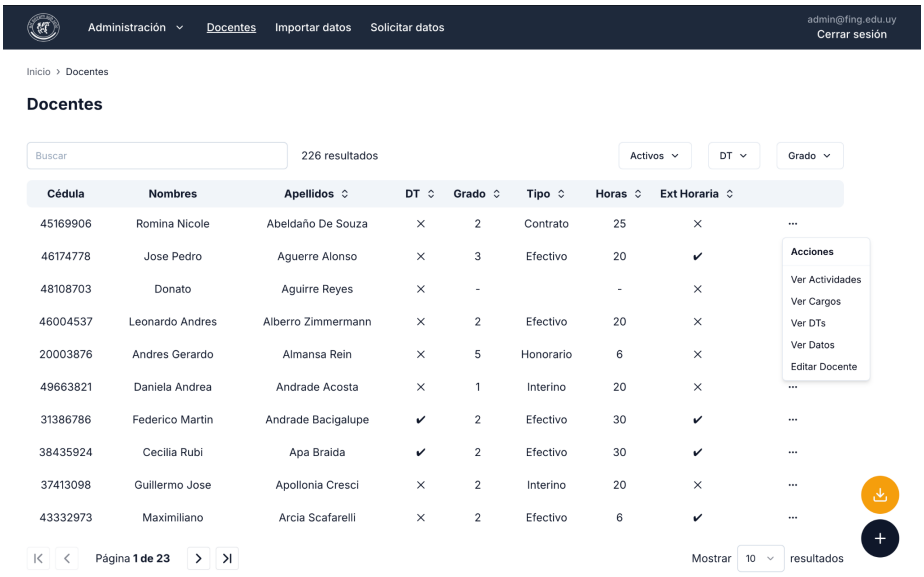


Figura 4.11: Listado de docentes

Si bien los docentes suelen ser importados utilizando la herramienta de importación, el sistema igualmente permite crearlos manualmente mediante el formu-

lario que se ilustra en la Figura 4.12.

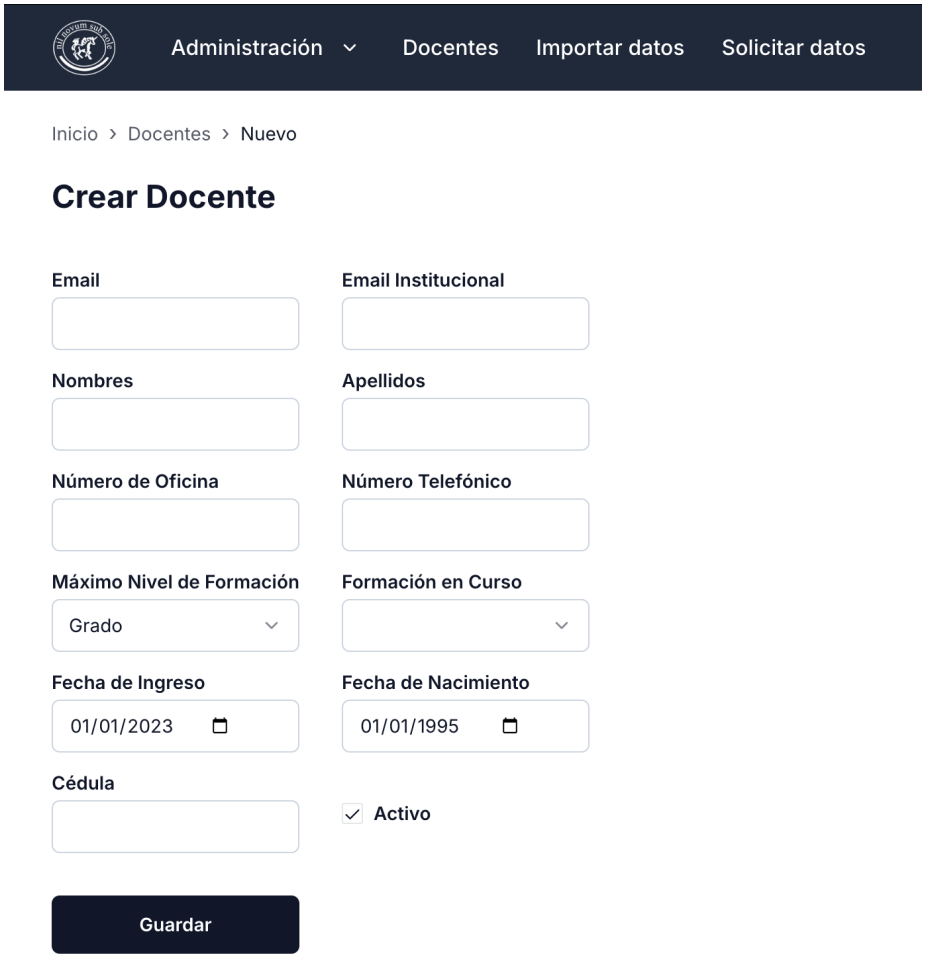


Figura 4.12: Formulario de creación de docente

El sistema también permite a los administradores con rol de escritura actualizar los datos de un docente, principalmente para permitir corregir inconsistencias que suelen contener las planillas que son importadas. La Figura 4.13 muestra el formulario utilizado para esta función.

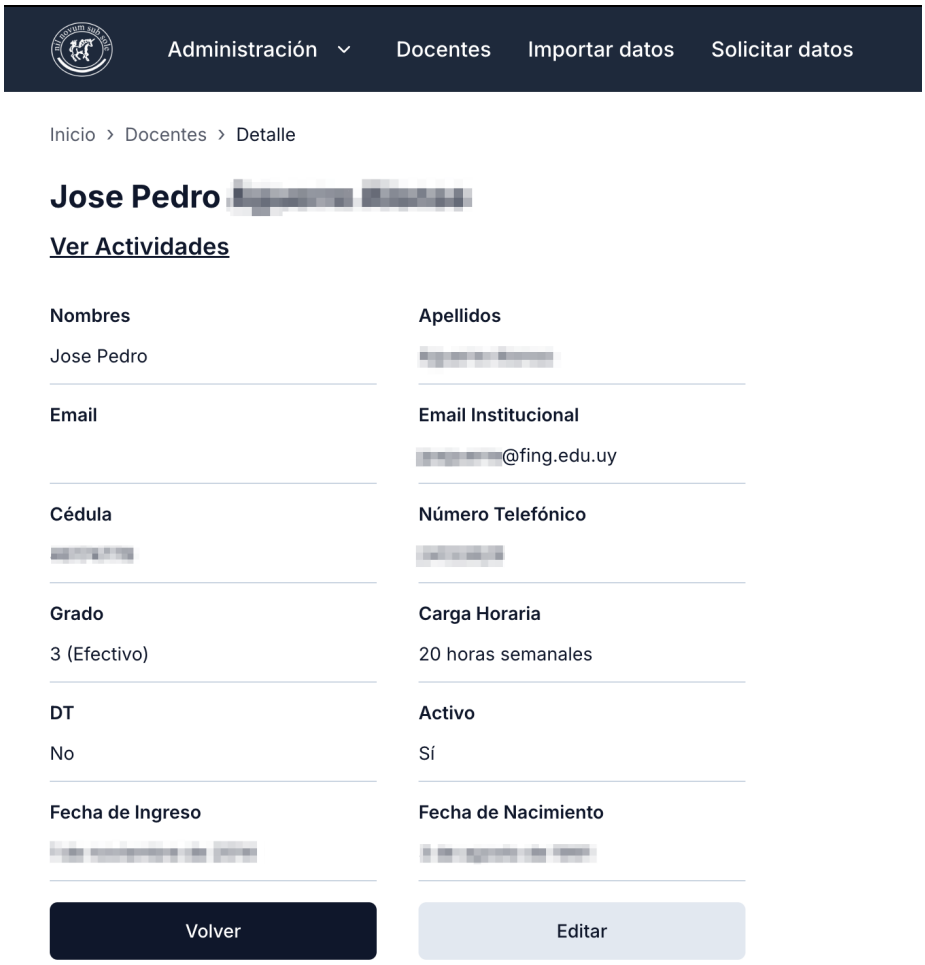


Figura 4.13: Formulario de edición de docente

El administrador puede acceder al listado de actividades de un docente con su histórico, pudiendo filtrarlas, según se muestra en la Figura 4.14.

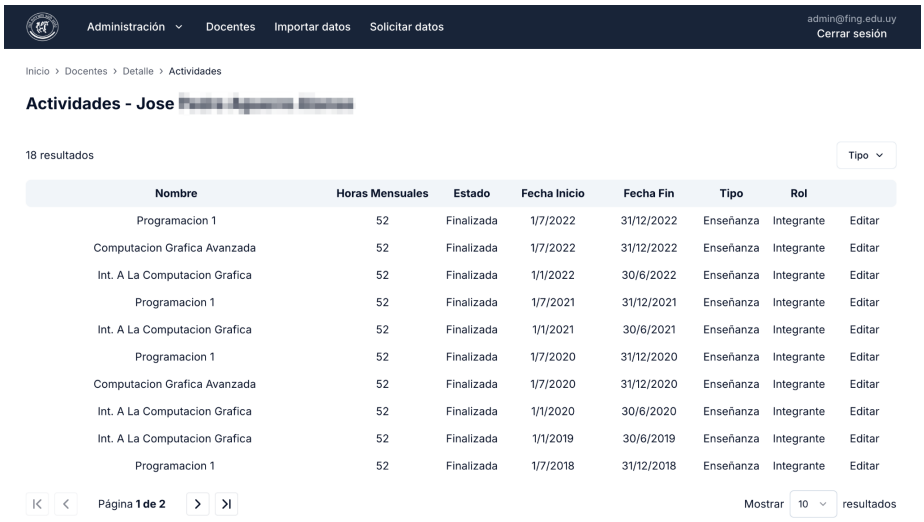


Figura 4.14: Listado de actividades del docente

Las actividades pueden ser modificadas por administradores con rol de escritura utilizando el formulario que se presenta en la Figura 4.15.

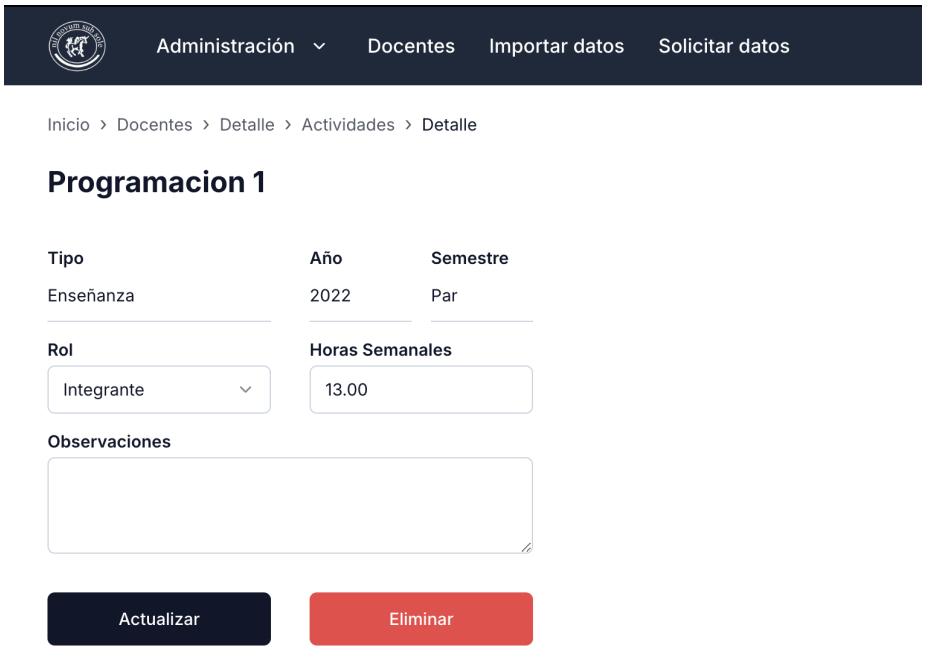


Figura 4.15: Formulario de edición de actividad docente

El administrador también puede acceder al listado de dedicaciones totales de un docente con su histórico, pudiendo filtrarlas, como se aprecia en la Figura 4.16.

Figura 4.16: Listado de DTs del docente

El administrador con rol de escritura puede actualizar las DTs de un docente para corregir errores que contengan las planillas, utilizando el formulario que se ilustra en la Figura 4.17.

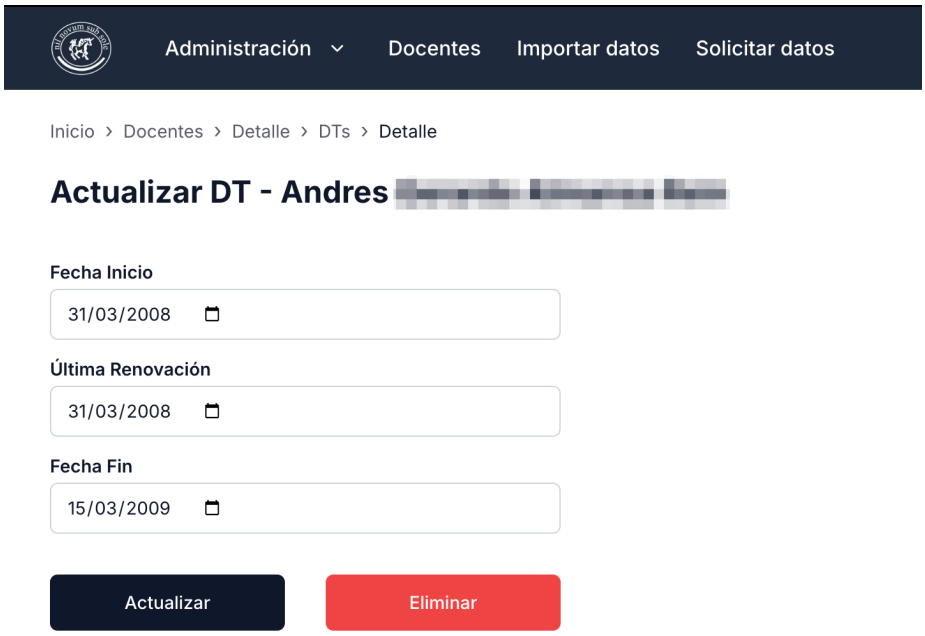


Figura 4.17: Formulario de edición de DT del docente

Para cada docente también se puede visualizar su historial de cargos, como se muestra en la Figura 4.18.

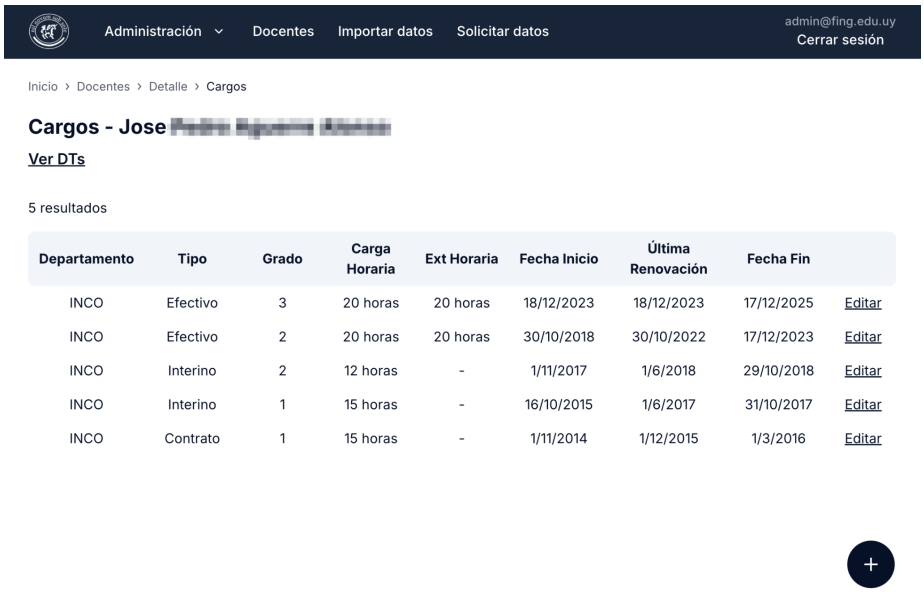


Figura 4.18: Listado de cargos del docente

El administrador con rol de escritura puede crear cargos nuevos o editar los cargos de cada docente para corregir errores que contengan las planillas, mediante el formulario que se presenta en la Figura 4.19.

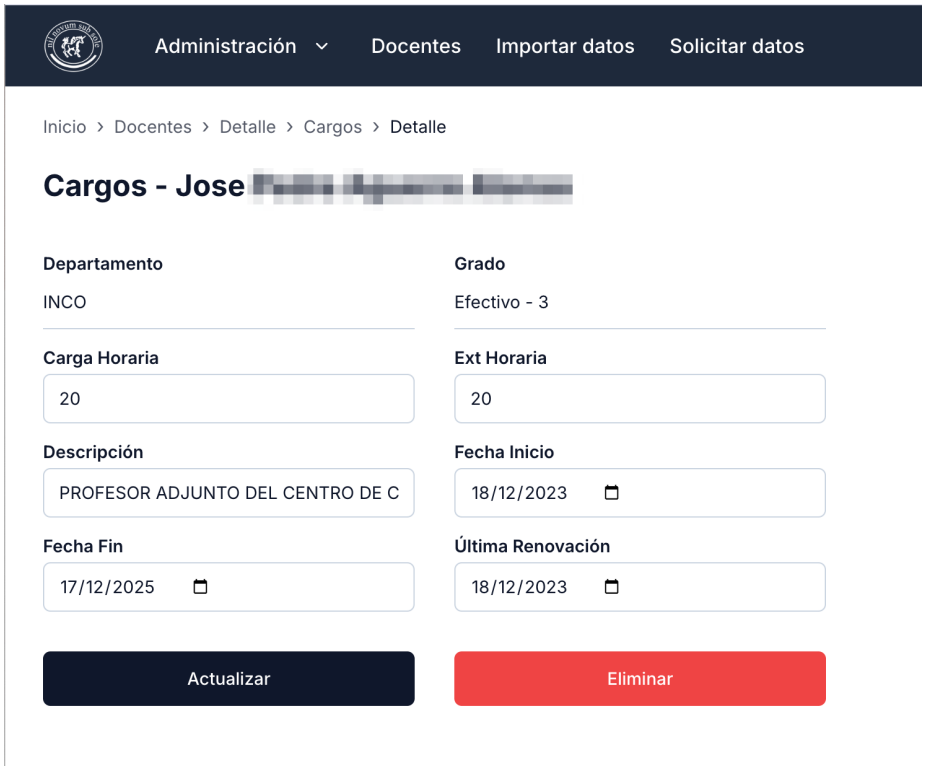


Figura 4.19: Formulario de edición de cargo del docente

Desde la pestaña “Importar Datos”, el administrador puede seleccionar uno de los cinco tipos de archivos en formato CSV disponibles para la importación: “Emails”, “Docentes”, “Unidades Curriculares”, “Asignaciones” o “Cargos & DTs”. Tras seleccionar el tipo de archivo, la interfaz muestra las columnas requeridas para orientar al administrador en la correcta preparación del archivo, como se observa en la Figura 4.20.

