Aplicativo web para la administración de la herramienta de inteligencia artificial (PIA) para el equipo de innovación de la empresa Pensemos S.A

Juan Diego Roa Porras

Trabajo de Grado para Optar el Título de Ingeniería de Sistemas

Director

Duvan Yahir Sanabria Echeverry

Magíster en Ingeniería de Sistemas e Informática

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2026

Dedicatoria

Dedico este trabajo a mi familia, especialmente a mis padres, quienes a lo largo de mi vida me han brindado un apoyo incondicional y han sido un ejemplo constante, no solo a través de sus palabras, sino mediante sus acciones y su forma de afrontar la vida. A mi hermana, por haber ido siempre un paso adelante, mostrándome el camino y motivándome a ser una mejor persona y, ahora, un futuro profesional.

Asimismo, dedico este logro a mis amigos, quienes han estado presentes en cada etapa de este proceso, sin importar la situación. Puede que no sean plenamente conscientes del impacto que han tenido, pero su compañía y apoyo han sido fundamentales para la culminación de esta etapa de mi vida.

Agradecimientos

Agradezco profundamente a mi familia por su apoyo incondicional, la paciencia que me han mostrado y la motivación constante que me han brindado durante el desarrollo de este proyecto. A mis amigos, por su compañía, por su apoyo y por los momentos de distracción necesarios, así como por creer en mí y dar por sentado que lo lograría.

Agradezco también a mi director de trabajo de grado, Duván Yahir Sanabria, por su orientación y su disposición constante, así como por acompañarme en cada etapa de este proceso. De igual manera, agradezco a Pensemos S.A. por haberme acogido, por la paciencia brindada y por ofrecerme un espacio para aplicar los conocimientos adquiridos durante mi formación académica.

Finalmente, extiendo mi gratitud a todos los docentes y compañeros que, de una u otra forma, contribuyeron a mi crecimiento académico y personal a lo largo de estos años.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

Págs.

Nota. Los apéndices están disponibles en el Repositorio Institucional.

Glosario

AdminUAI: Aplicativo web desarrollado en el presente proyecto, orientado a la administración de la herramienta PIA. Permite gestionar de forma centralizada clientes, asistentes, prompts y demás configuraciones necesarias para el funcionamiento del sistema.

Asistente: Componente funcional de PIA asociado a un módulo específico de la SVE, encargado de procesar consultas y generar respuestas. Su comportamiento depende de la configuración definida y de los prompts asociados.

PIA: Herramienta de inteligencia artificial desarrollada por Pensemos S.A., que apoya a los usuarios en el uso de la SVE mediante un sistema conversacional capaz de responder preguntas específicas a partir de la información almacenada en la plataforma.

Pinecone: Servicio externo utilizado para gestionar índices vectoriales de información, permitiendo realizar búsquedas semánticas sobre los datos utilizados por los asistentes de PIA.

Prompt: Conjunto de instrucciones o contexto definido para guiar el comportamiento de un asistente. En el sistema desarrollado, los prompts determinan cómo el modelo interpreta las consultas y generan coherencia en las respuestas entregadas.

SVE (Suite Visión Empresarial): Plataforma desarrollada por Pensemos S.A. para la gestión estratégica organizacional, la cual integra diferentes módulos funcionales y sobre la cual opera la herramienta PIA.

Resumen

Título: Aplicativo web para la administración de la herramienta de inteligencia artificial (PIA) para el equipo de innovación de la empresa Pensemos S.A.[*]

Autor: Juan Diego Roa Porras[**]

Palabras Clave: PIA, Herramienta de Inteligencia Artificial, Aplicativo Web, Equipo de innovación, Pensemos S.A., Mantenimiento y configuración.

Descripción: El presente trabajo de grado tiene como objetivo desarrollar un aplicativo web para la administración de la herramienta de inteligencia artificial PIA, utilizada por el equipo de innovación de Pensemos S.A. La propuesta surge a partir de la identificación de procesos manuales en la configuración y mantenimiento del sistema, los cuales generaban errores, retrabajos y falta de trazabilidad en la gestión de clientes, asistentes y prompts.

Para abordar esta problemática, se definieron los requerimientos funcionales y no funcionales, se diseñó una arquitectura basada en microservicios y se desarrolló un prototipo funcional empleando tecnologías como FastAPI, Python, PostgreSQL y Flutter. Como resultado, la solución centraliza las operaciones administrativas, estandariza los procesos de gestión y reduce la intervención manual en tareas críticas del sistema.

El desarrollo se llevó a cabo bajo un enfoque incremental, permitiendo la construcción progresiva del sistema mediante entregas funcionales parciales y su validación continua con el equipo de innovación. Asimismo, la solución fue evaluada mediante pruebas unitarias y funcionales, evidenciando su correcto comportamiento y su alineación con los requerimientos definidos. Finalmente, el prototipo fue desplegado en el entorno de pruebas de la organización y se capacitó al equipo de innovación para su uso, facilitando su adopción y su evolución continua en el tiempo.

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Duvan Yahir Sanabria Echeverry. Magíster en Ingeniería de Sistemas e Informática.

Abstract

Title: Web Application for the Administration of the Artificial Intelligence Tool (PIA) for the Innovation Team at Pensemos S.A.[*]

Author: Juan Diego Roa Porras[**]

Key Words: PIA, Artificial Intelligence Tool, Web Application, Innovation Team, Pensemos S.A., Maintenance and Configuration.

Description: This undergraduate project addresses the development of a web application for the administration of the artificial intelligence tool PIA, used by the innovation team at Pensemos S.A. The need for this solution emerged from the presence of manual processes in system configuration and maintenance, which led to errors, rework, and limited traceability in the management of clients, assistants, and prompts.

To overcome these limitations, functional and non-functional requirements were defined, and a microservices-based architecture was designed. Based on this approach, a functional prototype was implemented using technologies such as FastAPI, Python, PostgreSQL, and Flutter, enabling the centralization and standardization of administrative operations while reducing manual intervention in critical tasks.

The development followed an incremental approach, enabling the progressive construction of the system through functional partial deliveries and continuous validation with the innovation team. The solution was further validated through unit and functional testing, confirming its correct behavior and compliance with the defined requirements. Finally, the prototype was deployed in a testing environment, and the innovation team was trained in its use, supporting its adoption and continuous evolution over time.

[*] Degree Work

[**] Faculty of Physicomechanical Engineering. School of Systems and Computer Engineering. Advisor: Duvan Yahir Sanabria Echeverry. Master in Systems Engineering and Computer Science.

Introducción

Pensemos S.A. es una empresa colombiana dedicada al desarrollo de soluciones tecnológicas orientadas a fortalecer la gestión organizacional y la planeación estratégica en las empresas. A través de sus productos y servicios, busca optimizar la toma de decisiones mediante herramientas de analítica, gestión de calidad y confiabilidad operacional, con la meta de consolidarse como un referente en el desarrollo de software para la gestión empresarial en el futuro.

Como parte de su compromiso con la innovación, la compañía ha desarrollado PIA, una herramienta de inteligencia artificial integrada a la Suite Visión Empresarial (SVE), plataforma orientada al seguimiento de indicadores, auditorías y planeación estratégica. PIA fue concebida por el equipo de innovación de Pensemos S.A. para optimizar la interacción de los usuarios con la SVE, ofreciendo asistencia contextualizada y respuestas personalizadas mediante el uso de Large Language Models (LLM).

Durante reuniones con el equipo de innovación se identificó que las tareas de administración y mantenimiento de PIA, tales como la actualización de fuentes de información y la gestión de permisos de acceso, se realizan de forma manual y sin un sistema de control definido. Esta falta de trazabilidad provoca que los errores de configuración solo se detecten cuando los clientes reportan fallos en el funcionamiento de la herramienta, lo que obliga al equipo a realizar correcciones posteriores y reduce la eficiencia en la administración general de PIA.

Frente a esta situación, el presente proyecto propuso el desarrollo de un aplicativo web para la administración de PIA, implementado bajo una arquitectura basada en microservicios y desarrollado con tecnologías modernas como FastAPI, Python y PostgreSQL. La solución permite centralizar las operaciones de gestión y facilitar su correcta ejecución mediante flujos estructurados.

Para el desarrollo del aplicativo se adoptó un enfoque incremental el cual permitió abordar de manera progresiva las diferentes etapas del sistema, facilitando la construcción y validación continua de la solución propuesta. Con ello, el proyecto buscó fortalecer la administración técnica de PIA y contribuir a la eficiencia de los procesos internos asociados con su mantenimiento y configuración.

1. Planteamiento y justificación del problema

Pensemos S.A. es una empresa dedicada al desarrollo de soluciones de software innovadoras para la gestión organizacional, con un enfoque principal en la planeación estratégica, que además integra áreas como gestión de calidad, confiabilidad operacional y analítica avanzada, entre otros. Su propósito es fortalecer la toma de decisiones y apoyar a las organizaciones en el cumplimiento de sus objetivos (Pensemos S.A., s.f.-a; Pensemos S.A., s.f.-b).

Sus servicios se encuentran integrados en la Suite Visión Empresarial (SVE), un sistema de gestión estratégica que incorpora módulos como indicadores, auditorías y Balanced Scorecard (BSC), entre otros (Pensemos S.A., s.f.-c). En los últimos años, y en respuesta al auge de los Large Language Models (LLM) en el ámbito tecnológico (Management Solutions, s.f.),

Pensemos ha desarrollado una herramienta de inteligencia artificial denominada PIA, orientada a reducir la curva de aprendizaje de la SVE y a mejorar la experiencia de usuario por medio de respuestas personalizadas (Pensemos S.A., s.f.-d).

PIA funciona a través de asistentes especializados que responden preguntas sobre los distintos módulos de la SVE (Pensemos S.A., s.f.-d). Dado que es un complemento de esta suite, los usuarios de la SVE deben poder acceder también a PIA. Sin embargo, su gestión requiere tareas de configuración y mantenimiento, como la actualización de las fuentes de información de los asistentes y la gestión de permisos de acceso a PIA por parte de los clientes.

Actualmente, estas actividades se realizan de forma manual, sin mecanismos de control ni flujos definidos, y tampoco existe trazabilidad de las modificaciones. Esto genera errores humanos y dificulta la sostenibilidad del sistema, lo que repercute directamente en la eficiencia y la confiabilidad de las operaciones.

En relación con estas tareas de mantenimiento y configuración, cuando Pensemos S.A. formaliza un contrato con un nuevo cliente, el equipo de soporte se encarga de desplegar la instancia de la SVE en el servidor de la empresa o, si el cliente lo solicita, en su propio entorno. Sin embargo, dicho equipo no cuenta con acceso ni capacitación para manipular la base de datos de PIA, por lo que el equipo de innovación debe esperar a que la instancia de la SVE esté disponible para generar manualmente los registros correspondientes en la base de datos de PIA según la licencia adquirida por el cliente.

Actualmente, la creación de nuevos clientes en PIA se realiza directamente a través de herramientas de administración de bases de datos relacionales, como pgAdmin, donde se deben

ingresar manualmente más de diez campos que, en muchos casos, podrían obtenerse automáticamente desde la SVE (por ejemplo, los módulos adquiridos, el Id de la licencia o la versión del sistema). Este proceso no solo demanda tiempo, sino que también aumenta la posibilidad de errores por digitación o inconsistencia en los datos.

De igual manera, dependiendo de los módulos contratados por cada cliente, es necesario crear los asistentes correspondientes en PIA, los cuales pueden variar entre uno y seis. Dado que esta tarea implica repetir configuraciones con ligeras modificaciones, el equipo de innovación suele recurrir a la copia manual de registros para agilizar el proceso. No obstante, cualquier descuido durante esta actividad puede generar inconsistencias, afectando el correcto funcionamiento de los asistentes y reduciendo el aprovechamiento de PIA por parte del cliente.

En este contexto, el frontend de PIA funciona como un chatbot conformado por asistentes especializados que operan de manera transparente para el usuario. El sistema determina automáticamente qué asistente debe intervenir según el contexto de la conversación; dicho asistente utiliza sus propios prompts para mantener el contexto necesario y ofrecer respuestas precisas de acuerdo con el módulo correspondiente de la SVE. Actualmente, los prompts de cada asistente deben crearse y asociarse manualmente, siendo alrededor de 20 en total para los cuatro asistentes principales, lo que incrementa la probabilidad de errores en su asignación y puede dejar a un asistente sin un prompt fundamental para su funcionamiento o vincularlo a uno incorrecto, reduciendo su rendimiento.

Asimismo, PIA incorpora el uso de Pinecone, un servicio en la nube que permite almacenar y consultar representaciones de texto embebidas, lo que le facilita realizar búsquedas semánticas sobre la información de la SVE. No obstante, es necesario crear manualmente un

índice en Pinecone para cada cliente, así como actualizar de forma manual las fuentes de información que utiliza PIA, como los manuales de la SVE, cada vez que estos se modifican.

Además, con el fin de mantener actualizada la información utilizada por los asistentes, el equipo de innovación ha incorporado el uso de Celery, una herramienta que permite sincronizar los documentos almacenados en la SVE con sus respectivas representaciones en la base de datos de PIA. Gracias a ello, los asistentes pueden acceder a información vigente y consistente. Sin embargo, la configuración de Celery también se realiza manualmente, lo que limita la personalización de los procesos por cliente y la gestión de la carga sobre el servidor.

Por otra parte, el equipo de innovación también ha identificado como desafío la gestión de los registros de funcionamiento (*logs*) de los diferentes servicios que componen PIA. En la actualidad, ante cualquier incidencia reportada, el equipo debe acceder al servidor y revisar manualmente los archivos de registro mediante comandos en Linux, lo que resulta un proceso lento y poco práctico.

Finalmente, con base en estas problemáticas, se propone el desarrollo de un aplicativo web para la administración de la herramienta de inteligencia artificial PIA, utilizando las tecnologías FastAPI**,** Python **y** PostgreSQL, con el fin de mejorar el control y la ejecución de las tareas de mantenimiento y configuración del sistema PIA, beneficiando así al equipo de innovación de la empresa Pensemos S.A.

2. Objetivos

2.1 Objetivo General

Desarrollar un aplicativo web para la administración de la herramienta de inteligencia artificial (PIA), utilizando las tecnologías FastAPI, Python y PostgreSQL, para mejorar el control y la ejecución de las tareas de mantenimiento y configuración del sistema PIA para el equipo de innovación de la empresa Pensemos S.A.

2.2 Objetivos Específicos

1. Definir los requerimientos funcionales y no funcionales del aplicativo web, mediante un análisis de la arquitectura y funcionamiento de PIA, y de las necesidades del equipo de innovación de Pensemos S.A. para la administración del sistema PIA.
2. Diseñar la arquitectura del aplicativo web conforme a los requerimientos definidos, basada en microservicios que integren el backend, el frontend y la base de datos de PIA.
3. Desarrollar el prototipo funcional del aplicativo web utilizando las tecnologías definidas, de acuerdo con la arquitectura propuesta y los requerimientos establecidos.
4. Validar la aplicación desarrollada por medio de pruebas unitarias y funcionales para asegurar su correcto funcionamiento con la herramienta PIA.
5. Capacitar al equipo de innovación de Pensemos S.A. en el uso de la aplicación web implementada.

3. Cumplimiento de los objetivos del proyecto

Con el propósito de facilitar la comprensión del documento, en esta sección se presenta una tabla que permite identificar en qué apartados del capítulo de resultados se evidencia el cumplimiento de los objetivos planteados.

Tabla 1

Cumplimiento de los objetivos del proyecto

Objetivo	Cumplimiento
Definir los requerimientos funcionales y no funcionales del aplicativo web, mediante un análisis de la arquitectura y funcionamiento de PIA, y de las necesidades del equipo de innovación de Pensemos S.A. para la administración del sistema PIA.	Ver sección 6.1 (Pág. 33-44) Especificación de requerimientos funcionales y no funcionales
Diseñar la arquitectura del aplicativo web conforme a los requerimientos definidos, basada en microservicios que integren el backend, el frontend y la base de datos de PIA.	Ver sección 6.2 (Pág. 44-67) Arquitectura de la solución
Desarrollar el prototipo funcional del aplicativo web utilizando las tecnologías definidas, de acuerdo con la arquitectura propuesta y los requerimientos establecidos.	Ver sección 6.3 (Pág. 67-94) Implementación del sistema AdminUAI
Validar la aplicación desarrollada por medio de pruebas unitarias y funcionales para asegurar su correcto funcionamiento con la herramienta PIA.	Ver sección 6.4 (Pág. 94-105) Validación de la solución
Capacitar al equipo de innovación de Pensemos S.A. en el uso de la aplicación web implementada.	Ver sección 6.5 (Pág. 105-109) Capacitación del equipo de innovación en el uso de AdminUAI

4. Marco de referencia

4.1 Fundamentos teóricos

4.1.1 Requerimientos funcionales y no funcionales

En el desarrollo de software, los requerimientos funcionales son aquellas especificaciones detalladas en las que se definen las acciones, los comportamientos y las funcionalidades del software. Básicamente, describen lo que el sistema debe hacer incluyendo las tareas a realizar, como deberá interactuar con los usuarios y como deberá responder a diversas entradas o eventos específicos.

Por otro lado, los requerimientos no funcionales describen las características que definen cómo debe comportarse el sistema, más allá de sus funciones. Incluyen aspectos como el rendimiento, la seguridad, la usabilidad y la escalabilidad, así como restricciones técnicas o normativas que puedan afectar su desarrollo. Aunque no son siempre visibles para el usuario, son esenciales para garantizar que el software funcione de manera eficiente, confiable y conforme a estándares de calidad (Visure Solutions, s.f.).

4.1.2 Arquitectura de microservicios

La arquitectura de microservicios se basa en dividir una aplicación en servicios pequeños, independientes y bien definidos, donde cada uno cumple una función específica dentro de un contexto de negocio. Estos servicios son desarrollados y gestionados por equipos reducidos, lo que facilita la colaboración y acelera los ciclos de desarrollo. El valor agregado de esta arquitectura radica en que cada microservicio puede implementarse, actualizarse o reemplazarse de manera autónoma, sin afectar significativamente al resto del sistema. La comunicación entre ellos se establece mediante APIs bien estructuradas, lo que permite mantener sus

implementaciones internas aisladas. Además, esta arquitectura ofrece flexibilidad tecnológica, ya que cada servicio puede emplear el lenguaje y el entorno que mejor se adapten a sus necesidades.

Entre sus principales ventajas se destacan la escalabilidad independiente de los servicios, el aislamiento de errores, la facilidad para realizar cambios y actualizaciones, y una mayor flexibilidad tecnológica (Microsoft, s.f.).

4.1.3 Bases de datos relacionales

Una base de datos relacional es un sistema que organiza la información en tablas compuestas por filas y columnas, donde cada fila representa un registro único identificado por una clave y cada columna define un atributo de los datos. Esta estructura tabular permite establecer relaciones entre distintos conjuntos de información, facilitando el acceso, la gestión y la consulta eficiente de los datos.

Este tipo de base de datos se fundamenta en el modelo relacional, desarrollado como una solución al problema de las estructuras de datos dispares en los primeros sistemas. Dicho modelo ofrece una forma estándar, estructurada y flexible de representar y consultar información, garantizando la integridad y consistencia de los datos mediante reglas que evitan errores como la duplicación de registros o la pérdida de coherencia entre tablas.

Además, el lenguaje SQL (Structured Query Language) complementa este modelo al proporcionar un medio uniforme y preciso para realizar consultas y operaciones sobre los datos. Gracias a esta arquitectura y a sus mecanismos de integridad, las bases de datos relacionales aseguran la coherencia y confiabilidad de la información, incluso cuando múltiples usuarios o aplicaciones acceden a ella de manera simultánea (Oracle, 2021).

4.1.4 API REST

REST (Representational State Transfer) es un estilo arquitectónico para el diseño de sistemas distribuidos basado en la transferencia de representaciones de recursos a través de la web. Propuesto por Roy Fielding en el año 2000, se ha convertido en una de las metodologías más utilizadas para la creación de interfaces de programación de aplicaciones (APIs) en entornos web. Más que un protocolo o estándar, REST define un conjunto de principios que orientan la comunicación entre clientes y servidores mediante recursos identificables y operaciones uniformes.

Este estilo arquitectónico se sustenta en seis principios fundamentales: una interfaz uniforme, que estandariza la forma de acceder y manipular los recursos; una arquitectura cliente-servidor, que separa la lógica de presentación de la gestión de datos; la ausencia de estado (stateless), que exige que cada solicitud contenga toda la información necesaria sin depender de solicitudes previas; la capacidad de caché, que permite reutilizar respuestas para optimizar el rendimiento; un sistema en capas, que organiza los componentes de forma jerárquica; y la ejecución de código bajo demanda (opcional), que posibilita que el servidor amplíe las funciones del cliente mediante código ejecutable.

En este contexto, una API REST (o API RESTful) es una interfaz de programación que implementa los principios de REST para facilitar la comunicación entre un cliente y un servidor. Utiliza recursos identificados por URIs (Uniform Resource Identifiers) y los gestiona mediante métodos estándar del protocolo HTTP, como GET, POST, PUT y DELETE (Gupta, 2025).

4.1.5 Pruebas de software

Las pruebas de software son el proceso mediante el cual se evalúa y verifica que un producto o aplicación funcione correctamente, de forma segura y eficiente, conforme a los requisitos establecidos. Su propósito principal es identificar errores, optimizar el rendimiento y garantizar la entrega de un software de alta calidad. Estas pruebas pueden realizarse de manera manual o automatizada, utilizando scripts, herramientas web o incluso técnicas basadas en inteligencia artificial que permiten agilizar y mejorar la precisión del proceso de validación (Susnjara & Smalley, 2025).

4.1.6 Tipos de pruebas de software

Pruebas unitarias: Consiste en evaluar de forma aislada los componentes o funciones individuales del código para verificar que cada parte funcione según lo esperado. Generalmente se realizan durante la etapa de desarrollo y constituyen la primera capa dentro del proceso de pruebas automatizadas (BrowserStack, 2025a).

Pruebas de regresión: Se realizan cuando se introduce un cambio en el software existente, ya sea la incorporación de una nueva funcionalidad o la modificación de una ya existente. Este tipo de prueba vuelve a ejecutar el código completo para garantizar que las actualizaciones no alteren el funcionamiento original del sistema (BrowserStack, 2025a).

Pruebas funcionales: Busca verificar que cada función de la aplicación se comporta de acuerdo con los requisitos específicos y cumpla con las diferentes expectativas bajo diferentes condiciones. Su objetivo principal es asegurar que el sistema responda correctamente a las acciones del usuario y satisfaga las necesidades del negocio (BrowserStack, 2025b).

Cabe destacar que, además de las pruebas mencionadas, existen otros tipos de pruebas de software como las pruebas de integración, de aceptación, de rendimiento, entre otras, que se seleccionan según los objetivos y el contexto del desarrollo (Pittet, s.f.).

4.2 Marco tecnológico

4.2.1 Python

Python es un lenguaje de programación de alto nivel, orientado a objetos, fácil de interpretar y con una sintaxis clara y legible. Se caracteriza por ser multipropósito, de código abierto y por su portabilidad entre distintos sistemas operativos (como Windows, macOS y Linux), además de ofrecer una excelente integración con otros lenguajes como C o Java. Entre sus principales ventajas se destacan su amplia biblioteca estándar, la posibilidad de escribir código de manera más concisa que en muchos otros lenguajes y una comunidad activa que proporciona soporte, recursos educativos y soluciones.

Con Python es posible desarrollar una gran variedad de aplicaciones, desde sitios web (mediante frameworks como Django o FastAPI) y scripts para automatizar tareas, hasta software de escritorio y prototipos rápidos. Asimismo, es ampliamente utilizado en ciencia de datos y aprendizaje automático para la limpieza, análisis y visualización de información, así como para el entrenamiento de modelos predictivos, gracias a su extenso ecosistema de bibliotecas especializadas (Amazon Web Services, s.f.-a).

4.2.2 FastAPI

FastAPI es un framework moderno y de alto rendimiento para construir APIs web con Python, diseñado para aprovechar las anotaciones de tipo del lenguaje (type hints) introducidas a partir de Python 3. Estas anotaciones permiten validar datos, generar documentación automática

y detectar errores de forma anticipada, mejorando la productividad del desarrollador. FastAPI se basa en librerías robustas como Starlette (para el manejo asíncrono) y Pydantic (para la validación y el modelado de datos), lo que le otorga una combinación destacada de velocidad, simplicidad y fiabilidad en producción.

Entre sus características más importantes se encuentran su alto rendimiento, comparable al de frameworks como Node.js o Go; su capacidad para reducir la cantidad de código repetitivo; su integración con herramientas de validación y documentación automática; y su soporte completo para desarrollo asíncrono. Además, FastAPI se integra de manera nativa con OpenAPI, generando documentación interactiva (a través de Swagger UI o ReDoc) basada directamente en los tipos y modelos declarados, lo que facilita que otros desarrolladores exploren, prueben y consuman la API (FastAPI, s.f.).

4.2.3 Pytest

Pytest es un framework de pruebas para Python que simplifica la creación, organización y ejecución de tests mediante funciones sencillas y el uso directo de aserciones, sin requerir configuraciones complejas o estructuras de clases. Este enfoque lo hace más accesible que otros frameworks como unittest, al reducir la cantidad de código repetitivo necesario (boilerplate) y ofrecer una salida de resultados más clara y detallada, lo que facilita la detección y corrección de errores durante el desarrollo.

Entre sus principales ventajas se destacan su facilidad de aprendizaje, la legibilidad del output generado, la reducción del código estándar obligatorio, y el soporte para la parametrización de pruebas, que permite ejecutar un mismo test con diferentes conjuntos de datos. Además, Pytest cuenta con una arquitectura extensible basada en complementos (plugins),

lo que lo convierte en una herramienta versátil y escalable para proyectos de cualquier tamaño (Hillard, 2024).

4.2.4 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional (RDBMS) de código abierto que se ha mantenido competitivo durante décadas gracias a su fiabilidad, flexibilidad y apego a los estándares abiertos. Se distingue por su capacidad para manejar tanto datos relacionales como no relacionales, lo que lo convierte en una opción versátil para aplicaciones empresariales de diversa escala.

Entre sus características más destacadas se encuentran su diseño extensible, el sólido respaldo de una comunidad activa que impulsa su evolución continua, la compatibilidad con múltiples lenguajes de programación y su alta capacidad de escalabilidad, lo que le permite adaptarse desde proyectos pequeños hasta sistemas de gran complejidad. Asimismo, ofrece soporte para replicación, transacciones seguras y alta disponibilidad, consolidándose como una de las soluciones más robustas y confiables en el ámbito de las bases de datos. (IBM, s.f.)

4.2.5 Nginx

NGINX es un servidor web de código abierto que también puede funcionar como proxy inverso, balanceador de carga y sistema de caché. Se utiliza para distribuir y gestionar el tráfico entre servidores, reduciendo los tiempos de carga, mejorando el rendimiento y garantizando la disponibilidad del servicio. Gracias a su capacidad para atender múltiples conexiones al mismo tiempo, ofrece una experiencia de navegación rápida, estable y escalable incluso en entornos con alto volumen de usuarios.

Su arquitectura se basa en un modelo asíncrono y orientado a eventos, lo que le permite manejar miles de conexiones simultáneamente con pocos recursos. Utiliza un proceso principal (maestro) que controla varios procesos secundarios (trabajadores), encargados de atender las solicitudes de los usuarios de forma eficiente y sin interrupciones. Este diseño maximiza el rendimiento, reduce la carga del sistema y permite realizar actualizaciones sin detener el servicio (Papertrail, s.f.).

4.2.6 JSON Web Token (JWT)

El JSON Web Token (JWT) es un estándar abierto utilizado para transmitir información de manera segura entre dos partes mediante objetos JSON firmados digitalmente. Su principal uso se da en procesos de autenticación y autorización, ya que permite que un servidor valide la identidad de un usuario sin necesidad de mantener sesiones activas. Una vez el usuario se autentica, el servidor genera un token firmado que el cliente envía en cada solicitud posterior para acceder a recursos protegidos, garantizando así una comunicación confiable y eficiente.

El uso de JWT como medida de seguridad se debe a su capacidad para asegurar la integridad y autenticidad de los datos transmitidos. Gracias a su firma digital, cualquier intento de modificación del token puede ser detectado, lo que previene accesos no autorizados. Además, su estructura ligera y formato estándar facilitan la integración con aplicaciones web modernas basadas en arquitecturas distribuidas, como las API REST, permitiendo un control de acceso más escalable y seguro (JWT.io, s.f.).

4.2.7 Flutter

Flutter es un framework de interfaz de usuario (UI) de código abierto desarrollado por Google que permite crear aplicaciones multiplataforma utilizando el lenguaje de programación

Dart. Su principal ventaja radica en la posibilidad de escribir un solo código base que puede ejecutarse en Android, iOS, web y escritorio, lo que optimiza el proceso de desarrollo y reduce costos. Además, ofrece un rendimiento optimizado, ya que el código se compila en formato nativo (ARM, Intel o JavaScript), y proporciona un conjunto robusto de herramientas para pruebas, depuración y automatización. Flutter también destaca por su amplia biblioteca de widgets personalizables, que facilita la creación de interfaces visualmente atractivas y coherentes, así como por su flexibilidad de integración con aplicaciones y servicios ya existentes.

En cuanto a su arquitectura, Flutter se compone principalmente de dos capas esenciales: el framework y el motor. El framework, escrito en Dart, gestiona las funciones orientadas al usuario, incluyendo la representación visual, la interacción, la gestión del estado y las animaciones. Por su parte, el motor, desarrollado en C++, se encarga de renderizar los elementos gráficos mediante la biblioteca Skia, además de manejar operaciones de bajo nivel, como transiciones, animaciones y la comunicación con el código nativo de cada plataforma (BrowserStack, 2025c).

5. Metodología

De acuerdo con los modelos de la ingeniería de software, se adoptó un modelo de proceso incremental como metodología general para el desarrollo del proyecto, el cual permitió construir el sistema mediante entregas sucesivas que incorporaban funcionalidades de manera progresiva (Pressman, 2010). Este enfoque facilitó la validación continua de los resultados y la incorporación de ajustes a lo largo del proceso, permitiendo abordar de forma integral las etapas de análisis, diseño, implementación y validación de la solución.