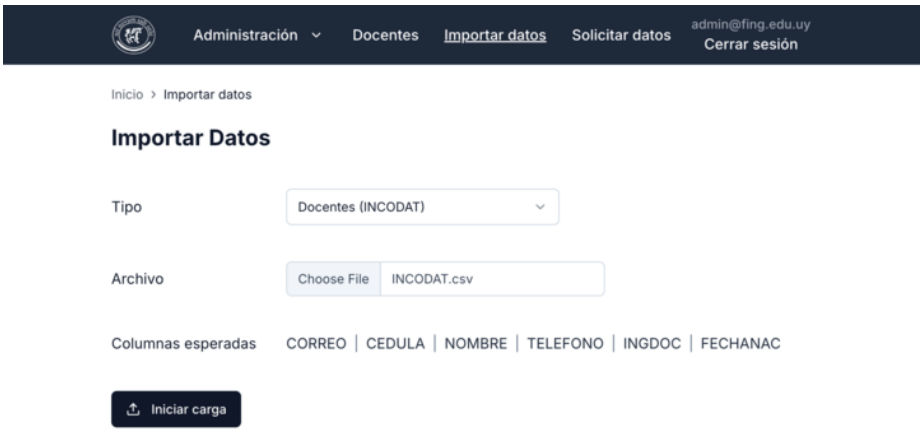


Figura 4.20: Formulario de importación de datos

Una vez que se sube el archivo, el sistema proporciona un resumen detallado del procesamiento, indicando cuántas filas fueron procesadas correctamente y cuántas presentaron errores. Para cada fila con error, se muestra su número, el contenido y los errores detectados, facilitando la corrección por parte del administrador antes de volver a importar el archivo corregido. Esta información se presenta según se ilustra en la Figura 4.21.

Una de las características más importantes de la funcionalidad de carga de archivos CSV es que permite volver a subir el archivo completo cualquier cantidad de veces sin duplicar información, garantizando así la idempotencia del proceso. El sistema es capaz de detectar qué datos son nuevos y no sobrescribe aquellos que fueron corregidos manualmente luego de una carga anterior. Esta funcionalidad se explica en más detalle en la Sección 4.8.

Figura 4.21: Resultado de la importación de datos

En la pestaña “Solicitar datos” se puede enviar vía correo electrónico a uno o todos los docentes activos una solicitud de actualización de sus actividades de gestión, que previamente no se recopilan en ningún otro sistema del Instituto por lo que no hay planilla para importarlos. La Figura 4.22 muestra el formulario correspondiente.

Figura 4.22: Formulario de solicitud de actualización de actividades de gestión

Los docentes que reciban esta solicitud recibirán un correo electrónico como el que se presenta en la Figura 4.23.

Figura 4.23: Email de solicitud de actualización de actividades de gestión

Al hacer *click* en el botón "Actualizar" que se ilustra en la Figura 4.23, obtendrán un *link* de acceso único a la plataforma, donde solamente podrán visualizar y modificar su propia información pero no la de otros docentes. La interfaz que visualizan se muestra en la Figura 4.24.

Figura 4.24: *Dashboard* de la plataforma de actualización de actividades de gestión (perspectiva del docente)

El docente puede declarar que realiza una nueva actividad, para la cual primero debe seleccionar una actividad. Para eso puede buscar y seleccionar una actividad que ya exista (previamente creada por otro docente o por un administrador), como se aprecia en la Figura 4.25.

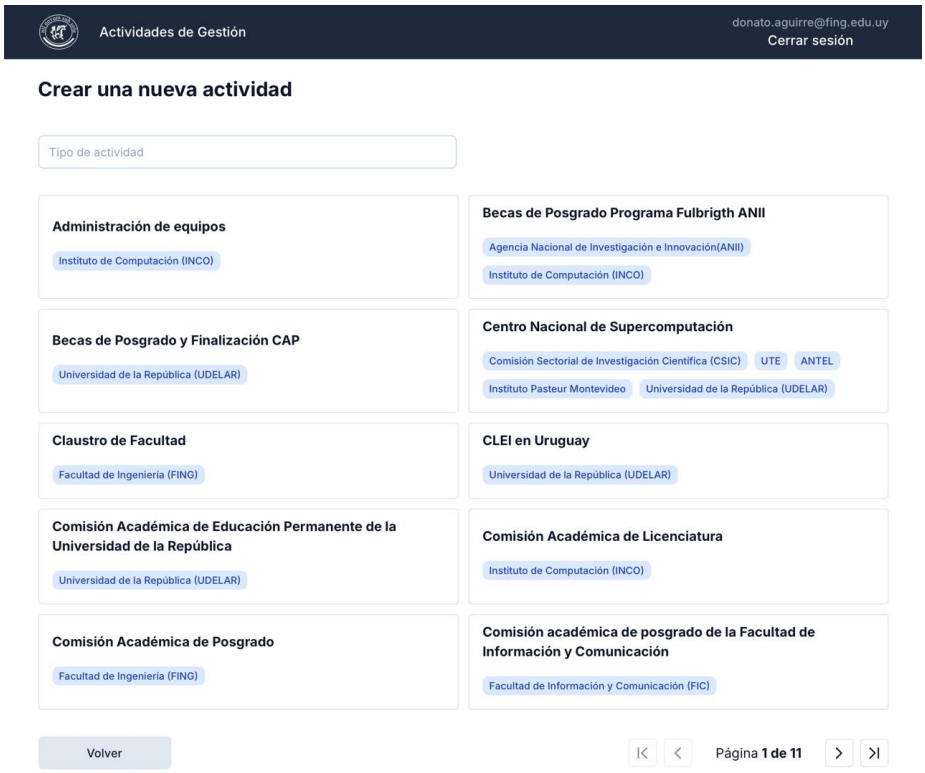


Figura 4.25: Listado de actividades de gestión (perspectiva del docente)

Si la actividad que busca no se encuentra previamente creada, el sistema le permite registrar la nueva actividad mediante la interfaz de búsqueda que se ilustra en la Figura 4.26.

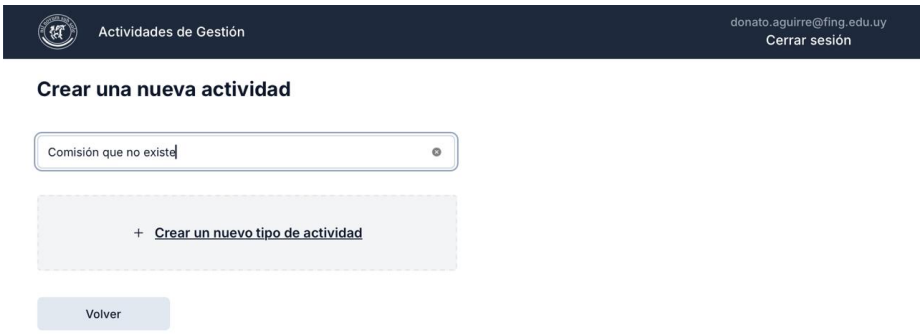


Figura 4.26: Búsqueda de actividades de gestión (perspectiva del docente)

Al crear una nueva actividad se le solicita que agregue un nombre, tipo y la institución. Para las instituciones, el docente puede seleccionar una institución previamente creada por otro docente o crear una nueva allí mismo, utilizando el formulario que se presenta en la Figura 4.27.

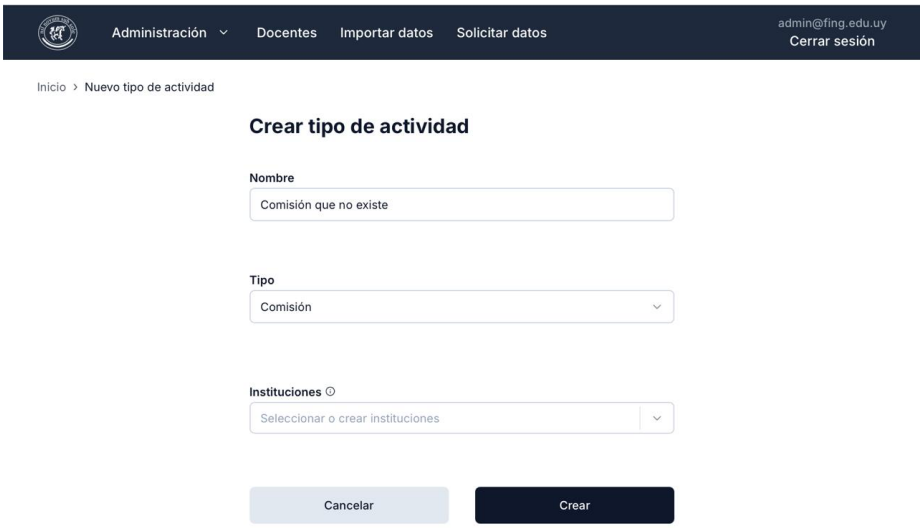


Figura 4.27: Creación de actividad de gestión (perspectiva del docente)

Una vez seleccionada la actividad, el docente debe completar datos sobre su rol en esa actividad, según se muestra en la Figura 4.28.

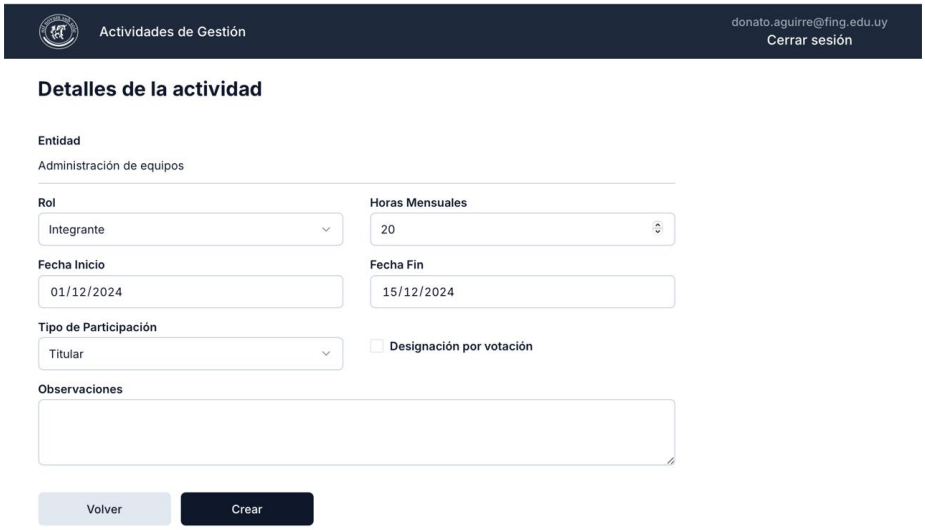


Figura 4.28: Creación de actividad docente de gestión (perspectiva del docente)

4.2. Lenguajes de programación y frameworks utilizados

En este capítulo se presentan las tecnologías seleccionadas para el desarrollo del sistema, incluyendo TypeScript con React y Next.js para el *frontend*, Node.js y Adonis.js para el *backend*, PostgreSQL como base de datos principal y Redis para trabajos asincrónicos. El despliegue se realiza mediante Docker en la plataforma MiNube (Antel, 2024b), con PM2 como gestor de procesos y Mailjet para el envío de correos electrónicos. Se analizan las decisiones tecnológicas fundamentales y su implementación específica en el contexto del proyecto.

4.2.1. JavaScript como principal lenguaje de programación

Se utilizó JavaScript (Mozilla Developer Network, 2024) para el *frontend* porque es el estándar en el desarrollo de aplicaciones web, siendo el único lenguaje de programación que se ejecuta de forma nativa en los navegadores.

Por otro lado, se eligió JavaScript también para el *backend* con el propósito de minimizar la barrera de mantenimiento para futuros desarrolladores que deban

encargarse de ambas partes del sistema como un todo. Al utilizar un mismo lenguaje en ambos entornos y proporcionar cierta coherencia entre los código base, se simplifica la curva de aprendizaje para los nuevos desarrolladores.

4.2.2. Tipado estático a través de TypeScript

JavaScript por sí solo no es un lenguaje tipado, lo que puede llevar a la no detección de errores durante la fase de desarrollo que pueden ser la causa de defectos en el sistema en producción. TypeScript (Microsoft, 2024b) es un superset de JavaScript ampliamente reconocido en la comunidad de desarrollo, que introduce el tipado estático opcional. El código TypeScript se compila a JavaScript estándar antes de ser ejecutado, asegurando total compatibilidad.

Se eligió TypeScript para este proyecto, tanto para el *backend* como para el *frontend*, principalmente porque reduce la posibilidad de errores y aumenta la velocidad y productividad de desarrollo. Gracias a su tipado estático, el Integrated Development Environment (IDE) ofrece funcionalidades como auto completado y señalización de errores, entre otros.

4.2.3. Tecnologías utilizadas en la aplicación web

Se decidió utilizar React (Meta, 2024b) con Next.js (Vercel, 2024) como *framework* para acelerar el desarrollo en el *frontend* y garantizar la mantenibilidad. React es una biblioteca de JavaScript extremadamente popular, conocida por su capacidad para construir interfaces de usuario dinámicas y eficientes mediante componentes reutilizables. Su amplia adopción y popularidad son factores clave, especialmente porque este proyecto será mantenido por futuros desarrolladores, lo que hace crucial evitar tecnologías de nicho y optar por herramientas bien conocidas y soportadas en la comunidad. Next.js, por su parte, es el *framework* basado en React más popular para *frontend*, que añade funcionalidades como el renderizado del lado del servidor (*Server-side rendering*) y la generación de páginas estáticas que mejoran el rendimiento.

4.2.4. Tecnologías utilizadas en el servidor de aplicaciones

Para el desarrollo del servidor de aplicaciones, se eligieron Node.js (OpenJS Foundation, 2024), Adonis.js (AdonisJS, 2024a) y TypeScript como las tecnologías principales.

Node.js provee un entorno de ejecución altamente eficiente para aplicaciones con operaciones intensivas de entrada/salida (I/O), como es el caso de este sistema, que procesa archivos CSV grandes y realiza múltiples interacciones con la base de datos en cada solicitud web. Node.js es una tecnología madura y ampliamente adoptada, que cuenta con una comunidad activa que provee soporte continuo.

Sobre Node.js, se utilizó Adonis.js, inspirado en otros *frameworks* populares de arquitectura Model-View-Controller (MVC) como *Laravel* (Laravel, 2024) y Ruby on Rails (Rails Core Team, 2024). Su enfoque opinionado proporciona una estructura clara que reduce la fatiga de decisiones. Adonis.js provee funcionalidades que aceleran el desarrollo, como un ORM robusto, un gestor de migraciones y APIs que simplifican la creación de *endpoints* y modelos. Adonis.js integra TypeScript de forma nativa, mejorando la calidad del código y reduciendo errores. Además, es ampliamente soportado por plataformas de *hosting*, lo que facilita su despliegue en entornos de producción.

4.3. Servidor de base de datos

La decisión de utilizar PostgreSQL (PostgreSQL Global Development Group, 2024) como base de datos se fundamenta en su naturaleza *Open Source*, su amplio reconocimiento en la industria y su probada eficacia en entornos de producción. PostgreSQL es conocido por su robustez, confiabilidad y el respaldo de una comunidad activa que continuamente mejora y actualiza la tecnología. Además, su popularidad garantiza una amplia compatibilidad con herramientas externas de análisis de datos y Business Intelligence (BI), lo que facilita la integración y el uso de estos recursos para futuras necesidades de análisis y reportes.

4.4. Procesamiento de trabajos asincrónicos

En el diseño de la aplicación *backend*, se identificó la necesidad de manejar tanto solicitudes HTTP como tareas que son más lentas y no requieren una respuesta inmediata al usuario.

Un ejemplo es la funcionalidad que permite al administrador, a través de una única acción, enviar un correo electrónico a cada docente solicitando que confirmen y actualicen sus datos. Si el envío de todos estos correos electrónicos se realizara de forma sincrónica dentro de una única solicitud HTTP, es probable que se superara el tiempo de espera (*timeout*) de la solicitud considerando que el envío de email requiere interactuar con un sistema externo. Además, gestionar esta operación sincrónicamente haría que la implementación fuera más compleja y menos fiable, ya que sería difícil garantizar que cada correo se envíe una sola vez, evitando duplicaciones.

Para abordar esto de manera eficiente, se optó por implementar el patrón *Background Jobs*, que propone delegar tareas largas o complejas a un proceso separado que se ejecuta de manera asíncrona, es decir, en segundo plano, sin bloquear el flujo principal de la aplicación.