5.1 Fases y actividades

El desarrollo del prototipo se estructuró en incrementos funcionales organizados de manera progresiva, de acuerdo con el modelo de proceso incremental. Cada incremento contempló la planificación y análisis, diseño, implementación y validación de un conjunto de funcionalidades, permitiendo construir el sistema de forma evolutiva. Este enfoque facilitó la incorporación de mejoras continuas y el ajuste de requerimientos a lo largo del proyecto, manteniendo la coherencia con las necesidades del equipo de innovación de Pensemos S.A.

El seguimiento del proyecto se realizó mediante reuniones diarias con el tutor asignado por Pensemos S.A., en las que se revisaron los avances, se identificaron posibles bloqueos y se definieron las tareas prioritarias de cada jornada. Además, se llevó a cabo una reunión semanal con el director del trabajo de grado, destinada a la presentación de los progresos generales y a la verificación del cumplimiento de los objetivos académicos. Al finalizar cada incremento, se realizaron las reuniones de planificación, revisión y retrospectiva con el equipo de innovación de Pensemos S.A., donde se evaluó el progreso alcanzado y se establecieron las prioridades para el siguiente ciclo de trabajo.

Fase 1: Planificación y análisis

- Se realizó la revisión de la documentación técnica y del modelo entidad–relación de PIA, con el propósito de identificar los componentes más relevantes y comprender el funcionamiento general del sistema.
- Se llevaron a cabo reuniones con el equipo de innovación de Pensemos S.A. para analizar la problemática, definir los requerimientos funcionales y no funcionales del aplicativo, y delimitar el alcance del proyecto.
- Se diseñó el plan de trabajo, estableciendo las fases, actividades y entregables que guiaron el desarrollo del proyecto.

Fase 2: Diseño del aplicativo web

- Se definió la arquitectura del aplicativo web, aplicando un enfoque basado en microservicios y utilizando las tecnologías establecidas en los objetivos del proyecto.
- Se definieron las historias de usuario que orientaron el desarrollo del sistema, conforme a los requerimientos previamente establecidos.

Fase 3: Implementación y validación

- Se desarrolló el backend y la API del aplicativo web utilizando FastAPI, garantizando la integración con la base de datos y los servicios de PIA.
- Se implementó el frontend de la aplicación, conectándolo con el backend y consumiendo los servicios expuestos por la API.
- Se incorporó un sistema de autenticación y seguridad mediante JSON Web Token (JWT) para la gestión de usuarios y el control de accesos.
- Se validó el funcionamiento del sistema a través de pruebas unitarias y funcionales, verificando el correcto desempeño de los módulos, las operaciones CRUD y la coherencia de los datos con respecto a la base de datos de PIA.
- Finalmente, se desplegó la aplicación en el servidor de pruebas de Pensemos S.A., donde se realizaron los ajustes necesarios a partir de la retroalimentación del equipo de innovación.

Fase 4: Documentación y capacitación

- Se elaboró la documentación técnica donde se describe la arquitectura general del sistema, las tecnologías empleadas y las principales decisiones de diseño tomadas durante el desarrollo.
- Finalmente, se capacitó al equipo de innovación de Pensemos S.A. en el uso y operación del sistema desarrollado.

6. Resultados

6.1 Especificación de requerimientos funcionales y no funcionales

Como resultado del proceso de análisis y levantamiento de información descrito en la metodología, se obtuvo la especificación de los requerimientos funcionales y no funcionales que orientaron el diseño e implementación del sistema propuesto.

Los requerimientos funcionales describen las capacidades y comportamientos que el sistema debe ofrecer para cumplir con los objetivos planteados, definiendo las operaciones que podrán realizar los usuarios y las interacciones entre los diferentes componentes del sistema. Estos requerimientos fueron identificados a partir del análisis del contexto operativo de la herramienta PIA, las necesidades del equipo de innovación de Pensemos S.A. y las funcionalidades requeridas para la administración de esta y sus servicios asociados.

Por su parte, los requerimientos no funcionales establecen las características de calidad, restricciones tecnológicas y criterios técnicos que deben cumplirse durante el desarrollo del sistema, con el fin de garantizar aspectos como mantenibilidad, trazabilidad, compatibilidad tecnológica y adaptabilidad de la interfaz.

Con el propósito de mantener la claridad y organización del documento, los requerimientos funcionales se presentan agrupados por módulo del sistema. Cada requerimiento se identifica mediante un código único que permite su trazabilidad dentro del proyecto.

6.1.1 Requerimientos funcionales

Los requerimientos funcionales fueron organizados por entidades del sistema con el fin de facilitar su comprensión y mantener una estructura coherente con la arquitectura del sistema desarrollado.

Cada requerimiento se identifica mediante un código con el prefijo RF (Requerimiento Funcional), seguido de la abreviatura de la entidad correspondiente y un número consecutivo. Las entidades definidas para la agrupación de los requerimientos funcionales son las siguientes:

- **SEC:** Seguridad y autenticación.
- **CLI:** Gestión de clientes.
- **AST:** Gestión de asistentes.
- **PRO:** Gestión de prompts.
- **SVE:** Versiones y manuales de la SVE.
- **GRP:** Gestión de grupos.
- **UST:** Gestión de tipos de usuario.
- **USR:** Gestión de usuarios.
- **LOG:** Visor de logs del ecosistema de PIA.

A continuación, se presentan los requerimientos funcionales identificados para cada una de estas entidades.

Tabla 2

Requerimientos funcionales de seguridad (SEC)

Identificador	Nombre	Descripción
RF-SEC-01	Iniciar sesión mediante redirección desde PIA	El sistema gestionará el acceso mediante la redirección desde sistemas externos autorizados, validando permisos a través de un token JWT recibido sin requerir un formulario propio.
RF-SEC-02	Validar continuamente la sesión mediante token JWT	El sistema deberá validar de forma continua la sesión del usuario mediante un token JWT almacenado en el *session storage*, verificando su validez en cada navegación.
RF-SEC-03	Cerrar sesión de manera local	El sistema permitirá al usuario finalizar su sesión localmente eliminando el token JWT del almacenamiento, redirigiéndolo al flujo externo sin afectar sesiones en otros sistemas integrados

Tabla 3

Requerimientos funcionales de gestión de clientes (CLI)

Identificador	Nombre	Descripción
RF-CLI-01	Listar clientes	El sistema presentará un listado paginado de clientes registrados, permitiendo la búsqueda por nombre y el acceso a las opciones de consulta detallada o actualización de información.
RF-CLI-02	Crear cliente de la SVE	El sistema permitirá el registro de nuevos clientes, automatizando la creación de sus respectivos asistentes y prompts por defecto según los módulos de la SVE seleccionados.
RF-CLI-03	Probar conexión con la SVE	El sistema validará automáticamente la conexión con la instancia de la SVE del cliente mediante su URL e ID de licencia, proporcionando

		retroalimentación visual inmediata.
RF-CLI-04	Obtener información de la licencia de la SVE	El sistema recuperará automáticamente los datos de la licencia tras una conexión exitosa, autocompletando campos como versión, límite de usuarios y módulos instalados en el formulario.
RF-CLI-05	Crear automáticamente el índice de Pinecone	El sistema generará o reutilizará un índice en la plataforma Pinecone mediante su API, asociándolo al cliente durante el registro para su uso operativo posterior.
RF-CLI-06	Crear automáticamente los asistentes de un cliente	El sistema registrará de forma automatizada un asistente por cada módulo instalado, vinculándolos al índice de Pinecone correspondiente y aplicando la configuración base definida.
RF-CLI-07	Crear automáticamente los prompts de los asistentes	El sistema descargará y registrará los prompts más recientes desde los repositorios de GitLab para cada asistente, garantizando que cuenten con la configuración funcional requerida.
RF-CLI-08	Consultar detalle del cliente	El sistema permitirá visualizar la información completa y configuración técnica de un cliente específico en modo de solo lectura, facilitando la copia segura de sus datos.
RF-CLI-09	Actualizar cliente	El sistema permitirá modificar los datos editables de un cliente existente y habilitará la adición de nuevos asistentes, manteniendo la integridad de la configuración estructural previa.
RF-CLI-10	Sincronizar prompts de un cliente especifico	El sistema actualizará masivamente los prompts de todos los asistentes asociados a un cliente, descargando las versiones vigentes desde GitLab para asegurar la consistencia funcional.

Tabla 4

Requerimientos funcionales de gestión de asistentes (AST)

Identificador	Nombre	Descripción
RF-AST-01	Listar asistentes	El sistema mostrará un listado paginado de asistentes, permitiendo filtrar por nombre o cliente y facilitando el acceso a las acciones de gestión y consulta detallada.
RF-AST-02	Crear asistentes para un cliente existente	El sistema permitirá registrar nuevos asistentes para módulos adquiridos posteriormente, integrándolos con el índice existente y generando sus respectivos prompts de forma totalmente automática.
RF-AST-03	Consultar detalle del asistente	El sistema presentará toda la información técnica y de configuración de un asistente seleccionado, permitiendo la visualización detallada en una vista de solo lectura.
RF-AST-04	Actualizar asistente	El sistema permitirá modificar campos específicos de la configuración, como el modelo o icono, restringiendo cambios en atributos estructurales para preservar la estabilidad operativa.
RF-AST-05	Sincronizar prompts de un asistente especifico	El sistema sincronizará los prompts de un asistente puntual con sus archivos en GitLab, actualizando los registros asociados para trabajar con la configuración funcional más reciente.

Tabla 5

Requerimientos funcionales de gestión de prompts (PRO)

Identificador	Nombre	Descripción
RF-PRO-01	Listar prompts	El sistema proporcionará una vista paginada de los prompts, agrupados por cliente y asistente, incluyendo un previsualizador de contenido para facilitar su identificación y gestión.
RF-PRO-02	Crear prompts	El sistema permitirá el registro manual de prompts individuales asociados a un asistente, facilitando la realización de pruebas de rendimiento y mejoras puntuales de comportamiento.
RF-PRO-03	Consultar detalle del prompt	El sistema mostrará la configuración y el contenido completo de un prompt seleccionado en una vista dedicada de solo lectura, permitiendo validar su configuración sin alteraciones.
RF-PRO-04	Actualizar prompts	El sistema permitirá editar el contenido de un prompt existente para ajustar el comportamiento del asistente.
RF-PRO-05	Eliminar prompts	El sistema permitirá remover registros de prompts que ya no sean requeridos, solicitando confirmación previa para evitar la pérdida accidental de configuraciones operativas críticas.

Tabla 6

Requerimientos funcionales de gestión de versiones de la SVE

Identificador	Nombre	Descripción
RF-SVE-01	Listar versiones de la SVE	El sistema presentará un catálogo paginado de versiones de la SVE registradas, permitiendo realizar búsquedas y acceder a la gestión detallada de cada una de ellas.
RF-SVE-02	Crear versiones de la SVE	El sistema permitirá registrar nuevas versiones de la plataforma SVE, vinculándolas a versiones anteriores y posibilitando la carga opcional de manuales en formato PDF.
RF-SVE-03	Consultar detalle de la versión de la SVE	El sistema mostrará la información técnica de una versión específica, permitiendo visualizar sus manuales asociados y copiar datos en un entorno de solo lectura.
RF-SVE-04	Actualizar versiones de la SVE	El sistema permitirá modificar los datos y manuales de una versión registrada, asegurando la vigencia de la información disponible para los procesos de administración y soporte.

Tabla 7

Requerimientos funcionales de gestión de grupos (GRP)

Identificador	Nombre	Descripción
RF-GRP-01	Listar grupos	El sistema listará de forma paginada los grupos creados, facilitando la búsqueda por nombre y el acceso a las funciones de edición, consulta o eliminación.

RF-GRP-02	Crear grupos	El sistema permitirá registrar nuevos grupos para organizar permisos, validando que el nombre no sea duplicado ni corresponda a valores reservados para la administración.
RF-GRP-03	Consultar detalle del grupo	El sistema proporcionará una vista completa de la información de un grupo seleccionado, presentando los datos en modo de solo lectura para la verificación de su configuración.
RF-GRP-04	Actualizar grupos	El sistema permitirá modificar el nombre de los grupos existentes, exceptuando los reservados, para mantener actualizada la estructura organizativa de los permisos funcionales.
RF-GRP-05	Eliminar grupos	El sistema permitirá la eliminación de grupos sin registros asociados, implementando validaciones de integridad referencial para evitar inconsistencias en la gestión de accesos del aplicativo.

Tabla 8

Requerimientos funcionales de gestión de tipos de usuario (UST)

Identificador	Nombre	Descripción
RF-UST-01	Listar tipos de usuarios	El sistema mostrará un listado paginado de perfiles definidos, permitiendo búsquedas por nombre y el acceso a las vistas de detalle y actualización de parámetros.
RF-UST-02	Crear tipos de usuario	El sistema permitirá registrar nuevos tipos de usuario definiendo parámetros como el modelo LLM, límites de consultas diarias y acceso a funcionalidades específicas de PIA.

RF-UST-03	Consultar detalle del tipo de usuario	El sistema presentará la configuración detallada de un tipo de usuario seleccionado, permitiendo la revisión de sus límites y permisos en una interfaz de solo lectura.
RF-UST-04	Actualizar tipos de usuario	El sistema permitirá ajustar los parámetros operativos de un tipo de usuario existente para adaptar el comportamiento de PIA según las necesidades del modelo de negocio.
RF-UST-05	Eliminar tipos de usuario	El sistema permitirá remover tipos de usuario que no tengan registros activos vinculados, garantizando una gestión organizada de los perfiles disponibles para asignación.

Tabla 9

Requerimientos funcionales de gestión de usuarios (USR)

Identificador	Nombre	Descripción
RF-USR-01	Listar usuarios	El sistema proporcionará un listado paginado de los usuarios de PIA, permitiendo filtrar por nombre o cliente asociado para facilitar su administración técnica e individual.
RF-USR-02	Consultar detalle del usuario	El sistema mostrará la configuración del usuario seleccionado, permitiendo validar su estado en modo de solo lectura.
RF-USR-03	Consultar perfil del usuario con la sesión iniciada en el sistema	El sistema permitirá al administrador visualizar su propia información y permisos vigentes, extraídos del token JWT activo, sin habilitar la edición de los datos.
RF-USR-04	Gestionar los usuarios con acceso al sistema	El sistema permitirá habilitar o revocar el acceso al panel de administración para múltiples usuarios de un cliente específico,

		proporcionando retroalimentación visual sobre los cambios que se van a aplicar.
RF-USR-05	Gestionar asignación de tipos de usuario	El sistema permitirá asignar tipos de usuario masivamente, controlando que no se superen los límites de usuarios premium configurados para cada cliente.

Tabla 10

Requerimientos funcionales del visor de logs (LOG)

Identificador	Nombre	Descripción
RF-LOG-01	Listar logs	El sistema permitirá la consulta centralizada de registros operativos, ofreciendo filtros avanzados por microservicio y rango de tiempo para facilitar el diagnóstico técnico de fallos.

Una vez definidos los requerimientos funcionales que describen el comportamiento esperado del sistema, se establecieron los requerimientos no funcionales que determinan las condiciones de calidad, restricciones tecnológicas y características operativas bajo las cuales deberá funcionar la solución desarrollada.

6.1.2 Requerimientos no funcionales

Los requerimientos no funcionales establecen las características técnicas y de calidad que el sistema debe cumplir durante su desarrollo e implementación. Estos requerimientos no describen funcionalidades específicas, sino restricciones o propiedades que garantizan el correcto funcionamiento, mantenimiento e integración del sistema dentro del ecosistema tecnológico de la plataforma PIA.