4.4.1. Redis como base de datos secundaria

Redis (Redis Ltd., 2024) es una base de datos en memoria de tipo clave-valor ampliamente adoptada, reconocida por su velocidad en operaciones de lectura y escritura. En este proyecto, se utilizó Redis como base de datos secundaria específicamente para almacenar y gestionar los trabajos asincrónicos que la aplicación debe realizar.

Aunque podría haberse usado PostgreSQL para esta tarea, separar el almacenamiento de los trabajos asincrónicos evita que estos interfieran con el rendimiento de las consultas web. Además, dado que los trabajos asincrónicos no requieren la complejidad de una base de datos relacional con estructuras de tablas y relaciones, Redis, con su velocidad superior, resulta mucho más adecuada para manejar estas operaciones de manera eficiente.

Una de las principales razones detrás de la utilización de Redis es que Bull (OptimalBits, 2024), la biblioteca de manejo de trabajos asincrónicos *Open Source* más popular para Node.js, la requiere. Internamente, Bull almacena en Redis las tareas, asegurando la ejecución "al menos una vez" y manejando errores y reintentos. En el anexo D se describe cómo la librería utiliza las estructuras de datos de Redis para gestionar la cola de trabajos.

Finalmente, en el futuro, Redis podría utilizarse como una caché para que el servidor responda a ciertas solicitudes frecuentes más rápidamente, evitando la necesidad de interactuar con PostgreSQL en cada consulta.

4.4.2. Implementación de flujos asincrónicos en la aplicación

En la Figura 4.29, se presenta un diagrama de secuencia que detalla cómo los diferentes procesos del servidor de aplicaciones se comunican entre ellos a través de Redis para satisfacer la funcionalidad de envío de emails de forma eficiente para el usuario final.

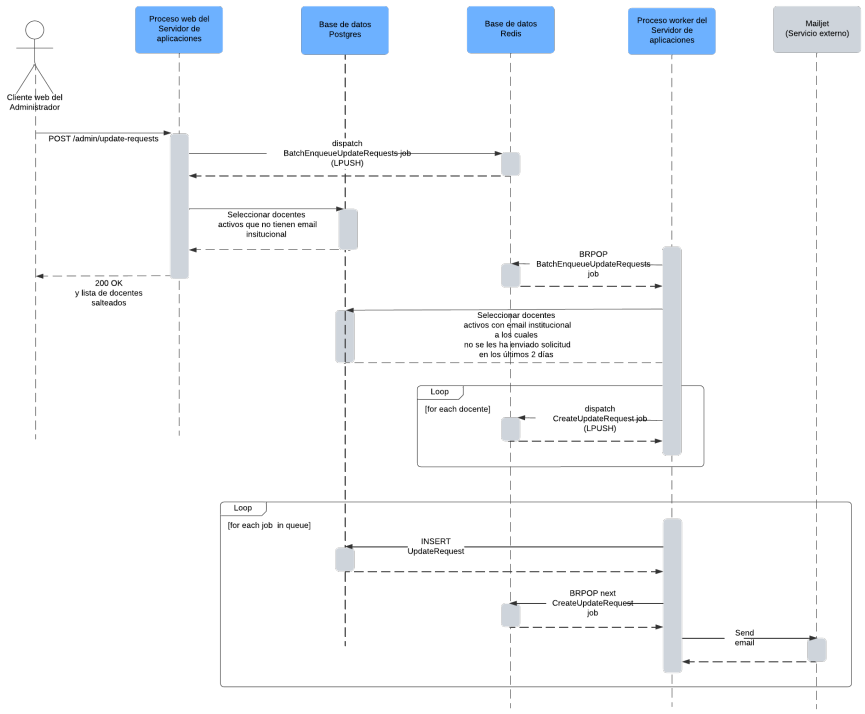


Figura 4.29: Diagrama de secuencia del envío de emails

1. El administrador interactúa con la aplicación web y, al confirmar la acción de envío de correos, el cliente web envía una solicitud HTTP al proceso web del servidor de aplicaciones.

2. El servidor recibe la solicitud y encola la tarea en la base de datos Redis, que actúa como la cola de trabajos encargada de almacenar las tareas pendientes de ejecución.

3. El servidor consulta la base de datos Postgres para obtener el listado de docentes que no tienen un correo electrónico institucional, ya que estos no podrán ser contactados. Esta información es crucial para que el administrador esté al tanto de quiénes no recibirán la solicitud.

4. El servidor responde a la solicitud HTTP del cliente de manera exitosa, incluyendo en el cuerpo de la respuesta el *array* de docentes sin correo institucional.

5. En paralelo, el proceso *worker* del servidor de aplicaciones monitoriza la cola de Redis y toma las tareas en el orden en que llegan. Cuando el *worker* recoge la tarea denominada `BatchEnqueueUpdateRequests`, primero

consulta en Postgres los docentes a quienes se les debe enviar el correo. Luego, para cada docente, encola un trabajo `CreateUpdateRequest` con el ID del docente como argumento.

6. El proceso *worker* continúa procesando los trabajos en la cola. Cuando se encuentra con una tarea `CreateUpdateRequest`, primero busca en la base de datos Postgres el registro del docente correspondiente al ID, realiza las validaciones necesarias, y genera una entrada en la tabla **update_-requests** para ese docente, incluyendo un *one-time access token* y una fecha de expiración. Finalmente, procede a enviar el correo electrónico a través del servicio externo Mailjet.

Aunque esta funcionalidad es la más crítica para ejecutarse de manera asincrónica, no es la única en el sistema. Otro ejemplo es un trabajo recurrente que se ejecuta una vez al día y cuya función es eliminar registros huérfanos, como actividades o entidades que no están referenciadas en ninguna actividad docente. Estos registros pueden crearse cuando los docentes llenan formularios y añaden nuevas actividades o entidades que el sistema no tenía previamente. Si un docente crea estos registros pero no se los asigna a sí mismo, y nadie más lo hace, el sistema los elimina automáticamente para mantener la base de datos limpia y evitar la acumulación de datos innecesarios.

4.5. Despliegue

En esta sección se describe la infraestructura y procesos involucrados en la ejecución del sistema en el entorno de producción. Se analizarán aspectos como la elección de la plataforma de *hosting*, la implementación de Continuous Deployment (CD), y las estrategias, tecnologías y patrones aplicados para asegurar el cumplimiento de los requerimientos y atributos de calidad relevados.

4.5.1. Docker

Docker (Docker Inc., 2024) es una una plataforma de software que permite crear, desplegar y gestionar aplicaciones junto con todas sus dependencias en contenedores aislados. Su utilización en este proyecto ofreció múltiples beneficios, entre ellos:

- **Simplificación del Entorno de Desarrollo**: Docker facilita la configuración y replicación exacta del entorno de desarrollo. Al proporcionar contenedores predefinidos con todas las dependencias necesarias, se eliminan problemas comunes relacionados con la configuración manual y las discrepancias entre diferentes sistemas operativos y configuraciones locales, lo que agiliza la incorporación de nuevos desarrolladores al proyecto.

- **Portabilidad y Flexibilidad en el Despliegue**: Durante el desarrollo del proyecto, fue necesario desplegar temporalmente el sistema en diferentes plataformas de *hosting* mientras se definía la infraestructura final. Docker facilitó el traslado de los componentes del sistema sin requerir modificaciones significativas, minimizando el re-trabajo.
- **Consistencia entre Entornos**: Docker reduce las diferencias en las configuraciones del sistema entre los entornos de desarrollo, *testing* y producción, evitando los errores que estas diferencias comúnmente suelen ocasionar.
- **Implementación No Intrusiva**: La integración de Docker no añade complejidad al proyecto y una vez integrado su utilización es opcional. Esto permite que el proyecto pueda ejecutarse tanto con o sin Docker, según las necesidades y capacidades específicas del entorno de despliegue.

4.5.2. Hosting en MiNube

4.5.3. Hosting en MiNube

MiNube (Antel, 2024b) es un servicio de nube proporcionado por Antel (Antel, 2024a), la empresa de telecomunicaciones Uruguaya. Este servicio ofrece Infrastructure as a service (IaaS), permitiendo a los usuarios desplegar y gestionar aplicaciones en la nube ya que ofrece servidores, almacenamiento, y recursos de red. Dentro de MiNube, uno de los principales servicios es Elastic Cloud (Elastic N.V., 2024), una solución de Platform as a Service (PaaS) que se ofrece a través de Jelastic (Virtuozzo, 2024). Jelastic es una herramienta *whitelabel* que permite a proveedores de servicios en la nube, como Antel, ofrecer un entorno PaaS bajo su propia marca.

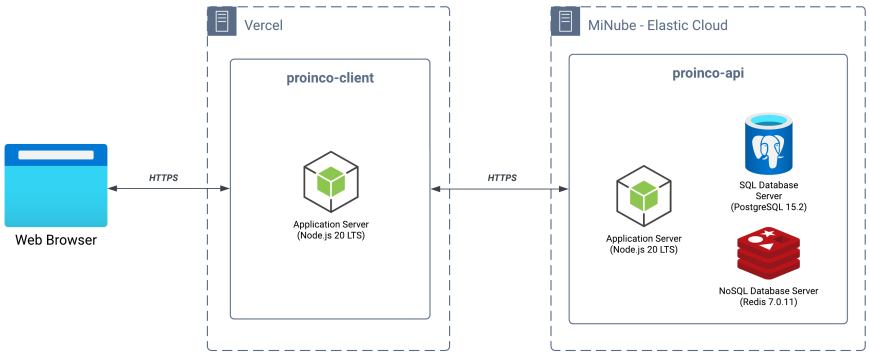


Figura 4.30: Diagrama de componentes

Se seleccionó MiNube para el despliegue de esta aplicación por dos razones principales. En primer lugar, el RNF 3 - Datos almacenados en Uruguay exige que la base de datos se encuentre alojada en territorio uruguayo, lo que llevó a descartar proveedores populares como AWS (Amazon, 2024) y Heroku (Salesforce, 2024a), los cuales no ofrecen esta opción. MiNube, al alojar toda su infraestructura dentro del país, permite cumplir con el requerimiento.

En segundo lugar, el RNF 1 - Utilización de software gratuito exige que el *hosting* no debe generar costos operativos, lo que elimina la mayoría de las alternativas disponibles, ya que los niveles gratuitos de los servicios confiables no son suficientes para un sistema *production-ready* que garantice escalabilidad, resiliencia y respaldo de datos. Sin embargo, FING mantiene un convenio con Antel que proporciona créditos para el uso de MiNube, lo que permite eliminar los costos operacionales y asegurar el cumplimiento de este requisito. Ver Figura 4.30.

Dominio de la aplicación

En relación al dominio de la aplicación, dado que INCO no dispone de un dominio propio adecuado para este proyecto y considerando las restricciones presupuestarias que impiden la adquisición de uno, se optó por utilizar el dominio predeterminado[1] proporcionado por MiNube durante el despliegue de la aplicación.

Escalabilidad

Elastic Cloud ofrece un entorno altamente escalable gracias a la integración de Jelastic. En esta infraestructura, se utiliza un *shared load balancer* como el único punto de entrada a la aplicación, distribuyendo eficientemente el tráfico entre las instancias del *backend*.

La infraestructura configurada en Elastic Cloud cuenta con mecanismos de *auto-scaling* tanto para el servidor de aplicaciones como para la base de datos PostgreSQL y Redis. Este *auto-scaling* permite ajustar automáticamente los recursos asignados a cada componente según la demanda, con límites mínimos y máximos de cómputo reservados para asegurar tanto la disponibilidad como la eficiencia en el uso de los recursos.

[1] `https://inco-rrhh-api.web.elasticloud.uy`

Backups

Para garantizar la integridad de los datos, se han configurado copias de seguridad diarias de la base de datos PostgreSQL, que se ejecutan automáticamente todos los días a las 3:00 AM.

Despliegue continuo

Además, un proceso de despliegue continuo para automatizar el despliegue de actualizaciones en la infraestructura cuando se hacen actualizaciones al código base. Esta inversión no solo agiliza la entrega de nuevas funcionalidades al permitir implementaciones más frecuentes, sino que reduce el riesgo de errores manuales y elimina la necesidad de otorgar acceso a la infraestructura a todos los futuros desarrolladores de la aplicación.

Para implementar el despliegue continuo, Elastic Cloud fue configurada para que verifique automáticamente el repositorio en intervalos regulares (cada minuto) y cada vez que se detecte un nuevo *commit* en la rama *main* del repositorio se despliegue el cambio a los ambientes de *testing* y producción. Para habilitar esta automatización, se generó un *token* de acceso en GitHub (GitHub Inc., 2024a), que fue integrado en Elastic Cloud para permitirle el acceso automatizado al repositorio.

4.5.4. PM2 como gestor de procesos

Para garantizar la escalabilidad, resiliencia y estabilidad del sistema, es fundamental considerar aspectos como la recuperación ante fallos y el uso eficiente de los recursos físicos disponibles. Si bien Elastic Cloud proporciona la infraestructura básica, fue necesario configurar una capa adicional que gestione y orqueste los procesos del *backend* de la aplicación. Es por ello que se ha integrado PM2 (Keymetrics, 2024) a la aplicación *backend*, un popular gestor de procesos de código abierto diseñado específicamente para aplicaciones Node.js.

PM2 permite ejecutar los múltiples procesos de la aplicación, tanto los procesos web como procesos workers. También facilita el monitoreo mediante un panel de control para visualizar métricas clave, como el uso de Central Processing Unit (CPU) y memoria por proceso.

Una de las principales razones por las que se introdujo PM2 es que permite el reinicio automático de las aplicaciones en caso de que estas se detengan inesperadamente, fundamental para la resiliencia de la infraestructura ya que Elastic Cloud no resuelve esta necesidad por defecto.

4.6. Manejo de sesiones

En este proyecto se utilizaron las tácticas de seguridad autenticación y autorización de usuarios. Los administradores se autentican ingresando su usuario y contraseña, y su acceso a recursos se verifica según su rol. Los docentes se autentican mediante un enlace de acceso temporal que reciben por correo.

4.6.1. *Tokens* opacos

Para la autenticación de ambos tipos de usuarios se utilizó el patrón de manejo de sesiones basado en *tokens* opacos. En este enfoque, el servidor genera *tokens* que se almacenan en la base de datos y se validan en cada petición HTTP, satisfaciendo las necesidades de seguridad, interoperabilidad y simplicidad de implementación.

El uso de *tokens* opacos fue elegido tras evaluar otras opciones, como JSON Web Token (JWT) (IETF, 2024), sesiones tradicionales y servicios de autenticación de terceros.

- **JWT**: Los JWT son populares en aplicaciones modernas ya que permiten almacenar datos directamente en el *token* como la información del usuario actual, lo cual puede resultar más eficiente ya que permiten evitar una consulta extra a la base de datos en cada petición HTTP. Sin embargo, presentan dificultades cuando se requiere invalidar un *token* antes de su expiración natural, ya que los JWT son difíciles de revocar una vez emitidos. En cambio, los *tokens* opacos, al ser simplemente identificadores almacenados en el servidor, pueden ser invalidados en cualquier momento, ofreciendo un mayor control sobre las sesiones.
- ***Cookies*:** En una autenticación basada en *cookies*, el cliente almacena la información de sesión en una *cookie*, que se envía al servidor con cada petición para su validación. Sin embargo, una limitación de este enfoque es que reduce la interoperabilidad con aplicaciones móviles, sistemas de terceros, y otros clientes que no manejan *cookies* de la misma manera que un navegador web. En nuestra solución basada en *tokens* opacos, el cliente web puede almacenar el *token* en una *cookie*, pero lo que se envía al servidor para la validación no es la *cookie* en sí, sino únicamente el contenido de ella, esto es, el *token* en texto plano, lo que permite una mayor flexibilidad y compatibilidad con diversos tipos de clientes.
- **Servicios de terceros**: Aunque los servicios de autenticación de terceros como Auth0 (Auth0 Inc., 2024) pueden ofrecer soluciones robustas, muy seguras, con muchas funcionalidades y fáciles de implementar, incurren en costos que el proyecto no se encuentra en condiciones de asumir. Además, los requerimientos de sesión relevados son sencillos por lo que no ameritan la complejidad agregada que conlleva la integración con plataformas

externas.

4.6.2. Autenticación de administradores

Para la implementación del patrón de autenticación para los usuarios administradores, se hizo uso del *sistema de autenticación API* (AdonisJS, 2023) provisto por Adonis.js. El *framework* provee funcionalidades que facilitan la generación, almacenamiento, validación, expiración y revocación de los *tokens*, contribuyendo a que el proceso esté implementado de una forma segura y fácil de mantener. Esto es porque, por ejemplo, las primitivas de seguridad de Adonis.js están diseñadas para proteger contra ataques de *timing* y ataques de fijación de sesión. Además, en Adonis.js las credenciales se almacenan mediante un proceso de *hashing* unidireccional, lo que significa que incluso si alguien accediera al servidor y a la base de datos, no podría desencriptarlas ni leer su contenido. (AdonisJS, 2024b)

Los *tokens* opacos (*opaque access tokens*) se estructuran de la siguiente manera:

- Un *token* se representa mediante un valor aleatorio seguro, seguido de un *checksum* CRC32.
- Un *hash* del valor del *token* se almacena en la base de datos, y este *hash* se utiliza para verificar el *token* en el momento de la autenticación.
- El valor final del *token* se codifica en base64.

Para soportar la funcionalidad de autenticación, se implementó una estructura de tablas que incluye la tabla **usuarios**, donde se almacenan su correo, contraseña encriptada y uno de los tres posibles roles que se utilizan para la autorización.

Para soportar que cada usuario administrador pueda mantener sesiones abiertas en diferentes dispositivos simultáneamente, se creó la tabla **api_tokens**, la cual contiene una clave foránea que referencia a la tabla **usuarios**, y un *token* opaco de tipo *string*. Las columnas **name** y type son parte del esquema requerido por la librería de autenticación utilizada, diseñada para gestionar diversos tipos de *tokens* con distintos alcances, aunque en este sistema esto no es necesario por lo que estas columnas siempre tienen el mismo valor en todas las filas. Ver Figura 4.31.