A continuación, se presentan los requerimientos no funcionales identificados durante el proceso de análisis.

Tabla 11

Requerimientos no funcionales del sistema

Identificador	Nombre	Descripción
RNF-SEC-01	Seguridad basada en autenticación JWT	El sistema deberá proteger el acceso administrativo mediante la validación de un token JWT en cada petición al backend, garantizando que solo usuarios autorizados operen el sistema.
RNF-SEC-02	Encriptación de datos sensibles	El sistema deberá garantizar la protección de información sensible, como tokens de integración y claves de servicios externos, mediante mecanismos de encriptación previos a su almacenamiento.
RNF-FIA-01	Manejo de excepciones en operaciones transaccionales	El sistema deberá asegurar la consistencia de los datos en operaciones críticas mediante el uso de transacciones y el manejo de excepciones con reversión automática ante fallos.
RNF-USA-01	Validaciones visuales y mensajes en la interfaz de usuario	El sistema deberá ofrecer una interfaz intuitiva con validaciones visuales y mensajes de retroalimentación, facilitando la ejecución de tareas administrativas.
RNF-USA-02	Diseño responsivo	El sistema deberá contar con un diseño responsivo que ajuste sus componentes según el tamaño de la pantalla, manteniendo la operatividad básica en diversos dispositivos.

RNF-INT-01	Uso de servicios externos de Pinecone y de GitLab organizacional	El sistema deberá integrarse de forma eficiente con servicios externos y APIs requeridas, como Pinecone, gestionando correctamente la disponibilidad y los errores de conexión.
RNF-LOG-01	Trazabilidad mediante registro de eventos	El sistema deberá registrar eventos administrativos y errores operativos para facilitar la trazabilidad de los procesos y el diagnóstico técnico de incidencias en los microservicios.
RNF-MAN-01	Organización modular básica del sistema	El sistema deberá mantener una estructura modular que separe la lógica de negocio, el acceso a datos y la interfaz, facilitando el mantenimiento y escalabilidad futura.
RNF-TEC-01	Restricción tecnológica del sistema	El sistema deberá desarrollarse bajo el stack tecnológico definido, empleando FastAPI (Python), Flutter y PostgreSQL para asegurar compatibilidad con la arquitectura del equipo de innovación.

Donde cabe resaltar que la especificación completa de cada requerimiento funcional y no funcional, incluyendo sus precondiciones, postcondiciones, entradas, salidas, criterios de aceptación y consideraciones especiales, se encuentra documentada de manera detallada en el **Apéndice A** del presente documento.

6.2 Arquitectura de la solución

Una vez definidos los requerimientos funcionales y no funcionales del sistema, fue necesario establecer la estructura arquitectónica que permitió su implementación dentro del ecosistema tecnológico de la herramienta PIA.

La arquitectura de la solución describe la organización de los componentes del sistema, las tecnologías empleadas y la forma en que estos interactúan para satisfacer los requerimientos previamente identificados. Asimismo, permite comprender la distribución de responsabilidades entre los distintos componentes del sistema y las integraciones con servicios externos necesarias para su funcionamiento.

En esta sección se presenta la arquitectura general de la solución, incluyendo una visión tecnológica del sistema, la descripción de sus componentes principales y los flujos operativos más relevantes representados mediante diagramas de actividades.

6.2.1 Arquitectura tecnológica del sistema

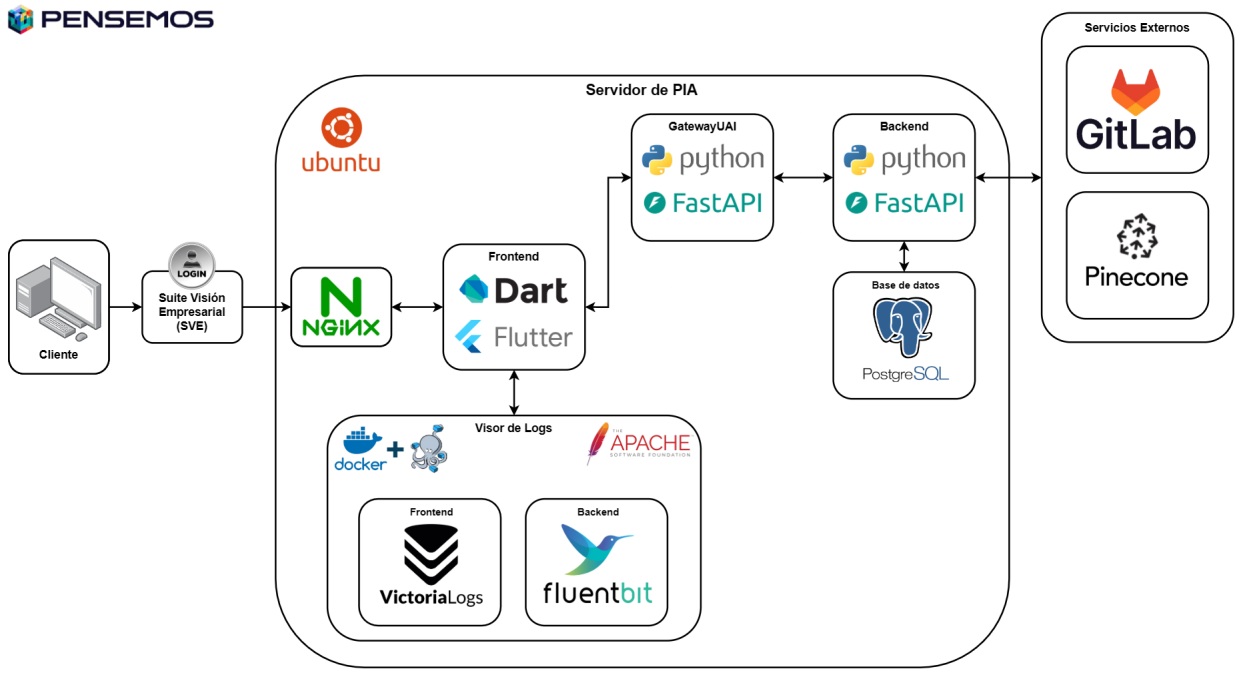
La arquitectura tecnológica del sistema describe el conjunto de tecnologías, herramientas y servicios utilizados para la implementación del aplicativo web de administración de PIA, así como la forma en que estos se articulan dentro del ecosistema tecnológico de la Suite Visión Empresarial (SVE). Esta arquitectura busca garantizar la interoperabilidad con los componentes existentes del sistema, mantener la coherencia con los lineamientos tecnológicos definidos por el equipo de innovación de Pensemos S.A. y facilitar la escalabilidad y mantenibilidad de la solución.

En este contexto, la arquitectura se diseñó considerando que el aplicativo desarrollado no funciona como un sistema independiente, sino como un componente adicional dentro del entorno operativo de PIA. Por esta razón, su implementación debía integrarse con los mecanismos de autenticación, las bases de datos existentes y los servicios externos ya utilizados por la herramienta de inteligencia artificial. La Figura 1 presenta una visión general de la arquitectura

tecnológica del sistema, mostrando los principales componentes utilizados en la solución y la forma en que estos interactúan dentro del ecosistema de PIA.

Figura 1

Arquitectura tecnológica del sistema AdminUAI



Desde la perspectiva del usuario, el acceso al sistema se realiza principalmente a través de un navegador web en un entorno de escritorio. No obstante, tal como se definió en los requerimientos no funcionales presentados en la sección 6.1, el aplicativo mantiene un nivel mínimo de operatividad en dispositivos móviles, permitiendo la consulta y ejecución básica de la mayoría de las funcionalidades administrativas.

El proceso de autenticación no es gestionado directamente por el aplicativo desarrollado. En su lugar, el acceso al panel de administración se realiza a través del mecanismo de autenticación de la Suite Visión Empresarial. Los usuarios deben iniciar sesión mediante el

sistema de login de la SVE, el cual valida las credenciales y gestiona los tokens de autenticación correspondientes. La información relacionada con los usuarios del sistema se encuentra almacenada principalmente en la base de datos de la propia suite, donde se mantienen tanto los registros de los usuarios de la SVE como aquellos asociados al funcionamiento de la herramienta PIA.

Una vez validado el proceso de autenticación, el usuario puede acceder al núcleo del sistema PIA. El tráfico hacia el aplicativo es gestionado por medio de un servidor web configurado como proxy inverso utilizando Nginx, el cual se encarga de direccionar las solicitudes hacia los distintos servicios que conforman la arquitectura. Este componente permite centralizar el acceso a los servicios internos y facilita la gestión del enrutamiento dentro del servidor.

En el caso del panel de administración desarrollado para este proyecto, denominado AdminUAI, el frontend fue implementado utilizando el lenguaje de programación Dart mediante el framework Flutter. Esta decisión tecnológica responde a lineamientos definidos por el equipo de innovación de Pensemos S.A., quienes han adoptado Flutter como tecnología para el desarrollo de interfaces modernas dentro de su ecosistema de aplicaciones. A través de esta interfaz, los administradores del sistema pueden ejecutar las operaciones relacionadas con la gestión de clientes, asistentes, prompts y demás configuraciones necesarias para el funcionamiento de PIA.

Las solicitudes generadas desde el frontend son enviadas mediante peticiones HTTP hacia un gateway previamente implementado por el equipo de innovación. Este gateway cumple la función de validar los tokens de autenticación emitidos durante el proceso de login,

asegurando que cada solicitud realizada por el cliente esté asociada a un usuario autorizado. Dicho componente sigue el mismo enfoque arquitectónico utilizado por los microservicios desarrollados por el equipo, por lo que fue implementado utilizando Python en su versión 3.11 y el framework FastAPI.

Una vez validada la solicitud, esta es redirigida hacia el backend del aplicativo AdminUAI, el cual también se encuentra implementado como un microservicio desarrollado con Python y FastAPI. Este servicio constituye el núcleo lógico del sistema, ya que es el encargado de procesar las operaciones administrativas solicitadas por los usuarios, gestionar las interacciones con la base de datos y coordinar la comunicación con servicios externos utilizados por la plataforma PIA.

El backend se conecta con una base de datos relacional PostgreSQL, la cual fue previamente creada y es administrada por el equipo de innovación de la empresa. Esta base de datos almacena la información necesaria para el funcionamiento de PIA, incluyendo registros relacionados con clientes, asistentes, configuraciones del sistema y otros elementos asociados a la operación de la herramienta de inteligencia artificial. Durante el desarrollo del presente proyecto no se realizaron modificaciones estructurales sobre esta base de datos, ya que su administración forma parte de las responsabilidades del equipo interno de la empresa.

Adicionalmente, dependiendo del tipo de operación solicitada por el usuario, el backend puede interactuar con diferentes servicios externos utilizados por la plataforma PIA. Entre estos se encuentran GitLab, utilizado para la gestión de repositorios de código asociados a los asistentes y componentes del sistema, y Pinecone, servicio especializado en el almacenamiento y consulta de representaciones vectoriales de texto empleadas para realizar búsquedas semánticas

sobre la información utilizada por los asistentes de PIA. La forma en que el backend interactúa con estos servicios externos y las operaciones específicas asociadas a cada uno de ellos se detallará con mayor profundidad en la sección correspondiente al diagrama de componentes del sistema.

Otro componente relevante dentro de la arquitectura corresponde al visor de logs del sistema. Como se mencionó previamente en la justificación del problema, la revisión de los registros de funcionamiento de PIA se realizaba originalmente de forma manual mediante acceso directo al servidor y el uso de comandos del sistema operativo. Este procedimiento resultaba poco eficiente y dificultaba la identificación rápida de errores o incidentes dentro de los servicios.

Con el objetivo de mejorar este proceso, se incorporó un visor de logs dentro del aplicativo de administración. Para ello se implementó una arquitectura compuesta por dos componentes principales. El primero corresponde a Fluent Bit, una herramienta especializada en el procesamiento y recolección de registros, la cual se encarga de leer los archivos de log generados por los diferentes servicios del sistema. Este componente también realiza tareas de procesamiento sobre los registros, como la aplicación de expresiones regulares que permiten identificar eventos complejos, por ejemplo aquellos asociados a errores que generan trazas de pila (stack trace) compuestas por múltiples líneas.

El segundo componente corresponde a VictoriaLogs, una herramienta orientada a la visualización y consulta de registros. Esta aplicación recibe los logs previamente procesados y permite presentarlos a los usuarios a través de una interfaz que facilita su exploración, filtrado y

análisis mediante consultas específicas. De esta forma, el equipo de innovación puede identificar de manera más rápida los eventos relevantes asociados al funcionamiento del sistema.

La selección de estas tecnologías respondió, entre otros factores, a criterios relacionados con su licencia de uso. En particular, ambas herramientas utilizan licencias compatibles con Apache 2.0, lo cual permite su integración dentro de soluciones comerciales sin generar conflictos con las políticas de uso de software adoptadas por la empresa. Para facilitar su despliegue y gestión, estos componentes fueron implementados mediante contenedores Docker y orquestados utilizando Docker Compose, lo que permite aislar su funcionamiento y simplificar las tareas de configuración y mantenimiento.

Finalmente, todos los componentes que conforman la arquitectura del sistema se encuentran desplegados sobre servidores que utilizan el sistema operativo Ubuntu. Este entorno constituye la base de infraestructura sobre la cual se ejecutan los distintos servicios que conforman el ecosistema de PIA. Por motivos relacionados con la política interna de la empresa, la información detallada sobre el proveedor de infraestructura utilizado para estos servidores no se incluye dentro de la documentación del proyecto.

En conjunto, esta arquitectura tecnológica permite integrar el aplicativo de administración dentro del ecosistema existente de PIA, aprovechando las herramientas y servicios previamente implementados por el equipo de innovación y garantizando la compatibilidad con los estándares tecnológicos de la organización. A partir de esta visión general, en la siguiente sección se profundiza en la estructura interna del sistema mediante la descripción de sus componentes principales y las relaciones existentes entre ellos.

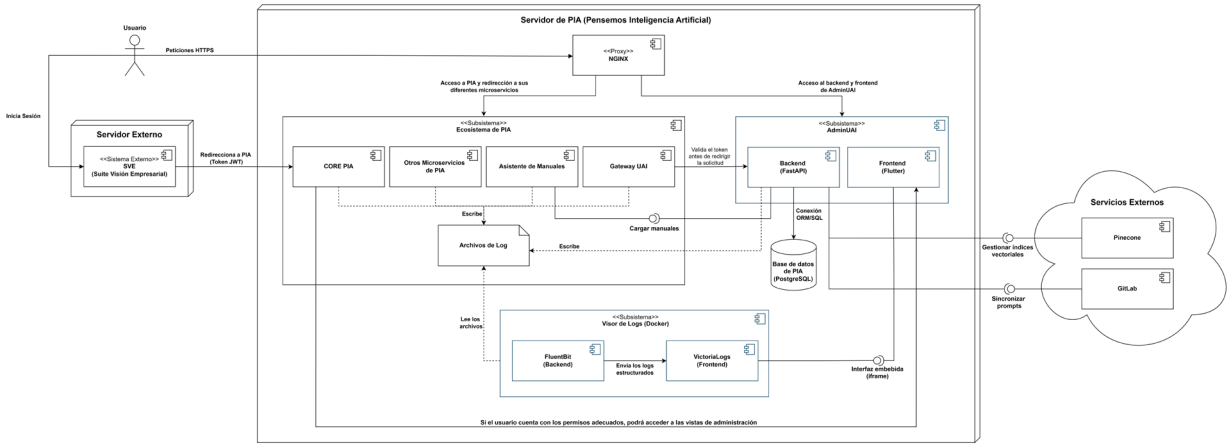
6.2.2 Arquitectura de componentes del sistema

Con el propósito de analizar con mayor detalle la organización interna de la solución propuesta, se elaboró un diagrama de componentes que permite identificar los principales módulos que conforman el sistema y las relaciones funcionales existentes entre ellos. Mientras que la arquitectura tecnológica presentada en la sección anterior describe las tecnologías utilizadas y la infraestructura general del entorno, el diagrama de componentes permite visualizar con mayor precisión la organización interna del sistema y la forma en que interactúan los diferentes subsistemas que participan en el funcionamiento de PIA y del aplicativo de administración desarrollado en este proyecto.