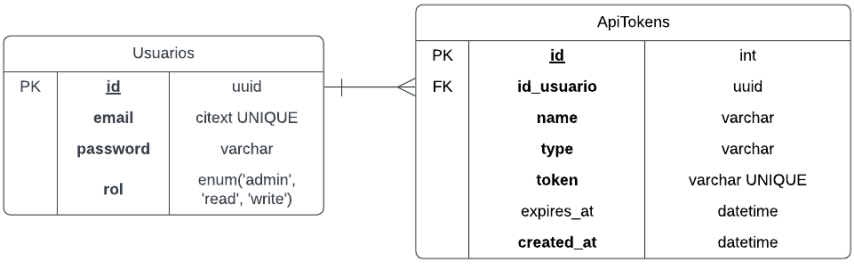


Figura 4.31: Tablas utilizadas para la autenticación de administradores

En la Figura 4.32, se presenta el diagrama de secuencia de la interacción cliente-servidor que describe el protocolo de autenticación de los administradores.

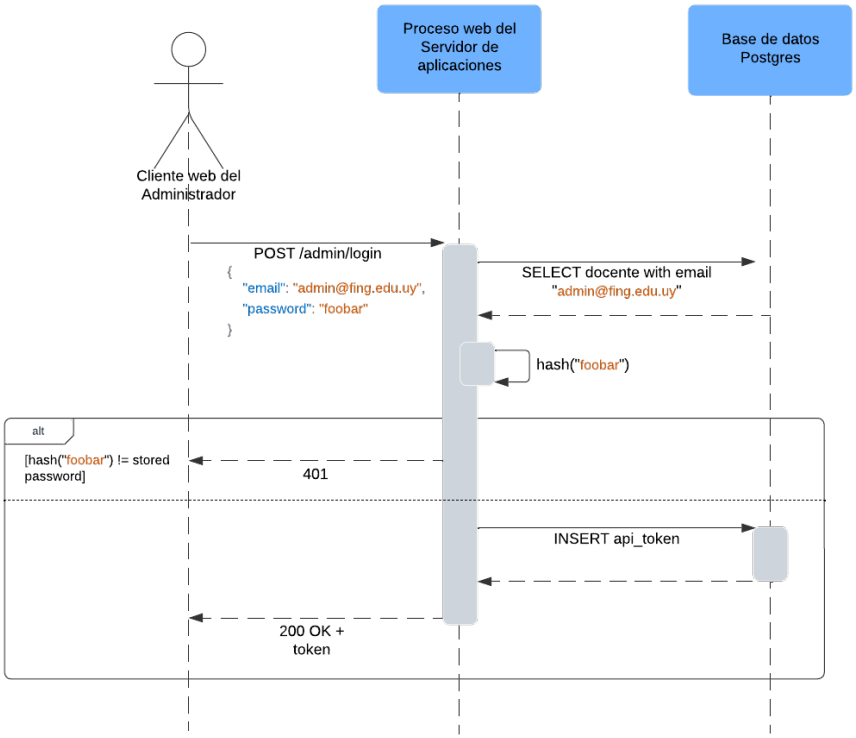


Figura 4.32: Diagrama de secuencia del proceso de creación de tokens

Cuando un administrador intenta iniciar sesión, envía sus credenciales (correo electrónico y contraseña) al servidor mediante una solicitud HTTP `POST` al *endpoint* de autenticación. La aplicación busca el registro con el email provisto en la tabla de **usuarios** de la base de datos Postgres. Dado que las contraseñas se almacenan con *hashing* unidireccional, para validar si la contraseña provista es la correcta, primero se debe aplicar el mismo algoritmo de *hashing* a la contraseña provista, y luego comparar el resultado con el valor que se encuentra en la base de datos.

Si la comparación es positiva, el servidor genera un *token* opaco, un identificador único que no contiene información del usuario ni de la sesión, y se almacena en la base de datos junto con los detalles de la sesión. El *token* se envía al cliente, quien lo almacenará en una *cookie*.

En cada solicitud posterior que requiera autenticación, el cliente envía el *token* opaco en los encabezados de la solicitud HTTP. El servidor valida este *token* consultando la base de datos para verificar su existencia y vigencia. Si el *token* es válido, el servidor permite el acceso al recurso solicitado; de lo contrario, responde con un error de autenticación. Ver Figura 4.33.

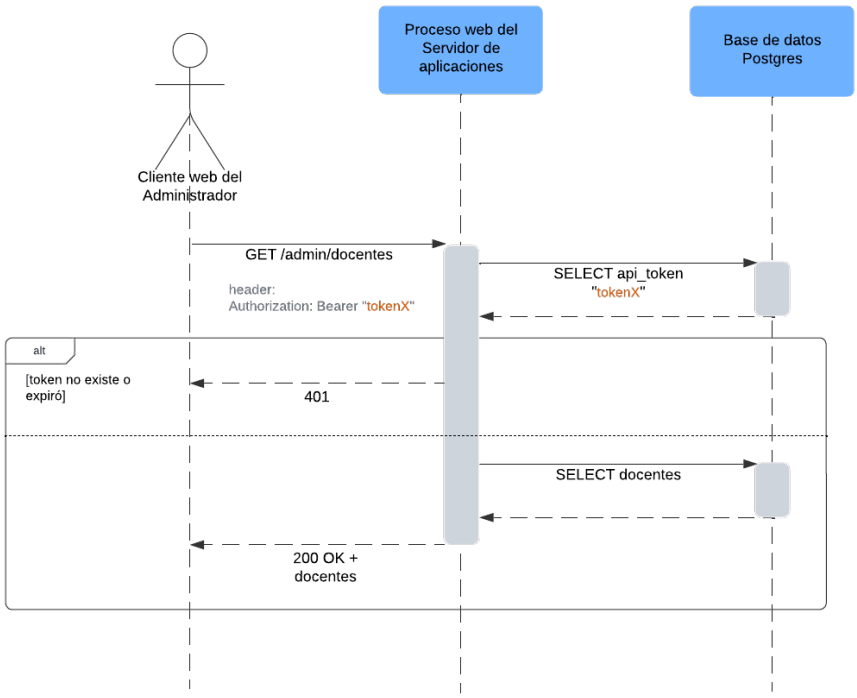


Figura 4.33: Diagrama de secuencia del proceso de validación de tokens

Cuando el usuario cierra la sesión, el servidor revoca el *token* marcándolo como expirado en la base de datos, lo que asegura que el *token* ya no pueda ser utilizado para acceder a recursos de la aplicación.

4.6.3. Autenticación de docentes

Para la implementación de la autenticación de los usuarios docentes, se creó la tabla **update_requests**, que incluye una clave foránea hacia la tabla **docentes**. El ID, de tipo `UUID`, actúa como el *token* opaco en este caso. Ver Figura 4.34.

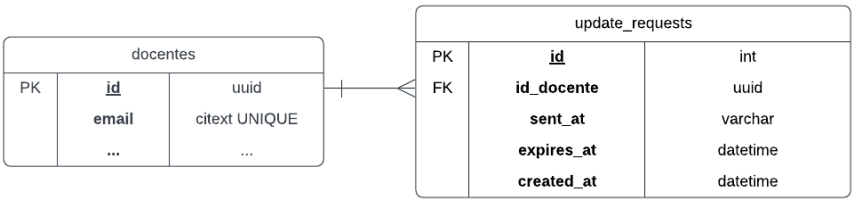


Figura 4.34: Tablas utilizadas para la autenticación de docentes

Cuando un docente recibe un correo electrónico solicitando la actualización de su información, el mensaje incluye un enlace a la ruta del *frontend*: `/docentes/<idDocente>/auth`, en el cual se añade un *token* como parámetro de consulta. Al seguir el enlace, la aplicación web almacena el *token* en una *cookie*, que se adjunta en los encabezados de las siguientes solicitudes enviadas al *backend*. Esta última interacción cliente-servidor se resume en el diagrama de secuencia presentado en la Figura 4.35.

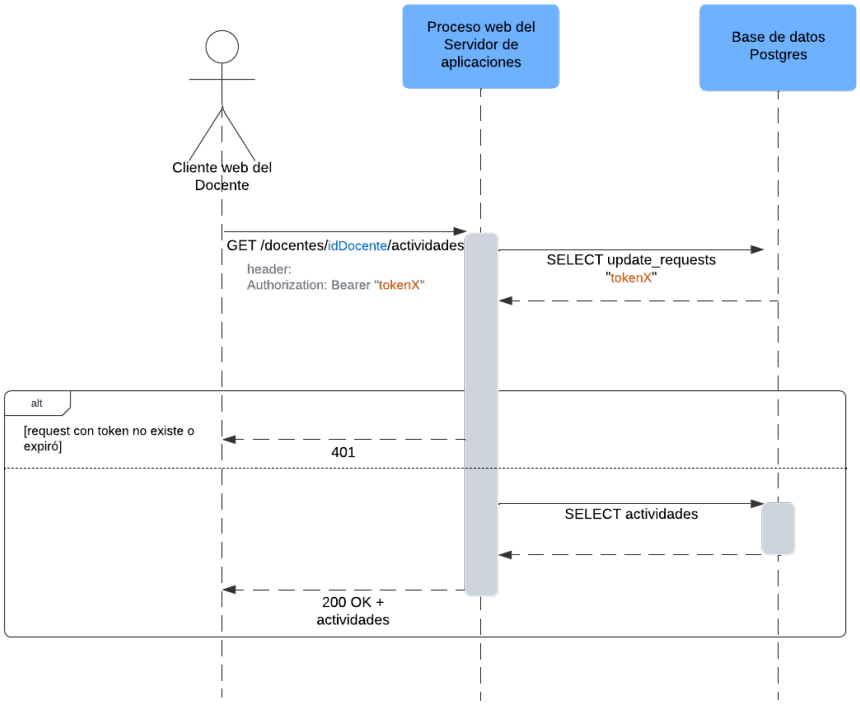


Figura 4.35: Diagrama de secuencia del proceso de validación de tokens de docentes

4.7. Plataforma para el envío de emails

El éxito de la aplicación depende en gran medida del cumplimiento del requerimiento funcional que establece que los docentes deben recibir correos electrónicos solicitándoles que verifiquen y actualicen sus actividades. La entrega efectiva de estos correos es esencial para lograr la participación activa de los docentes y la actualización de la información. Por lo tanto, la elección del mecanismo de envío de correos electrónicos fue un aspecto muy importante en el diseño del sistema.

4.7.1. Complejidades del envío de emails

El envío de correos electrónicos, aunque parece un requerimiento sencillo, no es trivial de implementar sin uso de plataformas externas especializadas en el envío de correos electrónicos, como SendGrid (Twilio SendGrid, 2024), Mailgun (Mailgun Technologies Inc., 2024) o Mailjet (Mailjet, 2024). A continuación, se listan algunas de las complejidades relacionadas con el envío de correos automatizados directamente desde la aplicación propia.

- **Fiabilidad en la Entrega**: El envío de correos electrónicos directamente desde un servidor propio conlleva el riesgo de que los mensajes sean bloqueados o marcados como *spam* por los proveedores de servicios de correo electrónico, debido a los estrictos filtros de seguridad que estos aplican.
- **Autenticación y Verificación**: La configuración de mecanismos de autenticación como Sender Policy Framework (SPF), DomainKeys Identified Mail (DKIM) y Domain-based Message Authentication, Reporting and Conformance (DMARC) es esencial para asegurar que los correos electrónicos enviados sean considerados legítimos por los servidores de correo receptores. Sin estos mecanismos, los correos electrónicos tienen una alta probabilidad de no llegar a la bandeja de entrada del destinatario o ser rechazados automáticamente. Implementar y mantener estos mecanismos correctamente es un proceso complejo que requiere un conocimiento técnico especializado, tiempo y acceso a la configuración Domain Name System (DNS) del dominio. Dado que el proyecto no cuenta con un dominio propio y las restricciones presupuestarias impiden la adquisición de uno, esta opción no era viable.
- **Seguridad y Conformidad**: El contenido de los correos electrónicos, especialmente si incluye información sensible, debe ser transmitido de manera segura. Esto requiere la implementación de cifrado durante el tránsito (usualmente a través de TLS) y el cumplimiento de regulaciones de privacidad de datos, como el General Data Protection Regulation (GDPR). Las plataformas externas suelen garantizar la seguridad del contenido mediante cifrado, cumpliendo con las regulaciones correspondientes. Al usar un servicio especializado, ellos se encargan de gran parte de la administración de la reputación, asegurando que sus direcciones Internet Protocol (IP) y dominios de envío mantengan una alta reputación. Aunque se configure el servicio para enviar los correos desde un dominio propio, el servicio utiliza su infraestructura de envío, lo que significa que los correos se envían a través de sus servidores, que están optimizados para evitar que los correos sean marcados como *spam*.
- **Mantenimiento Continuo y Complejidad**: Mantener un servidor de correo propio requiere un esfuerzo constante para adaptarse a cambios en las políticas de los proveedores de servicios de correo electrónico. Esto incluye la gestión de actualizaciones, la resolución de problemas de entrega,

y la administración de la reputación del dominio, lo cual puede resultar costoso y requiere de un equipo especializado.

Dado estos desafíos, la decisión de utilizar una plataforma externa especializada para el envío de correos electrónicos fue evidente. Esta elección contribuye al correcto funcionamiento de la funcionalidad, facilita su implementación y reduce significativamente los costos de mantenimiento.

4.7.2. Elección del servicio externo

Una vez analizados los desafíos técnicos asociados al envío de correos electrónicos, se procedió a elegir una plataforma que cumpliera con los requisitos específicos del proyecto. La selección se basó en los siguientes criterios clave:

- **Plan gratuito**: Debe ofrecer un plan gratuito, no un período de prueba, que permita el envío de al menos cuatrocientos correos electrónicos por mes, considerando que la base de datos incluye hasta doscientos docentes activos y se prevé un máximo de dos envíos mensuales por docente como se describe en RNF 7 - Usuarios concurrentes y RNF 18 - Cuota de correos electrónicos.
- **Reputación:** Dado que el sistema maneja datos sensibles, es fundamental elegir una plataforma con una reputación sólida y que ofrezca términos y condiciones claros que aseguren un manejo responsable y seguro de los datos.
- **Monitoreo**: Aunque no crucial, es valioso el disponer de herramientas que permitan verificar si cada correo electrónico ha sido entregado o si hubo algún fallo en el envío, para que el administrador pueda entender si tiene las direcciones correctas de los docentes y si el sistema está correctamente configurado.

A continuación, se presenta una tabla comparativa de algunas de las plataformas evaluadas.

Plataforma	**Plan Gratuito**	**Reputación**	**Monitoreo**
SendGrid	Hasta 100 correos/mes	Alta	Limitado en plan gratuito
Mailgun	Hasta 100 correos/mes	Alta	Limitado en plan gratuito
Mailjet	Hasta 6,000 correos/mes	Alta	Incluido en el plan gratuito
Gmail	N/A ∗	Alta	Limitado

∗ Para utilizar Gmail (Google, 2024a) con fines de envío masivo, es necesario contar con un dominio propio y registrarse como empresa. Además, Gmail requiere la configuración de autenticación avanzada (como SPF y DKIM) para evitar que los correos sean marcados como *spam*, lo que añade complejidad al proceso.

Tras evaluar todas las opciones, Mailjet fue seleccionada como la plataforma más adecuada para el envío de correos electrónicos, ya que cumple con todos los criterios establecidos. Su plan gratuito ofrece una cuota que excede ampliamente las necesidades del proyecto, sin comprometer la fiabilidad ni las funcionalidades de monitoreo.

4.7.3. Implementación de la integración con Mailjet

La aplicación *backend* se integra con Mailjet mediante el servidor SMTP que la plataforma provee. Mailjet luego utiliza el protocolo Simple Mail Transfer Protocol (SMTP) para entregar los correos a los docentes.

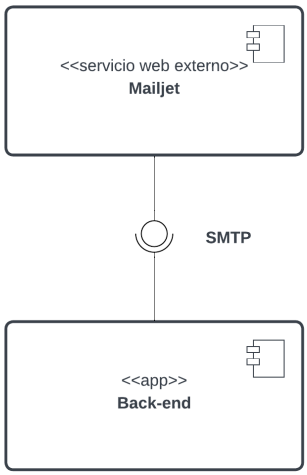


Figura 4.36: Diagrama de conectores y componentes (Mailjet - Backend)

La integración con Mailjet no se realiza mediante la invocación manual de llamadas SMTP desde el código base de la aplicación. En su lugar, se emplean librerías que encapsulan y abstraen los detalles específicos de la API, permitiendo una implementación más sencilla y mantenible. Esta capa de abstracción garantiza que, en caso de ser necesario un cambio de servicio externo en el futuro, la modificación implique cambios mínimos en una única parte del código. (Mailjet, 2023)

Específicamente, se utiliza el paquete `@Adonis.js/mail` (AdonisJS, 2024c) de Adonis.js, el cual a su vez está construido sobre Nodemailer (Nodemailer, 2024), una reconocida librería de JavaScript para el envío de correos electrónicos desde aplicaciones Node.js. Nodemailer soporta diversos servicios de correo electrónico, entre ellos Mailjet. Ver Figura 4.36.

4.8. Carga y procesamiento de archivos CSV

El sistema depende en gran medida de la carga de datos provenientes de archivos CSV suministrados por otras áreas. En su estado actual, la aplicación soporta la importación de cinco tipos distintos de archivo, con la posibilidad de añadir más formatos en el futuro de ser necesario. A raíz de esto, se decidió invertir tiempo en desarrollar un mecanismo robusto, reutilizable y de fácil mantenimiento para gestionar la carga de archivos. El mecanismo debe permitir la adición de nuevos tipos de archivos sin necesidad de duplicar código. Por lo tanto, debe proveer un *framework* que resuelva aspectos como la validación de los datos, el manejo de errores, y la retroalimentación al usuario sobre posibles fallos en la estructura o contenido de los archivos CSV cargados. A su vez, el mecanismo debe ser escalable y eficiente a medida que los archivos crecen en tamaño con el paso del tiempo.

Dado que es común que los administradores suban versiones corregidas de los archivos y que, periódicamente, suban archivos similares con nuevas filas, es fundamental que el procesamiento de archivos CSV sea idempotente. Esto significa que, al subir un archivo dos veces, no deben generarse registros duplicados en la base de datos.

La estructura de tablas diseñada para soportar la funcionalidad de importación de datos de manera genérica incluye las tablas **file_imports** y **file_import_errors**, representada en la Figura 4.37.

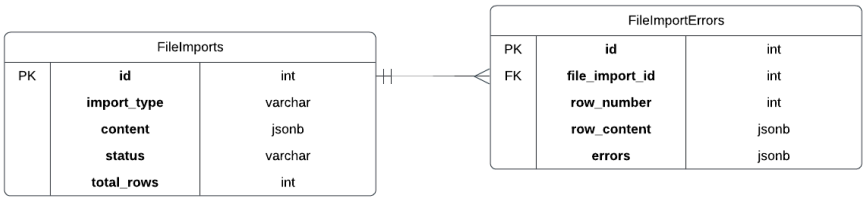


Figura 4.37: Tablas utilizadas durante el procesamiento de archivos CSV

Un registro en la tabla **file_imports** representa la acción de un administrador al cargar un archivo en el sistema. La tabla **file_import_errors** almacena los errores de validación identificados durante el procesamiento del archivo, donde cada error corresponde a una fila inválida del archivo. La columna **import_type** identifica el tipo específico de archivo que se está importando, mientras que la columna **content** almacena el contenido del archivo en formato JavaScript Object Notation (JSON). La columna **status** refleja el estado del procesamiento y puede tomar los valores: ***pending***, ***processing***, ***completed*** o ***failed***. Este

campo sirve para evitar condiciones de carrera, permitiendo evitar que un archivo sea procesado accidentalmente por más de un hilo concurrentemente. El estado ***failed*** indica que ocurrió un error en la aplicación que impidió procesar todas las filas correctamente, lo que requiere la intervención de un desarrollador.

La persistencia de los archivos importados y de los errores asociados, junto con los patrones utilizados, permiten que la funcionalidad de importación pueda, en un futuro, modificarse para ser ejecutada de manera asíncrona para un mayor rendimiento y escalabilidad, sin necesidad de modificar la arquitectura subyacente. Además, este enfoque ofrece la ventaja de que, en caso de que un administrador refresque accidentalmente la página durante el proceso de carga, los resultados previos de la importación siguen siendo accesibles. Asimismo, almacenar este historial facilita la labor de los desarrolladores, quienes pueden acceder a registros detallados de todas las cargas realizadas, lo que resulta útil durante la investigación de defectos y la recuperación de datos en caso de modificaciones o eliminaciones accidentales en el sistema.

Desde la perspectiva de la arquitectura del software, la aplicación *backend* dispone de un único *endpoint* genérico (`POST /admin/file-imports`) que permite la importación de todos los tipos de archivos soportados. Esto asegura que nuevos tipos de archivos puedan ser agregados sin la necesidad de definir nuevas rutas cada vez.

Cuando se recibe un archivo, el *endpoint* se encarga de realizar su lectura y *parsing* utilizando la librería **fast-csv** (C2FO, 2024), una solución ampliamente reconocida para el procesamiento eficiente de archivos CSV en aplicaciones Node.js. Esta librería se combina con la API de `Streams` de Node.js, lo que optimiza el uso de memoria al permitir la lectura del archivo en pequeños fragmentos, en lugar de cargarlo completamente en memoria.

Tras la lectura inicial del archivo, se persiste en la tabla **file_imports** y se invoca una tarea para comenzar a procesar su contenido.

Para estructurar el código de manera que siga el principio "Abierto para la extensión y cerrado para la modificación" (*Open/Closed Principle*), se implementa el patrón *Strategy*. Este patrón permite encapsular la lógica de negocio específica para cada tipo de archivo en clases separadas, promoviendo la modularidad del sistema.

Este patrón se materializa a través de la interfaz `Importer`, que define el método `process`, encargado de recibir un objeto `fileImport` que encapsula los detalles del archivo a procesar.

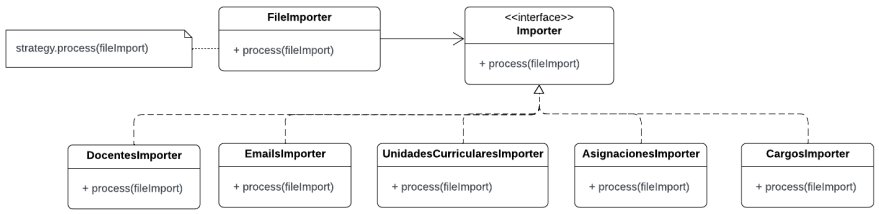


Figura 4.38: Diagrama de clases del procesamiento de archivos CSV

La clase `FileImporter` actúa como coordinador del proceso de importación. Su función es inicializar la estrategia adecuada (es decir, la subclase de `Importer`) según el tipo de archivo que se esté procesando. Para cada tipo de archivo soportado, como `DocentesImporter`, `EmailsImporter`, `UnidadesCurricularesImporter`, `CargosImporter` y `AsignacionesImporter`, existe una implementación específica de la interfaz `Importer`. Estas subclases son las encargadas de ejecutar el procesamiento del archivo, de acuerdo con las reglas de negocio correspondientes a cada tipo. Ver Figura 4.38.

El diseño implementado da como resultado un código fácil de entender, altamente extensible y reutilizable. Además, es especialmente eficiente en términos de rendimiento, ya que permite procesar archivos de hasta 10.000 filas en menos de 30 segundos, garantizando la escalabilidad del sistema sin comprometer la velocidad.

4.8.1. Procesamiento de datos importados

A continuación, se presentan los archivos soportados, su origen, los problemas de integridad de datos identificadas, y las consideraciones que el administrador debe tener en cuenta durante el proceso de carga.

Docentes (*INCODAT*)

El archivo *Docentes.csv* es provisto por el equipo de RRHH de FING y es comúnmente conocido como *INCODAT*. Este archivo contiene información básica de cada docente, como el correo personal, cédula de identidad, nombre, teléfono, fecha de ingreso a la institución y fecha de nacimiento.

Se ha observado que en ocasiones las columnas correspondientes al teléfono y correo electrónico se encuentran intercambiadas en el archivo, e incluso presentando datos arbitrarios en algunos casos. Ante esta situación, el sistema marca la fila como inválida y notifica al administrador para que corrija manualmente,

ya que ciertas inferencias automáticas serían demasiado complejas y propensas a errores.

Por otro lado, cabe mencionar que el archivo *INCODAT* incluye registros del personal administrativo, que no son relevantes para el sistema, que debe ser capaz de identificar y excluir estas filas. Tras un análisis exhaustivo de los archivos proporcionados, se observó que el personal administrativo generalmente no tiene una fecha de ingreso en la institución, lo que permitió implementar una regla para ignorar las filas correctamente.

Otra consideración a tener en cuenta es que el archivo *INCODAT* presenta una única columna *NOMBRE*, que incluye tanto el nombre como el apellido del docente, separados por una coma. Dado que el modelo de datos del sistema diferencia entre estos dos campos, la aplicación procesa esta columna dividiéndola en nombre y apellido utilizando la coma como delimitador. Por lo tanto, es crucial que el archivo *INCODAT* mantenga este formato para asegurar una correcta inferencia y asignación de los datos.

Por último, para asegurar la idempotencia y evitar que el administrador obtenga errores innecesariamente al cargar un mismo archivo varias veces, se utiliza la cédula como identificador único de los docentes. Esto permite que, al procesar una fila, si el docente con esa cédula ya existe en la base de datos, sus datos sean actualizados en lugar de intentar insertar un nuevo registro, previniendo duplicaciones.

Emails

El archivo *Emails.csv* consiste de solamente dos columnas, correspondientes al email y a la cédula. Su propósito es completar la base de datos con los correos electrónicos institucionales de cada docente, los cuales son necesarios para enviarles solicitudes de actualización de datos. Esto se debe a que el archivo *INCODAT* solo proporciona el correo personal.

El administrador debe construir este archivo manualmente, combinando el archivo *INCODAT* con los correos institucionales, que se obtienen de la Unidad de Recursos Informáticos (URI).

El procesamiento de este archivo no presenta mayores dificultades, más allá de la validación del formato de los correos electrónicos y la unicidad de ambos campos, para evitar registros duplicados.

Cargos y DTs (*INCOHIST*)

Este archivo, proporcionado por el equipo de RRHH de FING quien lo conoce como *INCOHIST*, es el más extenso, ya que incluye el historial completo de todos los cambios de cargos, dedicaciones, extensiones o reducciones horarias que han experimentado los docentes desde 1977.

A través de la subida de este archivo, se busca popular las tablas **dts**, **docente_departamento_cargo** y **cargos** del sistema. El archivo presenta las columnas **NOMBRE** (nombre y apellido del docente), **CÉDULA** (del docente), **GRADO**, **HORAS**, **PROVISIÓN** (tipo de registro), **DENOMINA** (descripción del cargo para el docente, si aplica), **POSESIÓN** (fecha de inicio) y **CESE** (fecha de fin).

Si bien el archivo *INCOHIST* plasma todo el historial de cambios de los cargos y dedicaciones de los docentes, el objetivo del sistema es registrar un historial consolidado, evitando almacenar cada modificación individual que estos experimenten. Por ejemplo, si un docente renueva la misma dedicación en varias ocasiones a lo largo de cinco años, *INCOHIST* presentará una fila por cada cese y/o renovación. Sin embargo, el sistema debe generar un único registro que cubre todo el período, con la fecha de inicio correspondiente a la primera asignación y la fecha de fin a la última, evitando almacenar todas las renovaciones intermedias que figuran en el archivo original. Es por eso que, a medida que se procesa el archivo, el sistema consolida y actualiza los registros para construir esta visión simplificada del historial del docente.

INCOHIST es, sin duda, el archivo cuyo procesamiento resulta más complejo, no solo debido a las transformaciones necesarias para adaptar los datos al modelo de tablas del sistema, sino también por la gran cantidad de datos irrelevantes para la aplicación, las inconsistencias y errores que contiene. El sistema es capaz de identificar y, cuando es posible, corrige automáticamente las inconsistencias para asegurar la integridad de los datos. A continuación, se ejemplifican algunos de los desafíos que presentan los datos y cómo el sistema los soluciona.

Uno de los problemas identificados al estudiar de cerca este archivo se relaciona con las cédulas de identidad, ya que en algunos casos hasta los últimos tres dígitos están alterados. Esto debe ocurrir debido a que el software que utiliza el equipo de RRHH, probablemente Excel, interpreta las celdas con puntos decimales (".") en el número de cédula como decimales, lo que resulta en un redondeo automático de los dígitos finales. Por ejemplo, un docente con cédula 1.831.699-6 figura incorrectamente como 1831700 en el archivo *INCOHIST*. Si no se resolviera este problema, al intentar subir el archivo, el sistema no podría procesar ninguna fila, ya que determinaría que los docentes no están registrados correctamente en el sistema debido a las inconsistencia en las cédulas. Para mitigar este inconveniente, la aplicación busca a los docentes utilizando solo los primeros cuatro dígitos de la cédula y verificando la correspondencia con el primer apellido. Esta solución ha demostrado ser efectiva con los datos

procesados hasta el momento, evitando que el administrador tenga que corregir manualmente las cédulas antes de cargar el archivo.

Otro de los desafíos encontrados es que el archivo no siempre respeta un orden cronológico. En la mayoría de los casos, la lógica de procesamiento de cada fila ha sido diseñada para corregir automáticamente inconsistencias relacionadas con el desorden temporal entre filas asociadas. Sin embargo, existe un caso particular en el que esto no es posible: ceses y renuncias. Esto se debe a que las filas que representan ceses o renuncias suelen aparecer antes que la fila correspondiente al inicio del cargo del docente, y no especifican una fecha de inicio. En tal situación, el sistema no puede procesar la renuncia de un cargo que aún no ha sido registrado, ya que la fecha de inicio es un dato obligatorio para completar el registro en la base de datos. Para resolver este inconveniente, la aplicación reordena el archivo *INCOHIST* antes de procesarlo, asegurando que las filas correspondientes a ceses y renuncias se procesen al final. Este enfoque evita que el administrador tenga que manipular manualmente el archivo antes de la carga.

Además, se observó que las fechas dentro de una misma fila a menudo presentan inconsistencias. Un caso común es cuando la fecha de posesión es posterior a la de cese en la misma fila. Tras analizar los casos en los que se identificó este error, se concluyó que probablemente se trata de un error humano en el que se confunden ambas columnas y se ingresan de manera invertida. Por esta razón, el sistema asume que es un error de registro y corrige automáticamente el orden de las fechas.

Otra situación relevante en el procesamiento de este archivo tiene que ver con las renovaciones de DTs. En el archivo *INCOHIST*, es común encontrar una fila que representa la interrupción de una DT, seguida de otra fila que representa la renovación de la misma, con una fecha de posesión uno o dos días posterior a la fecha de cese. Para manejar estos casos, el sistema implementa un umbral de tolerancia de tres días. Si se detecta un cese seguido de una nueva posesión dentro de ese período, el sistema lo trata como una renovación, combinando ambos registros en lugar de crear entradas separadas para cada una.

Por último, un aspecto importante fue garantizar que este archivo pueda subirse varias veces a medida que se actualiza con nuevas entradas a lo largo del tiempo, sin afectar las modificaciones manuales realizadas por el administrador a través de la aplicación. Específicamente, el sistema debe evitar sobrescribir las correcciones manuales hechas por el administrador sobre los datos importados, incluso si el mismo archivo, con errores originales sin corregir, se vuelve a cargar. Para lograr esto, el sistema compara la fecha de actualización de cada dato con la fecha de la entrada correspondiente y selecciona la versión más reciente. Sin embargo, hay una excepción: si el archivo contiene una nueva entrada que refleja una interrupción prematura, como un cese o renuncia, entonces se da prioridad a la información del archivo por sobre las modificaciones realizadas por el administrador.

Asignaturas (Unidades Curriculares)

Este archivo, provisto por la Comisión de Enseñanza, contiene el listado de asignaturas correspondientes únicamente a la carrera de Ingeniería en Computación, con las columnas correspondientes al código, nombre y tipo. A partir de esta información se actualizan las tablas **unidades_curriculares** y **carreras_unidades_curriculares**. El código de la asignatura se corresponde con la columna **id_bedelias** de la tabla **unidades_curriculares** en la base de datos, y se utiliza como identificador para lograr la idempotencia. Tras identificar y actualizar o crear la asignatura correspondiente, se analiza la columna **Tipo**, que indica si la unidad curricular es ***Obligatoria*** o ***Electiva***, lo que permite actualizar la relación entre la asignatura y la carrera en la tabla **carreras_unidades_curriculares**. Cabe destacar que ***Ingeniería en Computación*** es la única carrera registrada en la base de datos, y fue cargada manualmente al inicio del proyecto.

Cabe mencionar que la tabla **unidades_curriculares** incluye también las columnas **acrónimo** y **es_de_posgrado**, que no se completan automáticamente, ya que el archivo no proporciona esa información. Sin embargo, el sistema permite al administrador completar estos datos de forma manual desde la interfaz de usuario.

En general, este archivo suele estar correctamente estructurado al momento de su carga, requiriendo únicamente ajustes menores en el formato de mayúsculas y minúsculas que son realizadas automáticamente por el sistema.

Asignaciones

Este archivo, también proporcionado por la Comisión de Enseñanza, detalla la participación de los docentes en actividades de enseñanza. Debe incluir las columnas correspondientes a la cédula del docente, la asignatura (código de la unidad curricular), el año, el semestre (***1*** o ***2***) y el rol.

Para procesar cada fila, la aplicación primero busca la unidad curricular por su código de bedelías y el docente por su cédula. Por lo tanto, para que el archivo pueda ser procesado correctamente, es necesario que los docentes y las unidades curriculares ya se encuentren cargados en el sistema a partir de los archivos previamente mencionados. En caso contrario, la fila será marcada como inválida, y el administrador recibirá un mensaje de error indicando que el docente o la unidad curricular no existen en el sistema.

Si tanto la unidad curricular como el docente existen en el sistema, se procede a buscar o generar un registro en la tabla **act_enseñanza**, la cual se encarga de vincular la unidad curricular con el semestre correspondiente.

A continuación, el sistema busca o crea un registro en la tabla **actividades_docentes**. Para este proceso, se infieren los campos **fecha_inicio** y **fecha_fin** a partir del año y semestre proporcionados, de modo que si el semestre es ***1***, la

fecha de inicio será el 1 de enero y la de fin el 30 de junio; mientras que si el semestre es ***2***, la fecha de inicio será el 1 de julio y la de fin el 31 de diciembre.

En cuanto al campo **horas_semanales**, debido a que el archivo no proporciona esta información, se asigna por defecto el valor de ***13*** horas, lo cual puede ser modificado manualmente por el administrador en caso de ser necesario.

En cuanto a los roles, el archivo no sigue una nomenclatura uniforme, utilizando términos como ***co-responsable***, ***responsable***, ***referente***, ***subresponsable*** y ***docente***. Para garantizar coherencia con el modelado del sistema, que distingue entre ***responsable***, ***subresponsable*** e ***integrante***, el sistema clasifica los primeros tres términos como ***responsable*** y asigna ***integrante*** cuando el archivo indica ***docente***.