La Figura 2 presenta el diagrama de componentes del sistema. En este se distinguen dos grupos principales de elementos: por un lado, los componentes que ya formaban parte del ecosistema tecnológico de PIA y que son gestionados por el equipo de innovación de Pensemos S.A., y por otro lado los componentes que fueron desarrollados específicamente como parte de la solución implementada en el presente proyecto. Con el fin de facilitar su identificación dentro del diagrama, los elementos que hacen parte de la solución desarrollada se representan con un borde de color azul, mientras que aquellos mostrados en color negro corresponden a sistemas o servicios previamente existentes dentro de la infraestructura de la organización. Debido al nivel de detalle presentado, una versión en mayor resolución del diagrama puede consultarse en el **Apéndice B** del presente documento.

Figura 2

Arquitectura de componentes del sistema AdminUAI



En la parte izquierda del diagrama se observa el flujo inicial de acceso al sistema. El usuario interactúa con la plataforma a través de peticiones HTTPS generadas desde su navegador web. El proceso de autenticación es gestionado por la Suite Visión Empresarial (SVE), sistema externo al ecosistema de microservicios de PIA que se ejecuta en un servidor independiente y cuya implementación no forma parte del alcance del presente proyecto. Una vez que el usuario se autentica correctamente en la SVE, este sistema redirige la solicitud hacia el entorno de PIA utilizando un token de autenticación basado en JSON Web Token (JWT), el cual permite validar la identidad del usuario dentro de los diferentes servicios que componen la plataforma.

Después de esta redirección, el tráfico es recibido por el servidor web configurado como proxy inverso mediante NGINX. Este componente se encarga de centralizar las peticiones entrantes y redirigirlas hacia los diferentes microservicios disponibles dentro del ecosistema de PIA. Como se observa en el diagrama, a través de este mecanismo el usuario puede acceder tanto al núcleo funcional de la herramienta como a los diferentes servicios que conforman su arquitectura interna.

Dentro del ecosistema de PIA se distinguen varios componentes que ya existían previamente en la plataforma y que no fueron modificados durante el desarrollo del proyecto. Entre ellos se encuentra el Core de PIA, responsable de la interfaz conversacional utilizada por los usuarios finales de la herramienta, así como otros microservicios internos cuya lógica de negocio forma parte de la implementación propietaria del sistema. Debido a consideraciones relacionadas con la confidencialidad de la arquitectura interna de la plataforma, estos microservicios se representan de manera general dentro del diagrama sin detallar su implementación ni su lógica interna.

Otro componente relevante dentro de este subsistema es el Asistente de manuales, el cual expone una interfaz de programación (API) que permite gestionar la carga y actualización de los manuales de la Suite Visión Empresarial. Este servicio resulta particularmente importante para el funcionamiento del aplicativo de administración desarrollado, ya que permite registrar y actualizar la información utilizada por los asistentes de PIA para responder consultas relacionadas con la documentación de la SVE. En este sentido, su utilización se encuentra directamente relacionada con los requerimientos funcionales definidos en la sección 6.1, específicamente aquellos asociados a la gestión de versiones y manuales de la plataforma (RF-SVE).

En el caso del aplicativo de administración, el diagrama muestra el subsistema denominado AdminUAI, el cual agrupa los componentes principales implementados como parte de la solución desarrollada. Este subsistema está compuesto por dos elementos fundamentales: el frontend desarrollado con Flutter y el backend implementado mediante FastAPI. El frontend constituye la interfaz a través de la cual los administradores interactúan con el sistema, permitiendo ejecutar operaciones de gestión relacionadas con clientes, asistentes, prompts,

configuraciones del sistema y otros elementos necesarios para el mantenimiento operativo de PIA.

Las solicitudes generadas desde esta interfaz son enviadas hacia el backend del sistema, pasando previamente por el Gateway UAI, el cual cumple la función de validar los tokens de autenticación generados durante el proceso de inicio de sesión. Una vez verificada la validez del token, las solicitudes son procesadas por el backend de AdminUAI, el cual implementa la lógica de negocio necesaria para ejecutar las operaciones administrativas del sistema. Este componente mantiene una conexión con la base de datos relacional PostgreSQL utilizada por PIA, utilizando mecanismos de persistencia basados en ORM y consultas SQL para almacenar y recuperar la información requerida por la aplicación.

Adicionalmente, el backend interactúa con diferentes servicios externos que complementan el funcionamiento de la plataforma. Entre ellos se encuentra GitLab, servicio utilizado para almacenar los repositorios que contienen los prompts asociados a los asistentes de PIA. A través de la integración con la API de GitLab, el sistema puede consultar y sincronizar automáticamente estos archivos, permitiendo implementar funcionalidades como la descarga y actualización de prompts definidas en los requerimientos funcionales correspondientes al módulo de gestión de prompts (RF-PRO).

De manera similar, el sistema también se integra con el servicio Pinecone, el cual permite gestionar los índices vectoriales utilizados por los asistentes de PIA para realizar búsquedas semánticas sobre la información disponible. Esta integración resulta particularmente relevante para los procesos de creación de nuevos clientes dentro del sistema, ya que el aplicativo desarrollado puede generar automáticamente un índice vectorial asociado a cada cliente mediante

la API de Pinecone, cumpliendo así con los requerimientos funcionales relacionados con la gestión de clientes en el entorno de PIA (RF-CLI).

Otro componente importante representado en el diagrama corresponde al subsistema del visor de logs, el cual fue incorporado como parte de la solución desarrollada para mejorar la trazabilidad del funcionamiento de los diferentes microservicios. Como se observa en la figura, este subsistema está compuesto por dos herramientas principales: Fluent Bit y VictoriaLogs, las cuales se ejecutan dentro de contenedores gestionados mediante Docker.

Dentro de este proceso, los diferentes microservicios que conforman el ecosistema de PIA, así como los componentes de AdminUAI, generan de manera continua archivos de registro que contienen información sobre su funcionamiento y posibles eventos de error. Estos archivos de log son leídos por Fluent Bit, el cual se encarga de procesar y estructurar los eventos registrados por los servicios. Posteriormente, la información procesada es enviada hacia VictoriaLogs, donde puede ser consultada a través de una interfaz que permite realizar búsquedas, aplicar filtros y analizar los eventos registrados en el sistema.

La interfaz gráfica proporcionada por VictoriaLogs es integrada dentro del frontend de AdminUAI mediante un mecanismo de embebido basado en iframe, lo que permite que los administradores del sistema accedan a la consulta de logs directamente desde el panel de administración sin necesidad de interactuar con herramientas externas o acceder manualmente al servidor.

En conjunto, el diagrama de componentes permite comprender con mayor claridad la forma en que los diferentes subsistemas del entorno de PIA interactúan entre sí y cómo el aplicativo de administración desarrollado se integra dentro de esta arquitectura sin modificar la

estructura existente de la plataforma. De esta manera, la solución implementada actúa como una capa adicional de administración que aprovecha los servicios y recursos previamente disponibles en el ecosistema tecnológico de la organización.

Una vez descrita la estructura de componentes que conforma el sistema, en la siguiente sección se presentan los diagramas de actividades que describen el comportamiento de los principales procesos operativos del aplicativo de administración, permitiendo comprender con mayor detalle el flujo de interacción entre los diferentes componentes durante la ejecución de algunas de las funcionalidades implementadas.

6.2.3 Diagramas de actividades de los procesos principales

Con el fin de describir el comportamiento operativo del aplicativo desarrollado, se elaboraron diagramas de actividades que representan el flujo de ejecución de algunos de los procesos más relevantes dentro del sistema. A diferencia de los diagramas presentados en las secciones anteriores, los cuales se enfocan en la estructura y organización de la arquitectura, los diagramas de actividades permiten visualizar la secuencia lógica de acciones que realizan los distintos actores del sistema durante la ejecución de las funcionalidades implementadas.

Estos diagramas facilitan la comprensión del funcionamiento del aplicativo desde una perspectiva funcional, mostrando las decisiones que se presentan durante la interacción del usuario con el sistema, así como las validaciones y respuestas generadas por los diferentes componentes que participan en cada proceso.

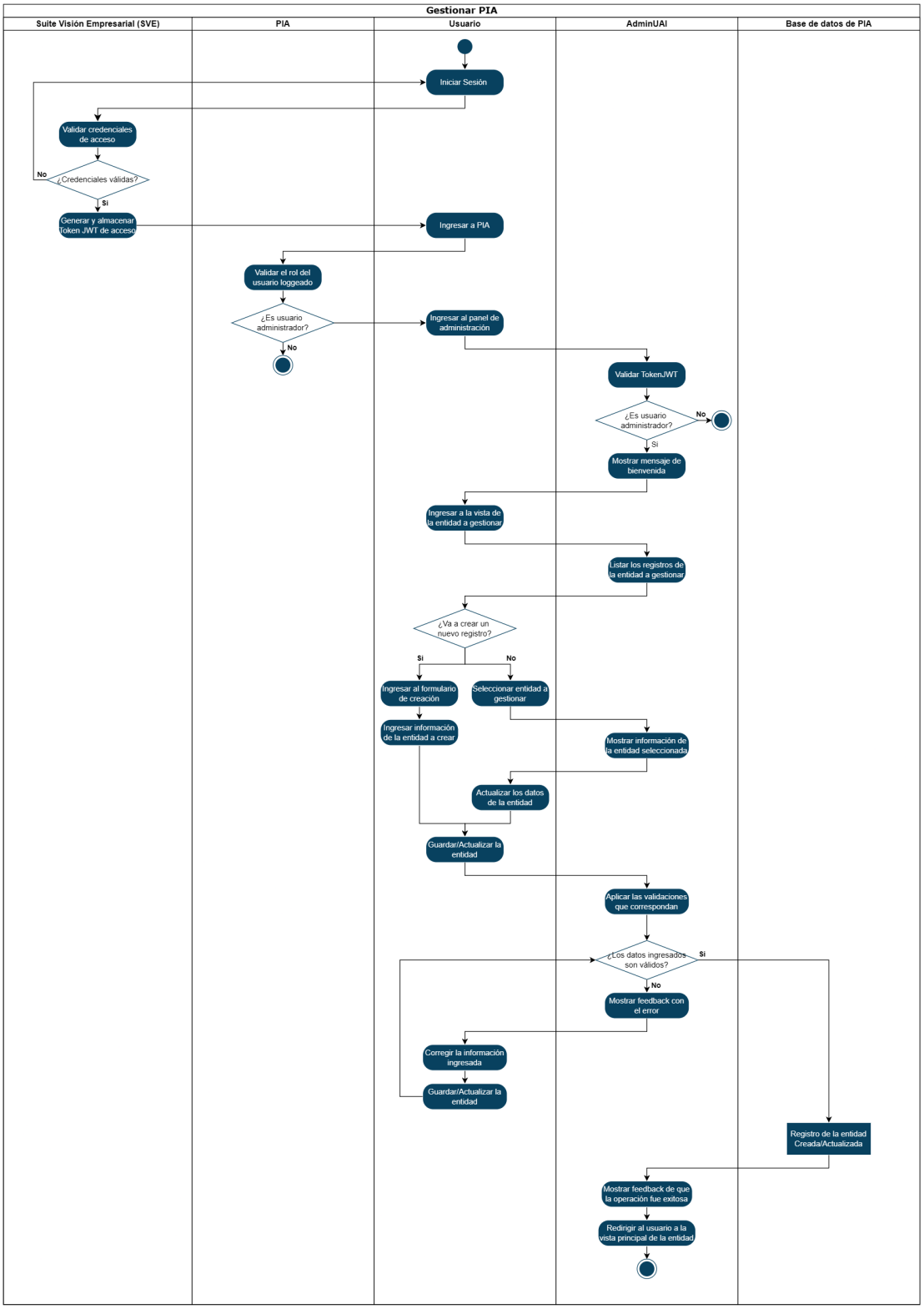
6.2.3.1 Flujo general del sistema. Como se observa en el diagrama, el proceso inicia cuando el usuario realiza el inicio de sesión a través de la Suite Visión Empresarial. Este sistema se encarga de validar las credenciales de acceso ingresadas y, en caso de que estas sean correctas, generar el correspondiente token de autenticación basado en JSON Web Token (JWT). Dicho token es utilizado posteriormente por los servicios del ecosistema de PIA para verificar la identidad y los permisos del usuario durante su interacción con la plataforma.

Una vez autenticado, el usuario accede a la herramienta PIA. Durante este proceso se validan los permisos asociados al usuario autenticado con el fin de determinar si cuenta con privilegios administrativos dentro del sistema. En caso de que el usuario posea estos permisos, la interfaz de PIA habilita el acceso al panel de administración AdminUAI, permitiendo que el usuario sea redirigido hacia este módulo para realizar tareas de gestión sobre las distintas entidades del sistema.

Al ingresar al panel de administración, el sistema realiza nuevamente una verificación del token JWT almacenado en el navegador del usuario. Esta validación permite garantizar que la sesión continúe siendo válida y que el token no haya sido alterado o expirado. En caso de que el token no sea válido, el sistema informa al usuario sobre la situación y lo redirige hacia el proceso de autenticación correspondiente. Si la validación es exitosa, el usuario puede navegar libremente por las diferentes secciones del panel de administración.

Figura 3

Flujo general del sistema AdminUAI



A partir de este punto, el usuario puede acceder a las distintas vistas del sistema según la entidad que desee gestionar, tales como clientes, asistentes, prompts, usuarios o grupos. Cuando se accede a una de estas vistas, el sistema consulta la información correspondiente y presenta al usuario el listado de registros disponibles, permitiendo aplicar filtros, seleccionar elementos específicos o consultar el detalle de un registro determinado.

En el caso de que el usuario únicamente desee consultar la información de una entidad, el sistema permite acceder a la vista detallada del registro seleccionado sin realizar modificaciones sobre los datos almacenados. Este comportamiento corresponde con las operaciones de consulta definidas dentro de los requerimientos funcionales del sistema presentados en la sección 6.1.

Por otro lado, cuando el usuario decide crear un nuevo registro o modificar uno existente, el sistema redirige al formulario correspondiente para la captura o actualización de la información. Durante esta etapa, el frontend del aplicativo realiza validaciones preliminares sobre los datos ingresados con el fin de garantizar que cumplan con los formatos y restricciones definidos en la interfaz, proporcionando retroalimentación visual al usuario en caso de detectar inconsistencias. Este comportamiento se encuentra alineado con el requerimiento no funcional relacionado con la usabilidad de la interfaz (RNF-USA-01).