4.9. Visualización de Datos con Mode Analytics

Aunque el sistema actual se enfoca en la recolección y gestión de datos de recursos humanos, existe potencial para expandir sus capacidades analíticas mediante la implementación futura de un *Data Warehouse* que permita generar reportes más sofisticados y análisis estratégicos. Mientras tanto, dentro del alcance del proyecto actual, se llevó a cabo una prueba de concepto que demuestra el potencial de los datos crudos recolectados y proporciona a los responsables de recursos humanos del INCO herramientas básicas de visualización y análisis, sin la necesidad de involucrar a un desarrollador para integrar estas funcionalidades en la aplicación.

La futura expansión del sistema prevé la implementación de un *Data Warehouse* para crear una capa semántica que abstraiga los detalles técnicos de la implementación actual. Esto facilitará la generación de reportes avanzados y un análisis más profundo de los datos, optimizando la toma de decisiones estratégicas dentro de la organización. Por ello, dentro del alcance del proyecto actual, se llevó a cabo una prueba de concepto que demuestra el potencial de los datos crudos recolectados y proporciona a los responsables de recursos humanos del INCO herramientas básicas de visualización y análisis, sin la necesidad de involucrar a un desarrollador para integrar estas funcionalidades en la aplicación.

Se evaluaron diversas herramientas de análisis y visualización de datos bajo los siguientes criterios: Disponibilidad de un plan gratuito, esfuerzo de integración, facilidad de uso y seguridad. Herramientas que no ofrecen un plan gratuito o que impliquen *hosting* propio, como Metabase (Metabase Inc., 2024), Holistics (Holistics Software Pte Ltd., 2024), Looker (Google, 2024c) y Tableau (Salesforce, 2024b), fueron descartadas, así como aquellas que requerían un preprocesamiento avanzado de los datos, como la creación de un *Data Warehouse* previo (por ejemplo, Power BI (Microsoft, 2024a) y Looker).

Mode Analytics (Mode Analytics Inc., 2024) fue seleccionada como la herra-

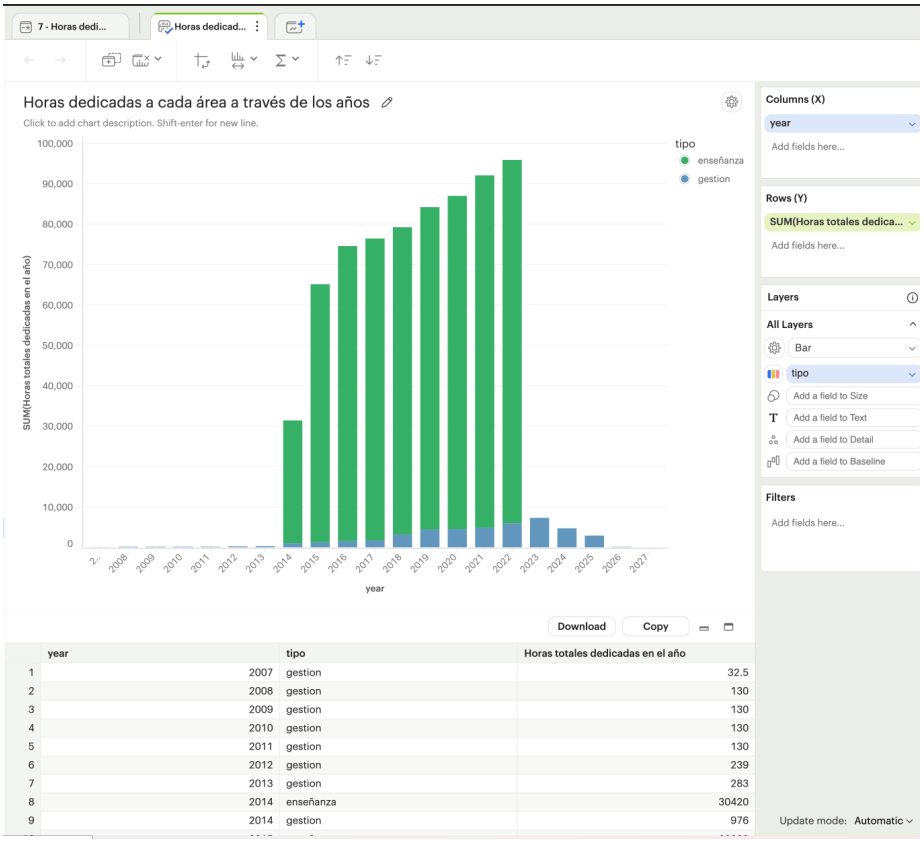
mienta más adecuada. A diferencia de muchas plataformas, Mode permite la creación de un número ilimitado de gráficos combinando datos de múltiples tablas directamente desde una base de datos relacional, sin necesidad de un preprocesamiento avanzado. Esto la convierte en una opción eficaz para obtener *insights* rápidos sobre los datos hasta que se implemente un *Data Warehouse* apropiado.

La integración de Mode con la base de datos PostgreSQL del proyecto es sencilla, ya que solo se requiere el *string* de conexión para habilitar la comunicación entre ambas plataformas. Además, al ser un servicio completamente administrado (*fully managed*), no es necesario desplegar ni configurar componentes adicionales dentro del sistema existente, lo que reduce significativamente el tiempo de configuración y permite obtener valor rápidamente en esta fase de prueba de concepto.

Mode es una opción segura y con buena reputación, utilizada por grandes empresas como Segment (Twilio Segment, 2024), Shopify (Shopify Inc., 2024) y DoorDash (DoorDash Inc., 2024). La plataforma se conecta a bases de datos utilizando TLS, logrando que la comunicación entre la aplicación y los servidores esté cifrada y protegida. Además, Mode cumple con diversas certificaciones de seguridad, con altos estándares de privacidad y protección de datos. (Mode Analytics, Inc, 2025)

Para la creación de reportes, Mode adopta un enfoque más intuitivo en comparación con otras alternativas. Utiliza SQL nativo como lenguaje de consulta para obtener los datos, lo que permite a los administradores del INCO generar sus propias consultas sin necesidad de aprender un lenguaje específico de la plataforma. Además, la creación de reportes gráficos a partir de los datos extraídos puede realizarse utilizando exclusivamente su Graphical User Interface (GUI), sin necesidad de escribir código. Esto simplifica significativamente el proceso, facilitando el uso independiente de la herramienta.

A continuación, se presentan capturas de pantalla que ilustran la Interfaz de Usuario (UI) de Mode Analytics para la generación de gráficas a partir de consultas SQL. Ver Figura 4.39.



	year	tipo	Horas totales dedicadas en el año
1	2007	gestion	32.5
2	2008	gestion	130
3	2009	gestion	130
4	2010	gestion	130
5	2011	gestion	130
6	2012	gestion	239
7	2013	gestion	283
8	2014	enseñanza	30420
9	2014	gestion	976

Figura 4.39: Gráfico obtenido para un consulta en Mode Analytics

Como parte de esta prueba de concepto, se desarrollaron aproximadamente diez gráficos diferentes que visualizan los datos recolectados por la aplicación, en línea con el requerimiento funcional RF 45 - Reportes. Estos gráficos se organizaron en un único reporte que permite visualizar los resultados en un solo lugar, con la capacidad de actualizarse en tiempo real para reflejar los datos más recientes. Ver Figura 4.40.

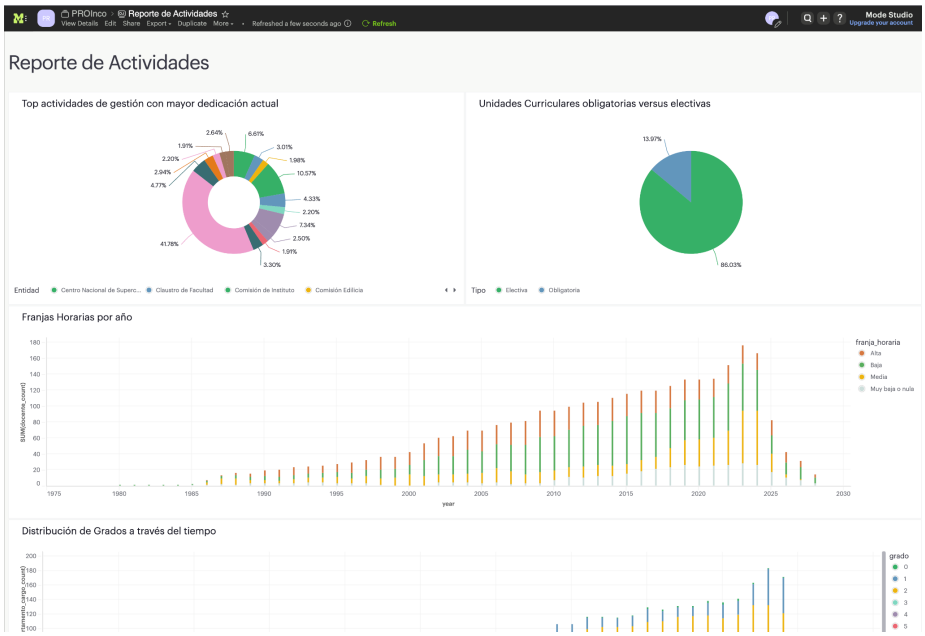


Figura 4.40: Reporte compuesto por varias consultas en Mode Analytics

Capítulo 5

Calidad de la herramienta implementada

La calidad de un sistema se define como el grado en el que este satisface las necesidades y expectativas de los diferentes involucrados. A partir de los requerimientos funcionales y no funcionales identificados durante la fase de relevamiento, se han determinado los atributos de calidad más relevantes para este proyecto. Entre estos atributos destacan la mantenibilidad, modificabilidad, usabilidad, interoperabilidad y seguridad, que guiaron la toma de decisiones.

A lo largo del desarrollo del proyecto, se implementaron diversas técnicas para garantizar la calidad del sistema, muchas de las cuales ya han sido mencionadas en secciones anteriores de este documento. En este capítulo se detallan algunas técnicas adicionales que han sido aplicadas con el fin de cumplir con estos atributos clave.

5.1. Verificación y validación

Para validar que las funcionalidades desarrolladas satisfacían las necesidades de los usuarios, se desplegaron de manera incremental a medida que se completaban. En la mayoría de los casos, comenzaron a utilizarse de inmediato. Esto permitió obtener *feedback* temprano y realizar iteraciones para mejorar funciones clave, como el listado de docentes y el formulario de carga y actualización de actividades de gestión. Todas las funcionalidades fueron validadas por Adriana Marotta, basándose en su experiencia con la plataforma y en los comentarios proporcionados por otros miembros del INCO interesados en el proyecto.

5.1.1. Pruebas automáticas

Para minimizar la introducción de fallos al incorporar nuevas funcionalidades y verificar la correctitud de las existentes, se implementaron pruebas automáticas en el *backend* y en el *frontend*. Estas pruebas incluyen principalmente pruebas unitarias, complementadas por un número más reducido de pruebas de integración centradas en los flujos más críticos del sistema, como el procesamiento de archivos CSV.

En el *backend*, se utilizó el *framework* Japa (Japa Team, 2024), recomendado oficialmente para Adonis.js, por su facilidad de uso y capacidad de integración con el ecosistema del *framework*. En el *frontend*, se empleó Jest (Meta, 2024a) junto con React Testing Library (Testing Library, 2024), una combinación que simplifica las configuraciones previas a cada prueba y facilita la simulación de interacciones del usuario en componentes React, garantizando que el comportamiento sea consistente.

Actualmente, la aplicación *backend* cuenta con un total de 359 pruebas unitarias, logrando una cobertura del 94.51 %, mientras que la aplicación *frontend* cuenta con 43 pruebas unitarias, alcanzando una cobertura del 69.02 %.

5.2. Gestión de la configuración

Para garantizar el respaldo, la trazabilidad y el control de versiones de los entregables, el código fue gestionado mediante un repositorio privado en GitHub (GitHub Inc., 2024a), mientras que los diagramas se elaboraron utilizando la herramienta LucidChart (Lucid Software Inc., 2024).

5.3. Herramientas de análisis de código

Para asegurar el cumplimiento de las buenas prácticas de desarrollo y mantener un código limpio y coherente, se configuraron herramientas de análisis de código en ambas aplicaciones, tanto en el *backend* como en el *frontend*.

Se utilizó ESLint (ESLint Team, 2024), una herramienta popular en la comunidad de desarrollo, que analiza el código y ayuda a detectar errores de sintaxis y posibles problemas en las buenas prácticas. Por otro lado, se implementó Prettier (Prettier Team, 2024), una herramienta diseñada para formatear automáticamente el código, asegurando que siga un estilo uniforme y consistente en todo el proyecto. Ambas fueron elegidas por su adopción masiva en la comunidad y su integración sencilla con proyectos de JavaScript y TypeScript.

5.4. Integración Continua

La integración continua es la práctica de fusionar frecuentemente los cambios realizados en el código base para detectar y resolver problemas de manera temprana. Durante este proyecto, todos los cambios en el código se gestionaron a través de un proceso controlado, donde cada nueva funcionalidad fue introducida mediante un *Pull Request*. Esto permitió mantener un registro claro de las modificaciones, facilitando la revisión individual y asegurando una mayor organización y trazabilidad en el flujo de trabajo.

A su vez, el repositorio está configurado con la herramienta GitHub Actions (GitHub Inc., 2024b), que ejecuta de manera automática todo el conjunto de pruebas unitarias y herramientas de análisis de calidad de código en cada *Pull Request* y cada vez que una rama se integra a la rama principal. Esto impide que se integren ramas con fallos en las pruebas o que no cumplan con las reglas de calidad predefinidas. Ver Figura 5.1.

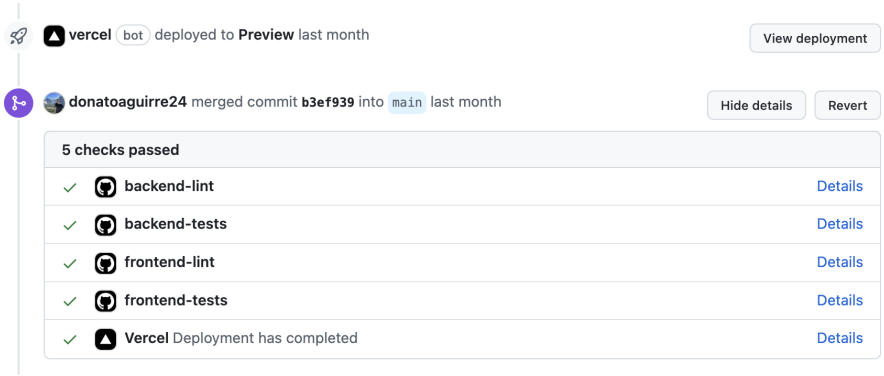


Figura 5.1: Validaciones previas a la integración de código

5.5. Manejo de errores de aplicación

Para garantizar la fiabilidad del sistema, es esencial contar con una herramienta que centralice la visualización de excepciones y notifique al equipo de desarrollo cuando se producen errores. Sin un mecanismo adecuado para la gestión de errores, los usuarios podrían encontrarse con fallos que afecten tareas críticas sin que el equipo sea consciente de ello, lo que comprometería tanto la experiencia de usuario como la estabilidad del sistema.

Por este motivo, resulta común y conveniente recurrir a herramientas de terceros especializadas en el manejo de excepciones. Estas soluciones no solo están

probadas y optimizadas, sino que también ofrecen funcionalidades avanzadas que permiten al equipo de desarrollo enfocarse en la implementación de las características específicas del sistema, en lugar de invertir recursos en el desarrollo de sistemas personalizados para la gestión de errores.

En la evaluación de plataformas de manejo de excepciones, se consideraron alternativas como Rollbar (Rollbar Inc., 2024) y Bugsnag (SmartBear Software, 2024a). Sin embargo, se optó por Sentry (Functional Software Inc., 2024) debido a su plan gratuito, que satisface plenamente las necesidades del proyecto sin incurrir en costos adicionales. Además, Sentry destaca por su integración nativa con Next.js, lo que facilita la captura de excepciones tanto en el lado del cliente como en el servidor. Esta funcionalidad es crucial en aplicaciones con renderizado híbrido, como es el caso de las desarrolladas con Next.js.

Sentry también proporciona soporte completo para aplicaciones *backend*, como la desarrollada en este proyecto en Node.js con Adonis.js, lo que permite una gestión unificada de errores tanto en el *frontend* como en el *backend*. Esto no solo simplifica el monitoreo de excepciones en un solo lugar, sino que también reduce la complejidad de depender de múltiples plataformas externas para el manejo de errores.

Una vez integrado, Sentry genera automáticamente un reporte por cada excepción no controlada que ocurra en el sistema. Estos reportes incluyen información relevante sobre el error, como el entorno en el que ocurrió, la fecha y la cantidad de veces que ha ocurrido, el *stacktrace*, la IP y otros atributos del usuario afectado, lo que facilita la investigación y resolución de problemas. Además, Sentry permite la captura manual de errores en puntos críticos del código, para identificar inconsistencias que, aunque no generen una excepción directa, requieren la atención del equipo de desarrollo.

Es importante señalar que la aplicación está diseñada para no reportar a Sentry errores derivados de acciones de los usuarios, como el ingreso de datos inválidos en formularios, dado que estos no representan defectos en el sistema que requieran la intervención de los desarrolladores.

La integración de Sentry en el sistema se realiza a través de su Software Development Kit (SDK) específico para Node.js en el *backend* y su SDK optimizado para Next.js en el *frontend*. Estas librerías, diseñadas específicamente para sus respectivos *frameworks*, no solo simplifican la configuración inicial, sino que también mejoran los datos que se reportan de manera predeterminada con cada excepción, facilitando la investigación de errores. Además, ambas librerías cuentan con amplio soporte, tanto por parte de la empresa como de la comunidad, lo que asegura su fiabilidad y mantenimiento continuo.