Una vez que el usuario completa la información solicitada y selecciona la opción de guardar, la solicitud es enviada al backend del sistema, donde se ejecutan validaciones adicionales relacionadas con la integridad de los datos y las reglas de negocio definidas para la entidad correspondiente. Estas validaciones permiten verificar, por ejemplo, que no existan conflictos con registros previamente almacenados o que la información ingresada cumpla con las restricciones establecidas por el sistema.

Si durante este proceso se detecta algún error o inconsistencia, el sistema notifica al usuario mediante un mensaje de retroalimentación que describe el problema identificado, permitiendo que el usuario realice las correcciones necesarias antes de volver a enviar la solicitud. En caso contrario, cuando la información es válida, el sistema procede a registrar o actualizar los datos en la base de datos de PIA.

Finalmente, una vez completada la operación de forma exitosa, el sistema informa al usuario que la acción fue realizada correctamente y lo redirige nuevamente a la vista principal de la entidad que se encontraba gestionando, donde el nuevo registro o la actualización realizada se refleja dentro del listado correspondiente.

Este flujo general describe el comportamiento común que comparten la mayoría de las operaciones administrativas implementadas en el sistema. No obstante, algunos procesos presentan una lógica de ejecución más compleja debido a las integraciones y validaciones adicionales que requieren. En la siguiente sección se describe con mayor detalle el proceso correspondiente a la creación de un nuevo cliente dentro de la plataforma, el cual constituye uno de los flujos más representativos del funcionamiento del aplicativo de administración.

6.2.3.2 Proceso de creación de un nuevo cliente. Como se expuso en la justificación del problema, el proceso de creación de nuevos clientes dentro de la herramienta PIA se realizaba anteriormente de forma manual mediante el uso de herramientas de administración de bases de datos, lo que implicaba ingresar múltiples registros de manera directa en las tablas del sistema. Este procedimiento no solo demandaba tiempo por parte del equipo de innovación, sino que también incrementaba la probabilidad de errores humanos al repetir configuraciones similares

para cada nuevo cliente, especialmente en la creación de asistentes, prompts y otros elementos asociados al funcionamiento de la herramienta.

Con el desarrollo del aplicativo de administración propuesto en este trabajo, dicho proceso fue transformado en un flujo guiado a través de un formulario que permite registrar la información principal del cliente mientras el sistema se encarga de ejecutar automáticamente gran parte de las tareas de configuración necesarias. De esta manera, operaciones que anteriormente requerían múltiples acciones manuales ahora pueden completarse en pocos minutos mediante una única interacción con la interfaz del sistema.

La Figura 4 presenta el diagrama de actividades correspondiente al proceso de creación de un nuevo cliente dentro del aplicativo de administración. Debido a la extensión y nivel de detalle de este diagrama, su visualización completa se encuentra disponible en el **Apéndice C** del presente documento.

Como se observa en el diagrama, el proceso inicia cuando el usuario accede al panel de administración y selecciona la opción de crear un nuevo cliente. En ese momento el sistema carga el formulario correspondiente y realiza de manera transparente una consulta a la base de datos para obtener las versiones disponibles de la Suite Visión Empresarial (SVE), las cuales son almacenadas temporalmente con el fin de habilitar mecanismos de autocompletado durante el diligenciamiento del formulario.

Una vez cargada la interfaz, el usuario procede a ingresar la información inicial del cliente, incluyendo la URL de la instancia de la SVE y el identificador de la licencia correspondiente. Durante esta etapa el sistema aplica validaciones básicas sobre los campos ingresados, verificando por ejemplo que la URL tenga un formato válido. En caso de detectar

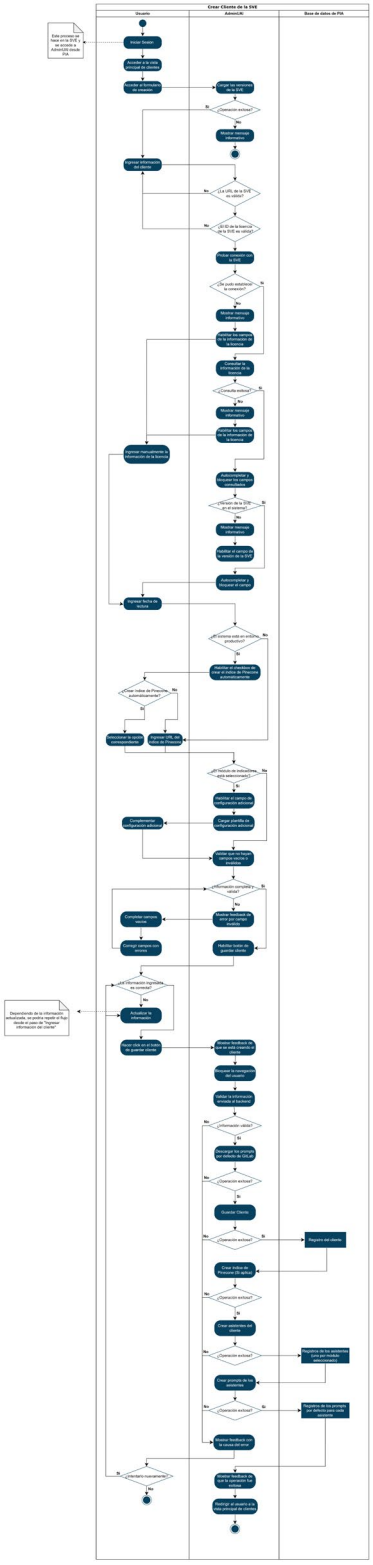
inconsistencias, el sistema muestra inmediatamente mensajes de retroalimentación visual al usuario, permitiendo realizar las correcciones necesarias antes de continuar con el proceso. Este comportamiento se encuentra alineado con el requerimiento no funcional relacionado con la usabilidad de la interfaz (RNF-USA-01).

Cuando la información mínima requerida ha sido ingresada, el sistema intenta establecer una conexión con la instancia de la SVE utilizando los datos proporcionados por el usuario, de acuerdo con el requerimiento funcional RF-CLI-03. Si la conexión no puede establecerse, el usuario puede continuar diligenciando el formulario, ya que esta situación puede presentarse en escenarios como entornos de pruebas o instancias de la SVE que aún no se encuentran disponibles. No obstante, cuando la conexión es exitosa, el sistema consulta automáticamente la información de la licencia a través de la API de la SVE, recuperando datos como la versión del sistema, los módulos habilitados y otras configuraciones relevantes. En caso de obtener una respuesta válida, los campos correspondientes dentro del formulario son completados automáticamente y bloqueados para evitar inconsistencias entre la información registrada y los datos oficiales de la licencia.

Posteriormente, el usuario continúa ingresando información adicional requerida para la configuración del cliente, como la fecha de lectura o las configuraciones específicas asociadas a determinados módulos. En esta etapa también es posible habilitar la creación automática del índice vectorial asociado al cliente dentro del servicio Pinecone, funcionalidad que corresponde al requerimiento RF-CLI-05. Dependiendo del entorno de despliegue del sistema o de la existencia previa de un índice con el mismo nombre, el usuario puede optar por deshabilitar esta opción e ingresar manualmente la referencia del índice que deberá utilizar el cliente dentro de la plataforma.

Figura 4

Proceso de creación de un cliente en el sistema AdminUAI



A lo largo del diligenciamiento del formulario, el sistema continúa aplicando validaciones sobre la información ingresada con el fin de asegurar que esta cumpla con las reglas de negocio definidas para la creación de nuevos clientes. Cuando todos los campos requeridos cumplen con estas validaciones, se habilita la opción de guardar el nuevo cliente dentro del sistema.

Al confirmar la operación, el aplicativo bloquea temporalmente la navegación del usuario y muestra un mensaje indicando que el proceso de creación se encuentra en ejecución. A partir de este momento el backend del sistema inicia una serie de operaciones automatizadas orientadas a completar la configuración del cliente dentro del ecosistema de PIA.

En primer lugar, el sistema descarga y sincroniza los prompts base almacenados en los repositorios de GitLab con el entorno local del backend, garantizando que los asistentes creados para el nuevo cliente utilicen la versión más reciente de estas configuraciones. Posteriormente se registra el cliente en la base de datos de PIA y, cuando corresponde, se realiza la creación o asociación del índice vectorial dentro del servicio Pinecone.

Una vez completadas estas operaciones, el sistema procede a generar automáticamente los asistentes asociados al cliente según los módulos habilitados en su licencia (RF-CLI-06). Finalmente, para cada uno de estos asistentes se crean los prompts correspondientes utilizando las configuraciones previamente sincronizadas desde GitLab (RF-CLI-07). Todo este proceso se ejecuta dentro de una transacción controlada por el backend del sistema, lo que permite garantizar la consistencia de los datos mediante mecanismos de reversión automática en caso de que ocurra algún error durante la ejecución de las operaciones. Este comportamiento se

encuentra alineado con el requerimiento no funcional relacionado con el manejo de excepciones en operaciones transaccionales (RNF-FIA-01).

Una vez finalizado el proceso de creación sin errores, el sistema informa al usuario que la operación fue completada exitosamente y lo redirige nuevamente a la vista principal de clientes, donde podrá visualizar el registro recién creado y realizar consultas o ajustes adicionales sobre su configuración.

Este flujo representa uno de los procesos más completos implementados dentro del aplicativo de administración, ya que integra múltiples operaciones automáticas que anteriormente debían ejecutarse manualmente por parte del equipo de innovación.

Una vez descritos los principales flujos operativos del sistema y la forma en que interactúan los distintos componentes durante la ejecución de estas funcionalidades, en la siguiente sección se presentan algunas consideraciones de diseño que orientaron la documentación de la arquitectura y las decisiones técnicas adoptadas durante el desarrollo del aplicativo.

6.2.4 Consideraciones de diseño

Durante el diseño y la documentación de la arquitectura del sistema se tomaron algunas decisiones relacionadas con los artefactos de modelado incluidos en este documento y con el alcance de la información técnica presentada. Estas decisiones se orientaron principalmente a evitar la redundancia con secciones previamente desarrolladas, mantener la claridad de la documentación y respetar las restricciones tecnológicas y de confidencialidad definidas por el equipo de innovación de Pensemos S.A.

En primer lugar, el aplicativo desarrollado cuenta únicamente con un rol activo dentro del sistema, correspondiente al usuario con permisos de administrador. Este usuario es el encargado de gestionar las distintas entidades que componen la base de datos de PIA, tales como clientes, asistentes, prompts, usuarios y demás configuraciones asociadas al funcionamiento de la herramienta. En este contexto, las funcionalidades disponibles para este rol se encuentran descritas de manera detallada en la sección 6.1 mediante la especificación de los requerimientos funcionales del sistema.

En segundo lugar, el presente documento no incluye el diagrama entidad–relación correspondiente a la base de datos de PIA. Esta decisión responde a dos razones principales. Por una parte, durante el desarrollo del proyecto no se realizaron modificaciones estructurales sobre la base de datos existente, ya que su administración y evolución forman parte de las responsabilidades del equipo de innovación de Pensemos S.A. Por otra parte, por solicitud directa de dicho equipo, se optó por no adjuntar este artefacto dentro de la documentación pública del proyecto, con el fin de preservar la confidencialidad de la estructura interna del sistema.

Finalmente, es importante señalar que el desarrollo del aplicativo se realizó respetando el stack tecnológico definido por el equipo de innovación de la empresa, de acuerdo con lo establecido en el requerimiento no funcional RNF-TEC-01. En consecuencia, la implementación del sistema se llevó a cabo utilizando el framework FastAPI sobre Python para el desarrollo del backend, Flutter para la construcción de la interfaz de usuario y PostgreSQL como sistema gestor de base de datos. La adopción de este conjunto de tecnologías permitió garantizar la compatibilidad del aplicativo con la arquitectura existente de la plataforma PIA y facilitar su integración dentro del ecosistema tecnológico previamente implementado por la organización.

Una vez presentadas las principales decisiones de diseño que orientaron la documentación de la arquitectura del sistema, en la siguiente sección se exponen los resultados de la implementación del aplicativo desarrollado. En esta sección se describen las funcionalidades implementadas a partir de los requerimientos funcionales definidos previamente, presentando una explicación general de su comportamiento.

6.3 Implementación del sistema AdminUAI

En esta sección se presentan los resultados de la implementación del aplicativo AdminUAI, a través de evidencias visuales que permiten ilustrar el comportamiento de las funcionalidades desarrolladas dentro del entorno de PIA**.**

6.3.1 Acceso y control de autenticación

Para ilustrar el comportamiento del sistema en el proceso de acceso, se presentan a continuación las vistas correspondientes al entorno de PIA con y sin permisos de administrador.

La Figura 5 muestra la interfaz de PIA cuando el usuario no cuenta con permisos de acceso al panel de administración, mientras que la Figura 6 presenta el comportamiento del sistema cuando el usuario dispone de dichos permisos, habilitando el acceso a AdminUAI.

Figura 5

Acceso a PIA sin permisos administrativos

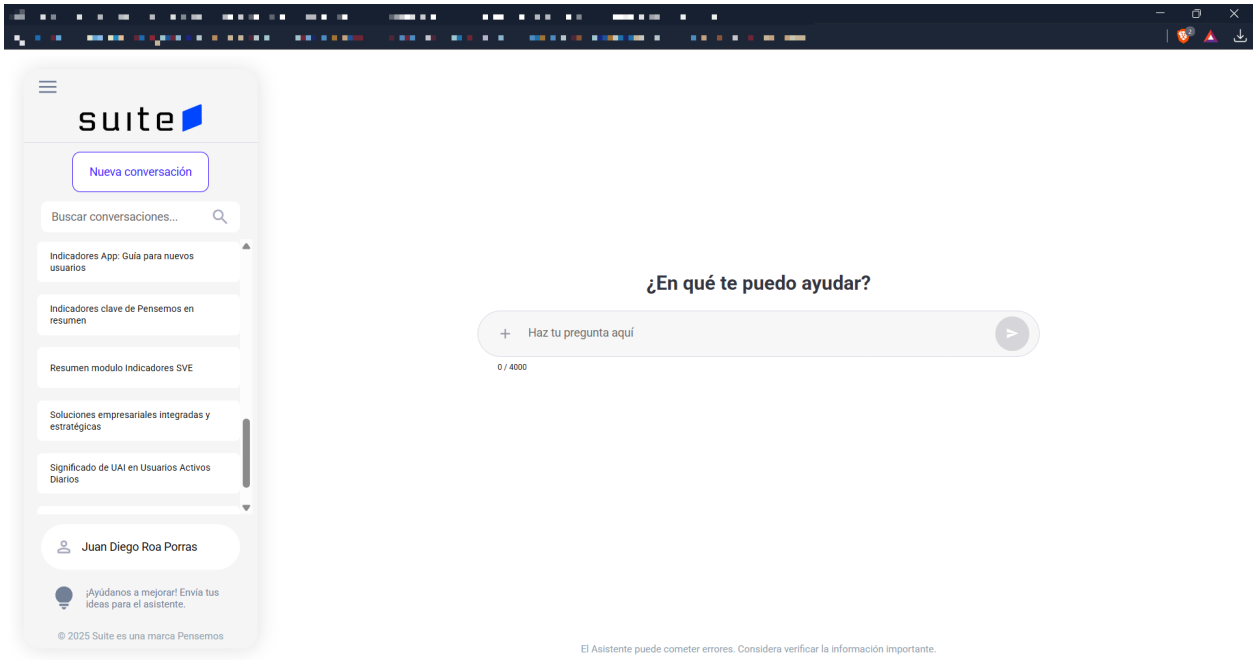
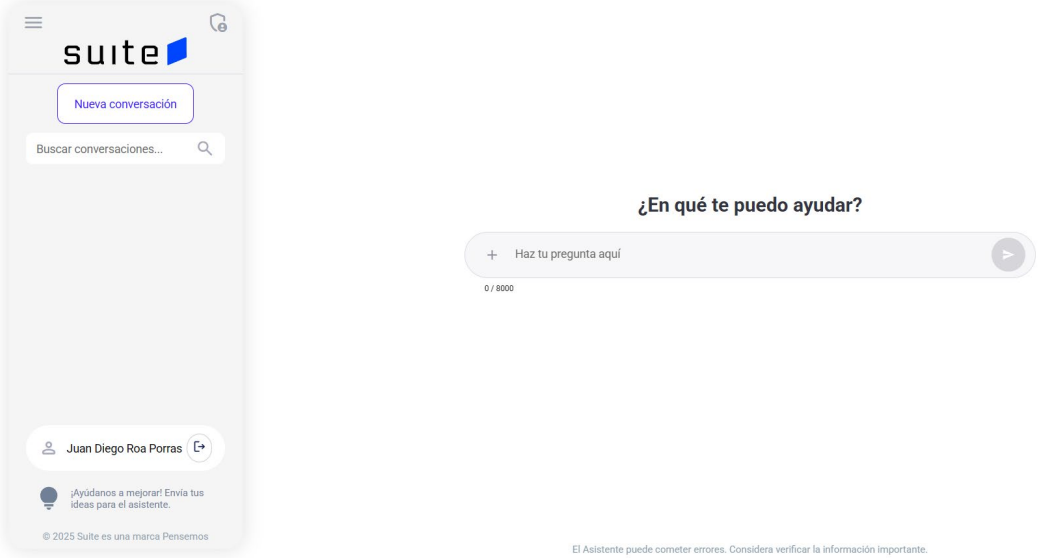


Figura 6

Acceso a PIA con permisos administrativos



Como se observa en las Figuras 5 y 6, al ingresar a la herramienta PIA se realiza la validación de los permisos asociados al usuario autenticado, lo que determina la interfaz que se presenta en pantalla. En caso de no contar con privilegios administrativos, el sistema muestra la vista correspondiente sin acceso al panel de administración (Figura 5); mientras que, cuando el usuario dispone de dichos permisos, se habilita el acceso a AdminUAI mediante un icono ubicado en la parte superior derecha del panel lateral (Figura 6). Al seleccionar este icono, el usuario es redirigido a la página de bienvenida del aplicativo de administración, como se muestra en la Figura 7.

Figura 7

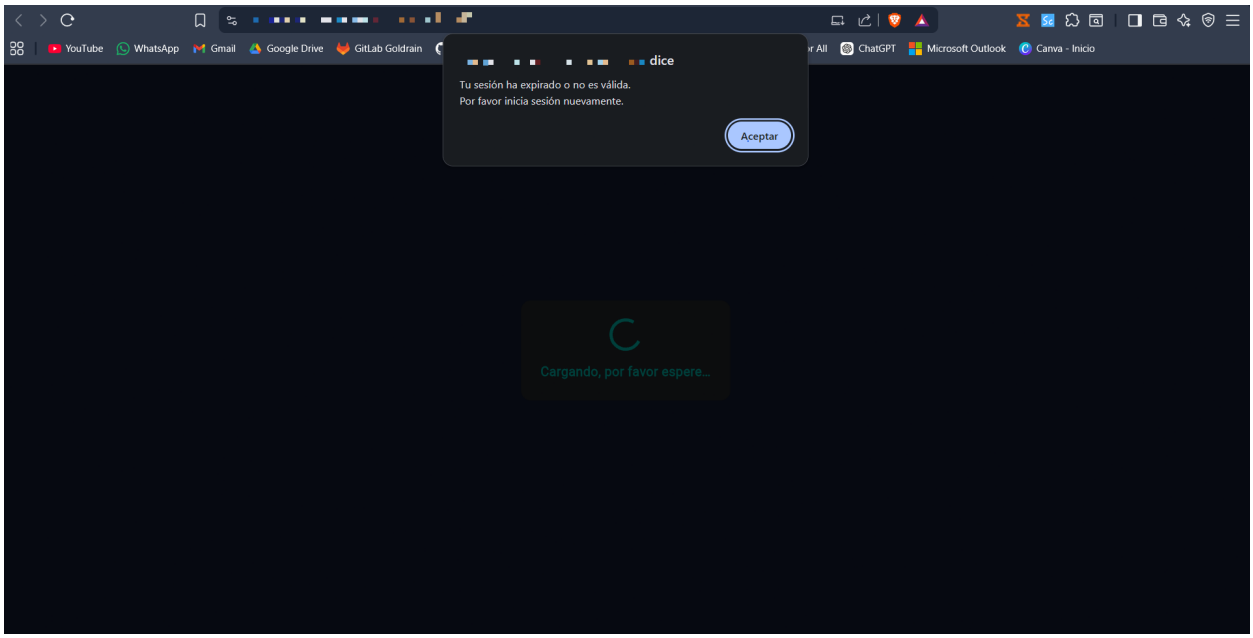
Página de bienvenida del sistema AdminUAI

Una vez dentro del sistema, se continúa validando la vigencia y los permisos asociados al token JWT durante la navegación. En caso de que el token no sea válido o haya expirado, el

sistema restringe el acceso a las funcionalidades administrativas y muestra una alerta informativa solicitando al usuario iniciar sesión nuevamente, como se observa en la Figura 8.

Figura 8

Alerta de sesión inválida o expirada en AdminUAI

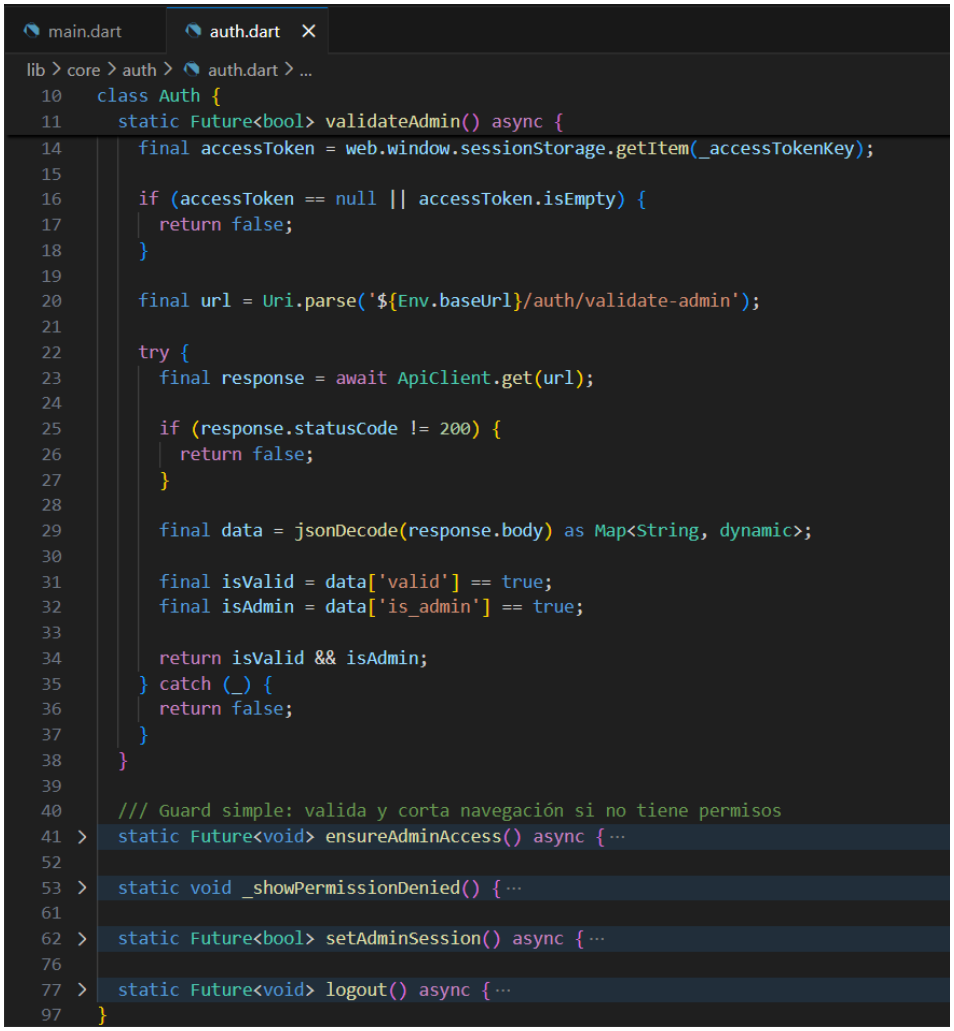


Este mecanismo de validación es gestionado tanto en el frontend como en el backend del sistema, como se detalla en las Figuras 9 y 10. En el frontend, la verificación se implementa mediante la clase Auth, a través del método validateAdmin(), el cual consume el endpoint correspondiente y evalúa la validez del token JWT junto con los permisos del usuario. De forma complementaria, la función ensureAdminAccess() ejecuta esta validación durante la navegación, restringiendo el acceso a las vistas en caso de no cumplir con los privilegios requeridos.

Adicionalmente, el método setAdminSession() permite establecer una cookie utilizada por el visor de logs, con el fin de garantizar que únicamente usuarios administradores puedan acceder a esta funcionalidad. Por su parte, la función logout() elimina tanto el token almacenado en el navegador como la cookie asociada, cerrando completamente la sesión del usuario.

Figura 9

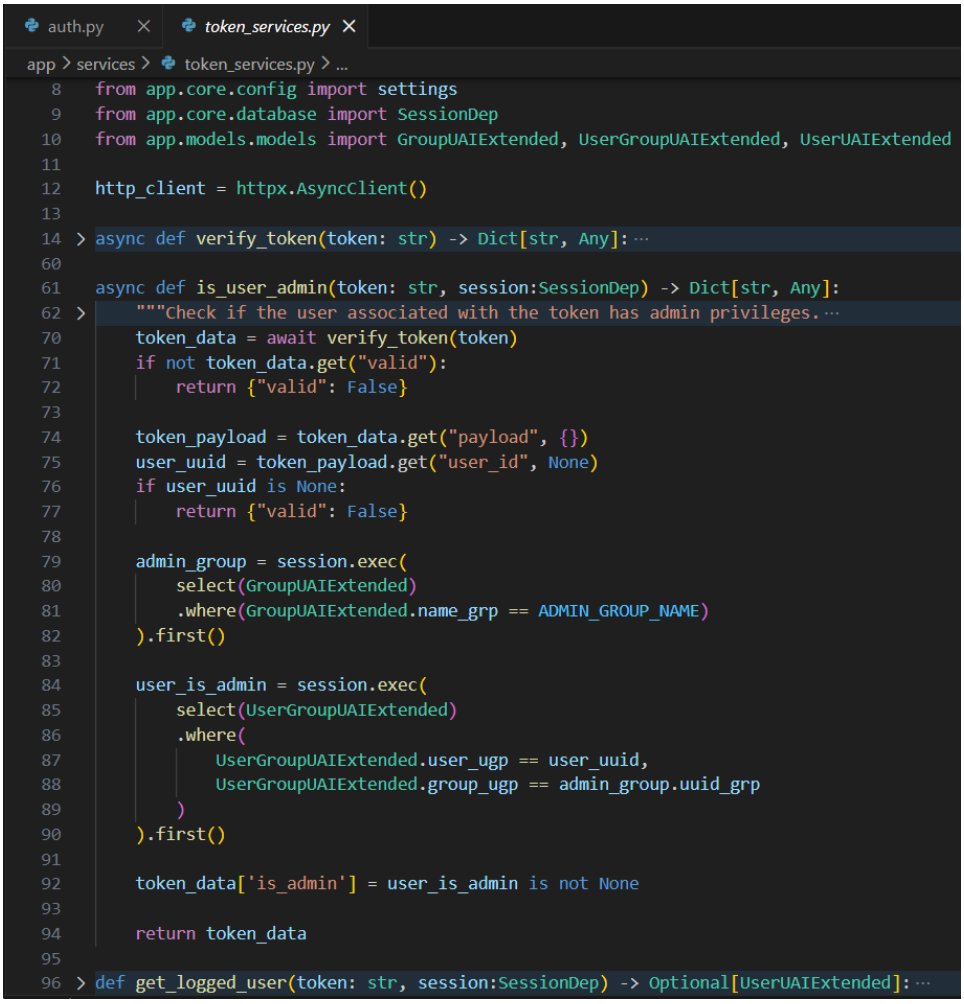
Validación de usuario administrador en el frontend del sistema AdminUAI



Por otro lado, en la Figura 10 se presenta la lógica implementada en el backend para la validación del usuario administrador. En este caso, la función verify_token se encarga de consumir un microservicio interno del ecosistema de PIA para comprobar la validez del token JWT. Posteriormente, a partir de la información contenida en el token, se realiza una consulta a la base de datos con el fin de verificar si el usuario pertenece al grupo de administradores, utilizando como referencia el nombre del grupo definido como una constante. Esta decisión permite mantener independencia entre los entornos de pruebas y producción.

Figura 10

Validación de usuario administrador en el backend del sistema AdminUAI



Cabe resaltar que, por lineamientos del equipo de innovación de Pensemos S.A., no se incluye mayor cantidad de código fuente dentro del presente documento. No obstante, los fragmentos presentados en estas figuras se incorporan como evidencia de la implementación del mecanismo de autenticación y control de acceso, así como del cumplimiento del requerimiento no funcional RNF-TEC-01 relacionado con la adopción del stack tecnológico definido en la arquitectura de la solución (Sección 6.2).

Haciendo uso continuo de estos mecanismos de validación, en la siguiente sección se presentan las interfaces implementadas en los diferentes módulos del sistema.

6.3.2 Interfaces del sistema AdminUAI

En esta sección se presentan las principales interfaces resultantes de la implementación del sistema AdminUAI, organizadas de acuerdo con los módulos definidos en la sección de requerimientos. A diferencia de la sección anterior, donde se detalló el comportamiento interno de los mecanismos de autenticación, en este apartado se prioriza la evidencia visual del sistema mediante pantallazos, con el objetivo de demostrar el cumplimiento de los requerimientos funcionales y no funcionales definidos en la sección 6.1.