Internamente, cuando ocurre una excepción en el sistema, los SDKs de Sentry generan automáticamente una solicitud HTTP `POST` que se envía a la API de Sentry, con información detallada sobre el error. Una vez recibido, Sentry procesa el reporte y lo presenta en su interfaz de usuario para su análisis y seguimiento por parte del equipo de desarrollo.

Dado que el proyecto fue desarrollado bajo metodologías ágiles y se puso en producción desde etapas tempranas para su validación por parte de los usuarios, fue extremadamente útil el contar con una solución de gestión de errores eficiente desde el principio. La capacidad de Sentry para detectar y reportar excepciones desde las primeras iteraciones fue clave para garantizar que los problemas fueran identificados y resueltos rápidamente, sin afectar el flujo de validación de las funcionalidades críticas.

Sentry también ofrece la posibilidad de configurar alertas por correo electrónico, permitiendo personalizar aspectos como la frecuencia de las notificaciones, los umbrales de severidad y las reglas de alerta. Durante el período de desarrollo, Sentry se configuró para enviar notificaciones por cada nueva excepción detectada, permitiendo que si ocurriesen, se tenga una respuesta rápida y oportuna a los problemas para evitar que estos interfieran con el proceso de validación de las funcionalidades por parte de los docentes.

En la Figura 5.2 se muestra un ejemplo de cómo se visualizan los errores reportados en Sentry y en la Figura 5.3 una notificación enviadas por correo electrónico.

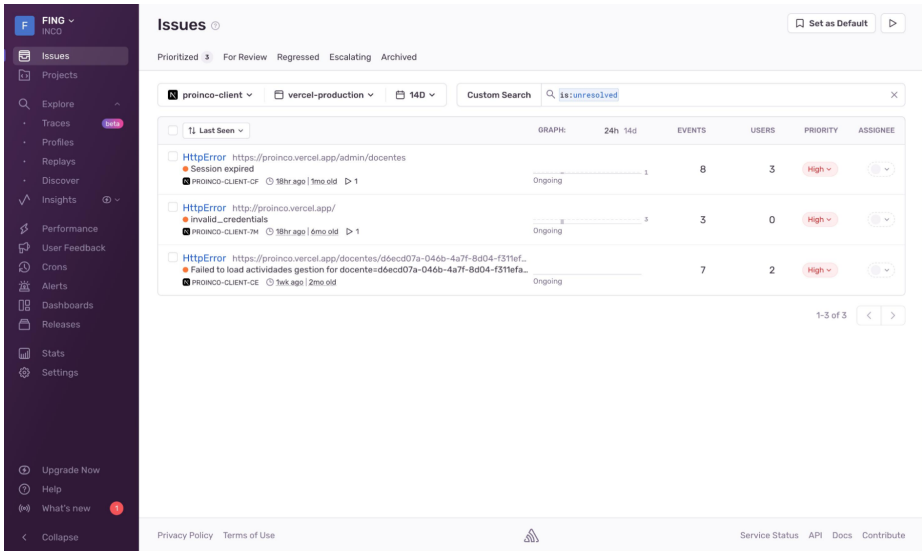


Figura 5.2: Listado de incidencias en Sentry

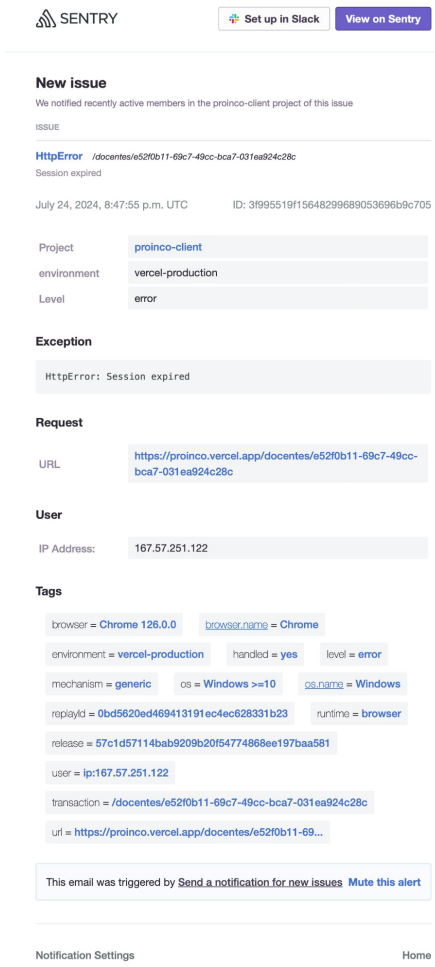


Figura 5.3: Email de alerta de nueva incidencia enviado por Sentry

5.6. Recopilación y visualización de *Logs*

Los *logs* son registros detallados que las aplicaciones generan para proporcionar información sobre su funcionamiento, incluyendo eventos, errores y advertencias. Estos registros son fundamentales para monitorear la salud del sistema y diagnosticar problemas. Por otra parte, "*Structured Logging*" es una técnica que consistente en generar los *logs* en un formato estructurado, como JSON, en lugar de texto plano. Esto facilita su procesamiento y análisis automatizado, permitiendo que luego sean filtrados, graficados y analizados mediante plataformas de monitoreo.

En el sistema desarrollado, el *backend* emite *logs* estructurados en JSON a `STDOUT`, lo que permite su procesamiento por plataformas externas. Para gestionar estos *logs*, se utiliza Papertrail (SolarWinds, 2024), una solución popular que ofrece un plan gratuito. Papertrail está configurado mediante PM2, que envía los *logs* al servidor usando User Datagram Protocol (UDP), garantizando su registro y acceso remoto. Cabe destacar que la aplicación está configurada para ofuscar datos sensibles como teléfonos y contraseñas, asegurando la privacidad de la información.

En la Figura 5.4, se presenta un ejemplo de los *logs* estructurados visualizados en Papertrail.

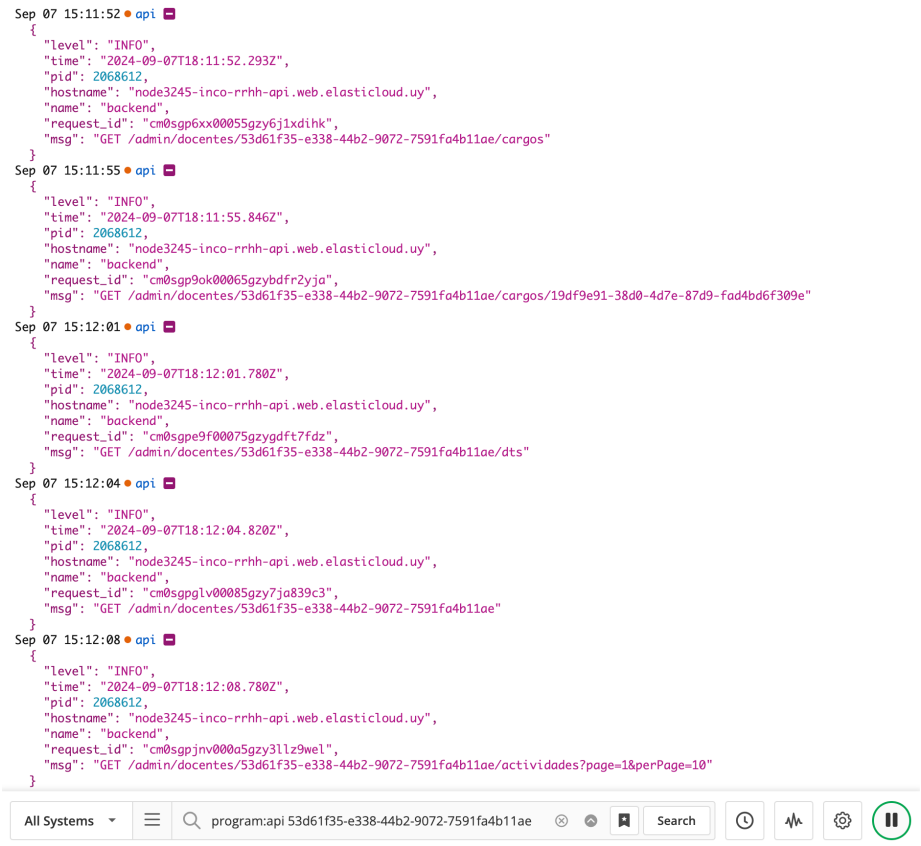


Figura 5.4: Logs en Papertrail

5.7. Consistencia en la Interfaz de Usuario

Para contribuir a que la UI del sistema sea consistente y visualmente agradable, se optó por utilizar SHADCNUI (shadcn, 2024), una librería de componentes de UI. SHADCNUI proporciona una colección de componentes pre construidos y personalizables que siguen buenas prácticas de diseño moderno. Esto no solo agilizó el desarrollo de la UI, sino que también permite mantener una apariencia coherente en todas las partes de la aplicación, mejorando la usabilidad y experiencia del usuario.

5.8. Documentación de la API

Con el fin de facilitar la comprensión y el mantenimiento del sistema por parte de futuros desarrolladores, se documentó exhaustivamente la API REST en formato OpenAPI (SmartBear Software, 2024b). En ella se describe cada *endpoint* especificando el método HTTP, parámetros, y formatos de las posibles respuestas. Ver Figura 5.5.

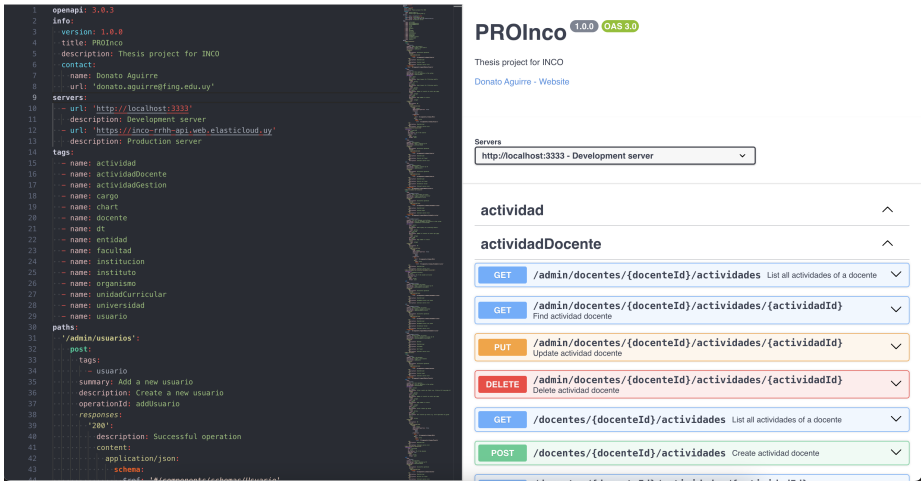


Figura 5.5: Documentación de la API

Adicionalmente, se generó una colección en *Bruno* (Bruno Team, 2024) que también está versionada en el repositorio. Esta colección permite a los desarrolladores interactuar con los diferentes *endpoints* de la API facilitando la comprensión de la API y acelerando el proceso de desarrollo y depuración. Ver Figura 5.6.

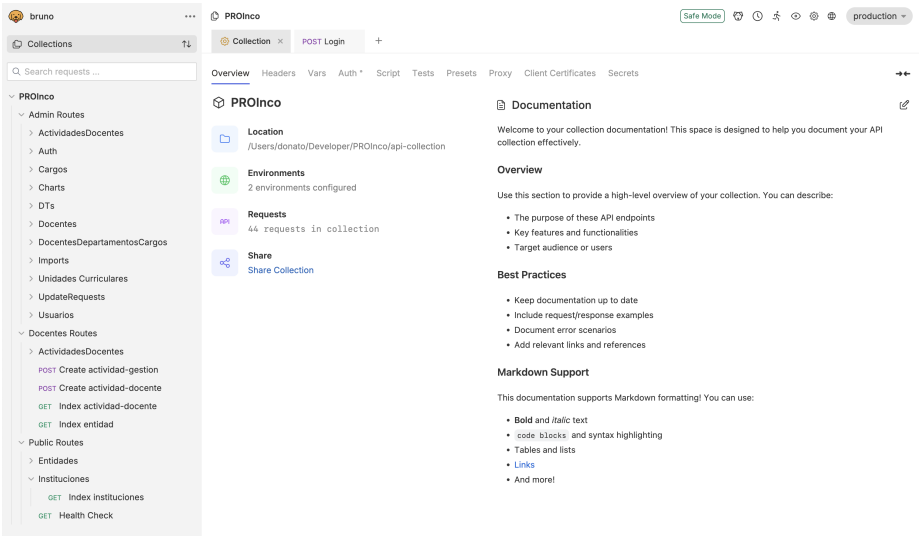


Figura 5.6: Colección de endpoints

Capítulo 6

Conclusiones y trabajo futuro

En respuesta a los problemas previamente identificados, se diseñó una solución integral que se materializa en una aplicación web destinada a los administradores, con funcionalidades adicionales para los docentes. A continuación, se enumeran las características a grandes rasgos de la solución final, alineando el índice de cada una con el problema correspondiente que resuelve.

1. **Recopilación de las actividades de gestión realizadas por los docentes.**
2. **Centralización de la información personal de los docentes**, incluyendo datos como cargo, grado, dedicación, extensiones y reducciones horarias, entre otros.
3. **Centralización de las actividades realizadas por los docentes**, consolidando toda la información en un único sistema.
4. **Validación de los datos ingresados**, mediante controles que detecten inconsistencias o datos inválidos. Este proceso alerta a quien realiza la carga, permitiéndole identificar errores y notificar el problema en la fuente.
5. **Generación de reportes básicos** que faciliten el análisis de tendencias y características del cuerpo docente.

La solución propuesta permite la recopilación centralizada de actividades de gestión docente, la consolidación de información personal detallada (cargo, grado, dedicación, etc.), la validación exhaustiva de datos para detectar inconsistencias y la generación de reportes básicos orientados al análisis del cuerpo docente. Estas funcionalidades, diseñadas específicamente para el contexto del INCO, permiten realizar una gestión más eficiente y granular que las alternativas disponibles en el mercado.

6.1. Trabajo futuro

El sistema desarrollado establece una base sólida para la gestión de recursos humanos del INCO. Sin embargo, existen diversas oportunidades de expansión y mejora que pueden implementarse en futuras iteraciones del proyecto:

1. **Construcción de un *Data Warehouse***: La implementación de un *Data Warehouse* constituye una evolución natural del sistema actual, permitiendo realizar análisis más profundos y sofisticados de los datos recolectados. Esta infraestructura facilitaría la creación de reportes avanzados, análisis de tendencias a largo plazo y la generación de métricas estratégicas para optimizar la toma de decisiones institucionales.

2. **Gestión del historial de cargas horarias**: El sistema actual mantiene únicamente la carga horaria más reciente de cada cargo docente, sobrescribiendo valores anteriores durante actualizaciones manuales o cargas de archivos CSV. La implementación de un sistema de versionado que preserve el historial completo de cargas horarias permitiría analizar la evolución temporal de las dedicaciones docentes.

3. **Implementación completa de licencias y reducciones horarias**: Aunque el modelo de datos contempla estas entidades, es necesario desarrollar la funcionalidad completa para la carga, procesamiento y visualización de datos relacionados con licencias y reducciones horarias, para contar con un seguimiento más preciso del tiempo efectivo de dedicación de cada docente.

4. **Incorporación de dedicaciones compensadas y especiales**: El sistema actual gestiona únicamente dedicaciones totales (DT). El modelo de datos ya está preparado y la información necesaria se encuentra disponible en el archivo *INCOHIST* utilizado para cargar las DT. La implementación del registro y visualización de dedicaciones compensadas y especiales completaría el panorama de las asignaciones docentes.

5. **Integración de actividades de extensión e investigación**: La incorporación de actividades de extensión e investigación al sistema constituiría el paso final para lograr una centralización completa de todas las actividades realizadas por el cuerpo docente del instituto.

Referencias

AdonisJS. (2023). *Authentication.* https://v5-docs.adonisjs.com/guides/auth/introduction.

AdonisJS. (2024a). *Adonis.js.* https://adonisjs.com. (Accessed: 2024-12-01)

AdonisJS. (2024b). *Hashing.* https://v5-docs.adonisjs.com/guides/security/hashing#usage.

AdonisJS. (2024c). *Mailer.* https://v5-docs.adonisjs.com/guides/mailer#usage.

Amazon. (2024). *Amazon web services.* https://aws.amazon.com. (Accessed: 2024-12-01)

Antel. (2024a). *Antel.* https://antel.com.uy. (Accessed: 2024-12-01)

Antel. (2024b). *Minube.* https://minubeantel.uy. (Accessed: 2024-12-01)

Asana Inc. (2024). *Asana.* https://asana.com. (Accessed: 2024-12-01)

Auth0 Inc. (2024). *Auth0.* https://auth0.com. (Accessed: 2024-12-01)

Bruno Team. (2024). *Bruno.* https://www.usebruno.com. (Accessed: 2024-12-01)

C2FO. (2024). *fast-csv.* https://c2fo.github.io/fast-csv. (Accessed: 2024-12-01)

Docker Inc. (2024). *Docker.* https://www.docker.com. (Accessed: 2024-12-01)

DoorDash Inc. (2024). *Doordash.* https://www.doordash.com. (Accessed: 2024-12-01)

Elastic N.V. (2024). *Elastic cloud.* https://www.elastic.co/cloud. (Accessed: 2024-12-01)

Elmasri, R., y Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson.

ESLint Team. (2024). *Eslint.* https://eslint.org. (Accessed: 2024-12-01)

Functional Software Inc. (2024). *Sentry.* https://sentry.io/welcome. (Accessed: 2024-12-01)

GitHub Inc. (2024a). *Github.* https://www.github.com. (Accessed: 2024-12-01)

GitHub Inc. (2024b). *Github actions.* https://github.com/features/actions. (Accessed: 2024-12-01)

Google. (2024a). *Gmail.* https://workspace.google.com/gmail. (Accessed: 2024-12-01)

Google. (2024b). *Google forms.* https://www.google.com/forms/about. (Accessed: 2024-12-01)
Google. (2024c). *Looker.* https://www.lookerstudio.google.com/overview. (Accessed: 2024-12-01)
Holistics Software Pte Ltd. (2024). *Holistics.* https://www.holistics.io. (Accessed: 2024-12-01)
IETF. (2024). *Json web tokens.* https://jwt.io. (Accessed: 2024-12-01)
INCO. (2024). *Instituto de computación: Acerca del instituto.* https://www.fing.edu.uy/es/inco/institucional. (Accessed: 2024-08-16)
Japa Team. (2024). *Japa.* https://japa.dev. (Accessed: 2024-12-01)
Keymetrics. (2024). *Pm2.* https://pm2.keymetrics.io. (Accessed: 2024-12-01)
Laravel. (2024). *Laravel.* https://laravel.com. (Accessed: 2024-12-01)
Lucid Software Inc. (2024). *Lucidchart.* https://lucidchart.com. (Accessed: 2024-12-01)
Mailgun Technologies Inc. (2024). *Mailgun.* https://mailgun.com. (Accessed: 2024-12-01)
Mailjet. (2023). *Standard configuration.* https://dev.mailjet.com/smtp-relay/configuration.
Mailjet. (2024). *Mailjet.* https://mailjet.com. (Accessed: 2024-12-01)
Meta. (2024a). *Jest.* https://jestjs.io. (Accessed: 2024-12-01)
Meta. (2024b). *React.* https://react.dev. (Accessed: 2024-12-01)
Metabase Inc. (2024). *Metabase.* https://www.metabase.com. (Accessed: 2024-12-01)
Microsoft. (2024a). *Power bi.* https://www.microsoft.com/en-us/power-platform/products/power-bi. (Accessed: 2024-12-01)
Microsoft. (2024b). *Typescript.* https://www.typescriptlang.org. (Accessed: 2024-12-01)
Mode Analytics Inc. (2024). *Mode analytics.* https://www.mode.com. (Accessed: 2024-12-01)
Mode Analytics, Inc. (2025). *Security.* https://mode.com/security.
monday.com Ltd. (2024). *Monday.* https://monday.com. (Accessed: 2024-12-01)
Mozilla Developer Network. (2024). *Javascript.* https://developer.mozilla.org/en-US/docs/Web/JavaScript. (Accessed: 2024-12-01)
Nodemailer. (2024). *Nodemailer.* https://nodemailer.com. (Accessed: 2024-12-01)
OpenJS Foundation. (2024). *Node.js.* https://nodejs.org. (Accessed: 2024-12-01)
OptimalBits. (2024). *Bull.* https://optimalbits.github.io/bull. (Accessed: 2024-12-01)
Pixel Paddock Ltd. (2024). *Float.* https://www.float.com. (Accessed: 2024-12-01)
PostgreSQL Global Development Group. (2024). *Postgresql.* https://www.postgresql.org. (Accessed: 2024-12-01)
Prettier Team. (2024). *Prettier.* https://prettier.io. (Accessed: 2024-12-01)

Rails Core Team. (2024). *Ruby on rails.* https://rubyonrails.org/. (Accessed: 2024-12-01)
Redis Ltd. (2024). *Redis.* https://redis.io. (Accessed: 2024-12-01)
Rollbar Inc. (2024). *Rollbar.* https://rollbar.com. (Accessed: 2024-12-01)
Salesforce. (2024a). *Heroku.* https://www.heroku.com. (Accessed: 2024-12-01)
Salesforce. (2024b). *Tableau.* https://www.tableau.com. (Accessed: 2024-12-01)
shadcn. (2024). *shadcn/ui.* https://ui.shadcn.com. (Accessed: 2024-12-01)
Shopify Inc. (2024). *Shopify.* https://www.shopify.com. (Accessed: 2024-12-01)
SmartBear Software. (2024a). *Bugsnag.* https://bugsnag.com. (Accessed: 2024-12-01)
SmartBear Software. (2024b). *Openapi specification.* https://swagger.io/resources/open-api/.
SolarWinds. (2024). *Papertrail.* https://www.papertrail.com. (Accessed: 2024-12-01)
Testing Library. (2024). *React testing library.* https://testing-library.com. (Accessed: 2024-12-01)
Twilio Segment. (2024). *Segment.* https://www.segment.com. (Accessed: 2024-12-01)
Twilio SendGrid. (2024). *Sendgrid.* https://sendgrid.com. (Accessed: 2024-12-01)
Vercel. (2024). *Next.js.* https://nextjs.org. (Accessed: 2024-12-01)
Virtuozzo. (2024). *Jelastic.* https://www.virtuozzo.com/jelastic-information/. (Accessed: 2024-12-01)
Yarn Contributors. (2024). *Yarn.* https://yarnpkg.com. (Accessed: 2024-12-01)

Anexo A

Modelo Entidad Relación del Sistema

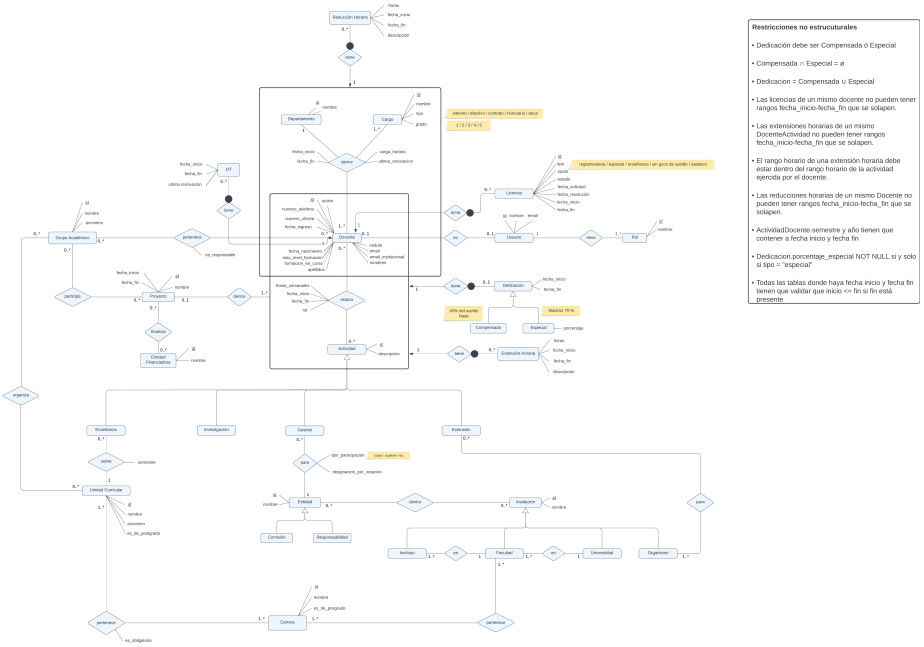


Figura A.1: Modelo Entidad Relación

Anexo B

Modelo Relacional del Sistema

En la siguiente carilla se presenta el diagrama del modelo relacional del sistema completo.

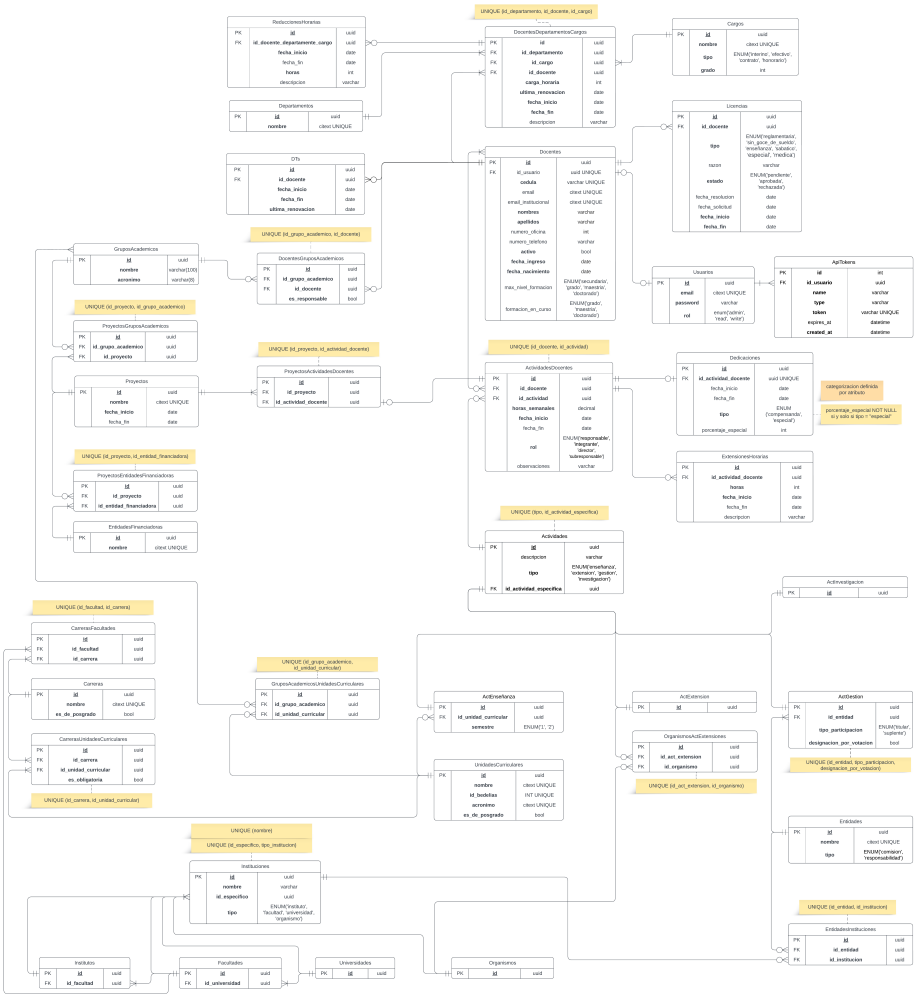


Figura B.1: Modelo Relacional

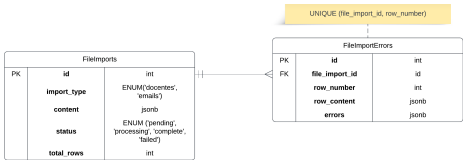


Figura B.2: Modelo Relacional - Exportación de archivos

Cabe destacar que las columnas **created_at** y **updated_at** no se incluyen

explícitamente en el diagrama del MR para evitar redundancia visual, pero se han agregado a todas las tablas en la base de datos. Estas columnas permiten registrar automáticamente la fecha y hora de creación y la última actualización de cada registro, mejorando la auditoría y trazabilidad del sistema.

Anexo C

Adherencia a los Principios de "*12 Factor App*"

"*12 Factor App*" es una metodología establecida para el desarrollo de aplicaciones modernas *cloud-native*. A través de sus doce principios, busca optimizar cada etapa del ciclo de vida del *software*, desde su desarrollo hasta la producción.

I. Código base: Este principio establece que una aplicación debe tener un único código base, versionado, con múltiples despliegues posibles, asegurando la consistencia del código y facilitando su gestión en diferentes entornos. En este proyecto, tanto el *frontend* como el *backend* se gestionan en un *monorepo* versionado en GitHub, con los despliegues de desarrollo local, *staging* (*testing*) y producción.

II. Dependencias: Las dependencias deben declararse y aislarse explícitamente para garantizar que la aplicación funcione de manera consistente en cualquier entorno, evitando problemas relacionados con dependencias implícitas o versiones conflictivas. En este proyecto, se utiliza `package.json` para gestionar las dependencias de JavaScript, junto con Yarn (Yarn Contributors, 2024), un gestor de paquetes popular que asegura la instalación reproducible de las dependencias. Además, Docker se emplea para aislar y controlar las dependencias del sistema operativo.

III. Configuraciones: La configuración debe almacenarse en el entorno, separada del código fuente, para permitir la flexibilidad y seguridad en diferentes entornos, con el fin de que ciertos aspectos de la aplicación puedan sin requerir cambios en el código y un despliegue adicional. En este proyecto, todas las configuraciones como credenciales de servicios externos, se almacenan en variables de entorno gestionadas desde la plataforma de despliegue o a través de archivos locales excluidos del control de versiones.

IV. *Backing services*: Los servicios externos que la aplicación consume deben

ser tratados como recursos conectables para facilitar su reemplazo o modificación sin afectar la lógica de la aplicación. En este proyecto, las conexiones a servicios como la base de datos se gestionan mediante variables de entorno, permitiendo cambiar entre diferentes proveedores con solo modificar la configuración correspondiente.

V. Construir, desplegar, ejecutar: El principio establece que las etapas de construcción, despliegue y ejecución deben estar claramente separadas para garantizar que el código que llega a producción sea exactamente lo que se probó en fases anteriores. En este proyecto, el proceso comienza cuando la plataforma de *hosting* detecta un nuevo *commit* en la rama principal del repositorio de GitHub e inicia la construcción del código, donde se instalan dependencias y se compilan recursos. Una vez que se construyó satisfactoriamente y sin errores, el ejecutable se despliega automáticamente para su ejecución en producción. En otras palabras, nunca se modifican directamente los archivos de código en los contenedores de Elastic Cloud. Esta separación asegura que solo el código que ha sido aprobado y probado llegue a producción, evitando sorpresas.

VI. Procesos: La aplicación debe ejecutarse como uno o más procesos sin estado para facilitar la escalabilidad y resiliencia del sistema. En este proyecto, cualquier solicitud HTTP puede ser completada por cualquier instancia del servidor, sin importar qué proceso ha atendido peticiones previas del mismo cliente, ya que el estado necesario para efectuar las transacciones se persiste en una base de datos accesible por todos los procesos.

VII. Asignación de puertos: Cada servicio de la aplicación se expone detrás de puertos configurables, facilitando la gestión y acceso dentro de un entorno de contenedores.

VIII. Concurrencia: La aplicación debe escalarse mediante el modelo de procesos para permitir la escalabilidad horizontal que busca distribuir la carga entre múltiples instancias. En este proyecto, la arquitectura del *backend* permite la escalabilidad horizontal mediante la adición de más contenedores (procesos), con la consideración de que las conexiones a la base de datos pueden convertirse en un cuello de botella.

IX. Desechabilidad: Los procesos deben ser desechables, lo que significa que pueden iniciarse y detenerse rápidamente, contribuyendo a la resiliencia del sistema. En el sistema construido, tanto la aplicación *backend* como la aplicación *frontend* pueden inicializar y finalizar en menos de cinco segundos, facilitando una rápida recuperación y escalabilidad.

X. Paridad en desarrollo y producción: El entorno de desarrollo, preproducción y producción deben mantenerse lo más similares posible para evitar errores relacionados con diferencias entre entornos, lo cual queda resuelto con la utilización de Docker.

XI. *logs*: Los *logs* que emite la aplicación deben tratarse como una transmisión de eventos para facilitar el monitoreo y proceso de debug del sistema. En este

proyecto, la aplicación emite cada *log* a la salida *stdout*. Esto permite que, herramientas como Papertrail para el análisis y visualización de *logs* puedan integrarse directamente en el entorno. Estas herramientas pueden consumir los *logs* directamente desde el *stdout* de la aplicación, sin requerir modificaciones en el código fuente, lo que asegura que la infraestructura de monitoreo y análisis se pueda actualizar o modificar sin afectar el funcionamiento de la aplicación.

XII. Administración de procesos: Las tareas administrativas, como las migraciones, deben ejecutarse como procesos puntuales y separados de la aplicación principal, asegurando que no interfieran con la ejecución de la aplicación en sí. En este proyecto, se utiliza PM2 para gestionar los procesos del *backend* y los *workers*, permitiendo que la aplicación se ejecute de manera continua y estable. De ser necesario ejecutar tareas administrativas como la apertura de una consola, se ejecutarían en procesos independientes, para no interferir con los procesos que se encuentran corriendo la aplicación.

Anexo D

Detalles de implementación de la librería Bull

A continuación se describe cómo Bull maneja el almacenamiento, recuperación y ejecución de trabajos en Redis:

Las estructuras de Redis utilizadas por la librería son las siguientes:

- **Listas (LPUSH, RPOP)**: Bull emplea listas de Redis para encolar trabajos. Los nuevos trabajos se añaden con LPUSH y se recuperan con RPOP para su procesamiento.
- **Hashes (HSET, HGET)**: Cada trabajo se representa como un hash en Redis, identificado por un JID (Job ID) único, y contiene detalles como datos, estado y progreso.
- **Conjuntos Ordenados (ZADD, ZRANGE)**: Los trabajos retrasados o programados se almacenan en conjuntos ordenados de Redis, utilizando la marca de tiempo de ejecución como puntaje.

Los trabajos se añaden a la cola con LPUSH para procesamiento inmediato o a un conjunto ordenado con ZADD si están retrasados.

Los *worker* obtienen trabajos de la lista con RPOP, y para los trabajos retrasados, utilizan ZRANGE y ZREM para recuperarlos cuando llega su tiempo de ejecución.

A continuación, se describen las propiedades relacionadas con seguridad y fiabilidad que hacen que Bull sea una opción robusta para este tipo de tarea:

- **Garantía de "al menos una vez"**: Bull garantiza que los trabajos se procesen al menos una vez, con la opción de reintento automático en caso de fallos.

- **Bloqueo de Trabajos**: Los trabajos se bloquean cuando un *worker* los toma. Si el *worker* falla, el trabajo regresa a la cola para evitar su pérdida.
- **Trabajos Estancados**: Si un *worker* no completa un trabajo a tiempo, este se considera "estancado" y se re-encola automáticamente.
- **Reintentos**: Bull tiene un mecanismo automático de reintentos para trabajos fallidos, esto es, aquellos cuya ejecución lanzó una excepción.