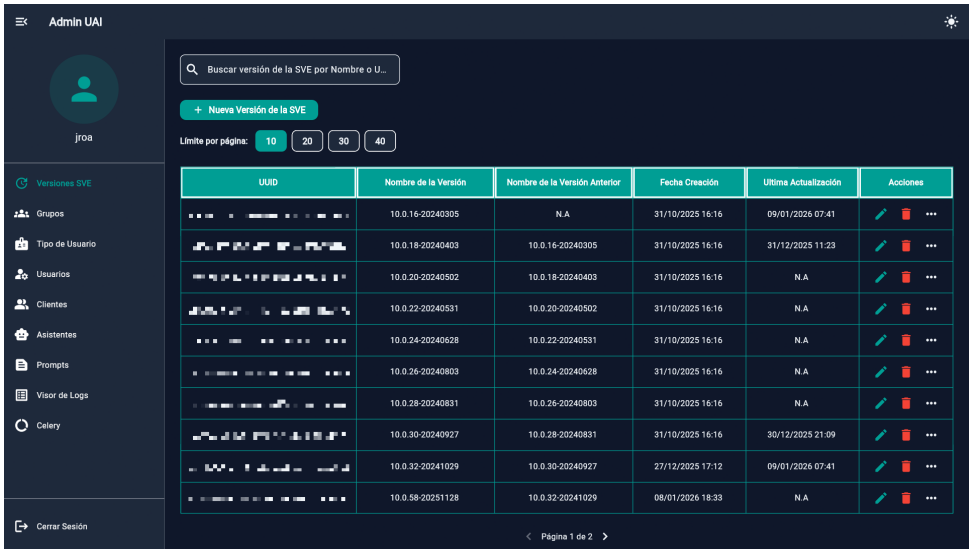
Cada figura corresponde a una vista implementada en el sistema y se acompaña de una breve descripción que permite relacionarla directamente con los requerimientos que satisface. Adicionalmente, se destacan comportamientos transversales, tales como validaciones visuales, mensajes de retroalimentación y confirmaciones de acciones, los cuales responden principalmente al requerimiento no funcional RNF-USA-01.

6.3.2.1 Módulo de versiones de la SVE. Como punto de partida en la navegación del sistema, se presenta el módulo de versiones de la SVE, el cual permite gestionar las versiones disponibles junto con sus manuales asociados.

La Figura 11 presenta el listado de versiones de la SVE, dando cumplimiento al requerimiento RF-SVE-01, evidenciando la paginación, la búsqueda por nombre y el acceso a acciones sobre cada registro.

Figura 11

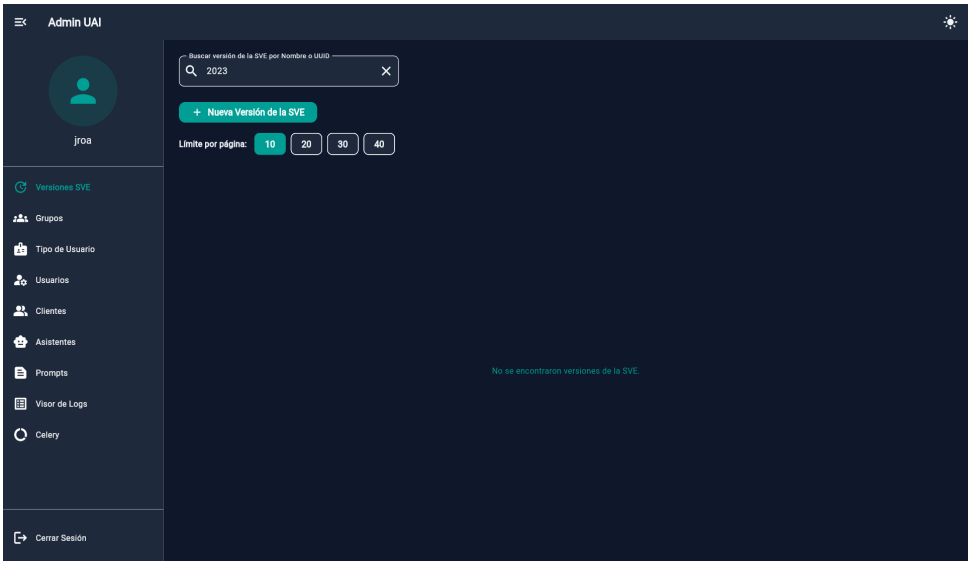
Listado de versiones de la SVE



En la Figura 12 se observa el mensaje de ausencia de resultados cuando no existen coincidencias con los criterios de búsqueda, comportamiento que responde tanto al RF-SVE-01 como al RNF-USA-01, y que se implementa de manera transversal en todos los módulos del sistema.

Figura 12

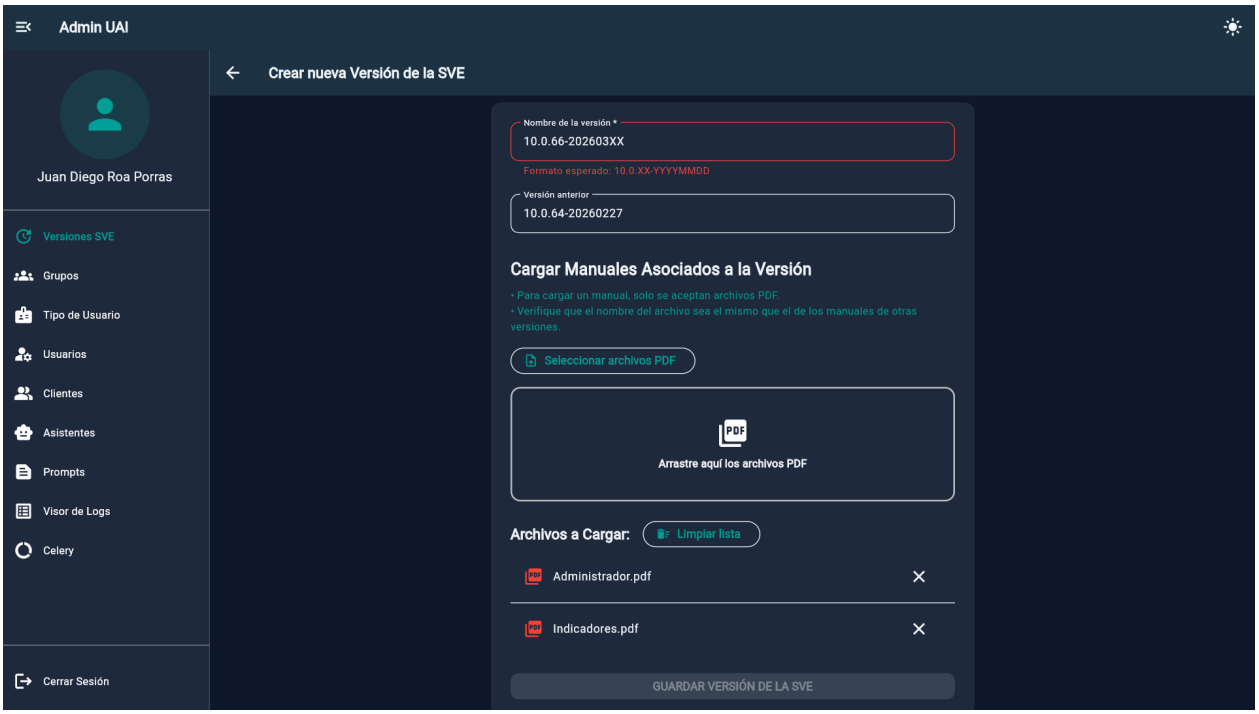
Mensaje de ausencia de resultados en el módulo de versiones de la SVE



En la Figura 13 se presenta el formulario de creación de versiones de la SVE, mediante el cual se da cumplimiento al requerimiento RF-SVE-02, permitiendo registrar nuevas versiones y asociarles sus respectivos manuales en formato PDF.

Figura 13

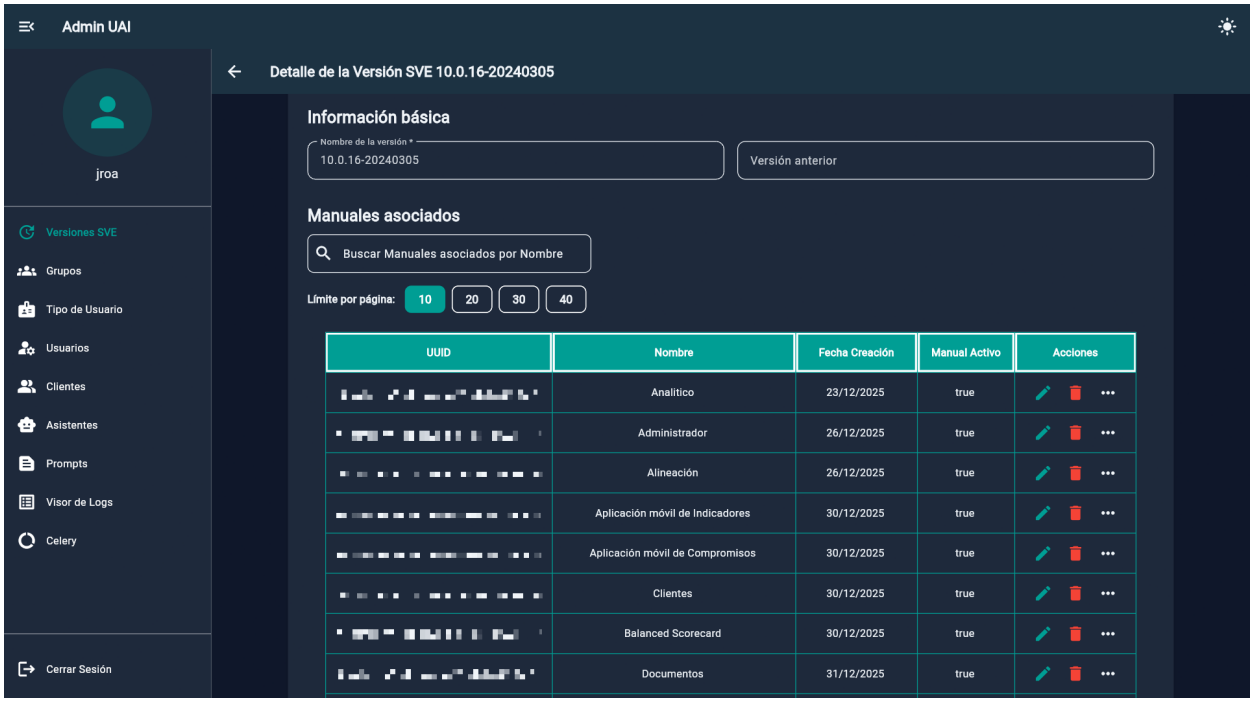
Formulario de creación de versiones de la SVE



Por su parte, la Figura 14 muestra el detalle de una versión específica junto con sus manuales asociados satisfaciendo el requerimiento RF-SVE-03. Esta misma interfaz es ampliada en el proceso de actualización (RF-SVE-04), habilitando los campos editables y reutilizando los componentes de carga y visualización de archivos mostrados en el formulario de creación.

Figura 14

Detalle de una versión de la SVE con manuales asociados

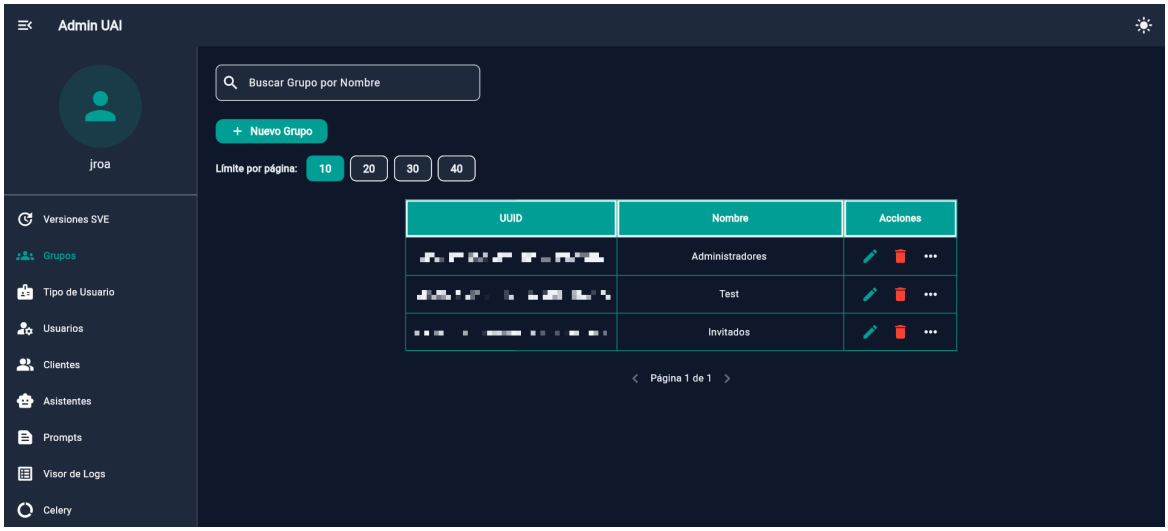


6.3.2.2 Módulo de grupos. Siguiendo el orden de navegación del panel lateral, se presenta el módulo de grupos, el cual permite definir estructuras básicas de acceso dentro del sistema AdminUAI.

En la Figura 15 se presenta el listado de grupos, el cual cumple con el requerimiento RF-GRP-01, incluyendo paginación, búsqueda y acceso a acciones. En esta vista se evidencia además la censura de información sensible como el UUID, de acuerdo con lineamientos internos del equipo de innovación.

Figura 15

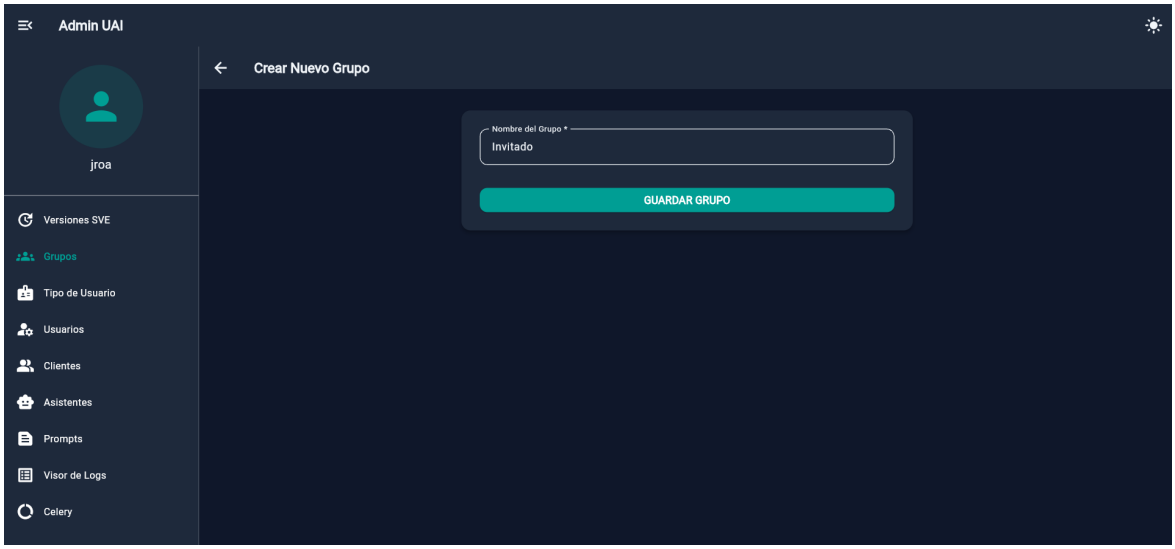
Listado de grupos



La Figura 16 presenta el formulario de creación de grupos, el cual permite registrar nuevos valores para esta entidad, garantizando que no haya múltiples registros con un mismo nombre y dando cumplimiento al requerimiento RF-GRP-02.

Figura 16

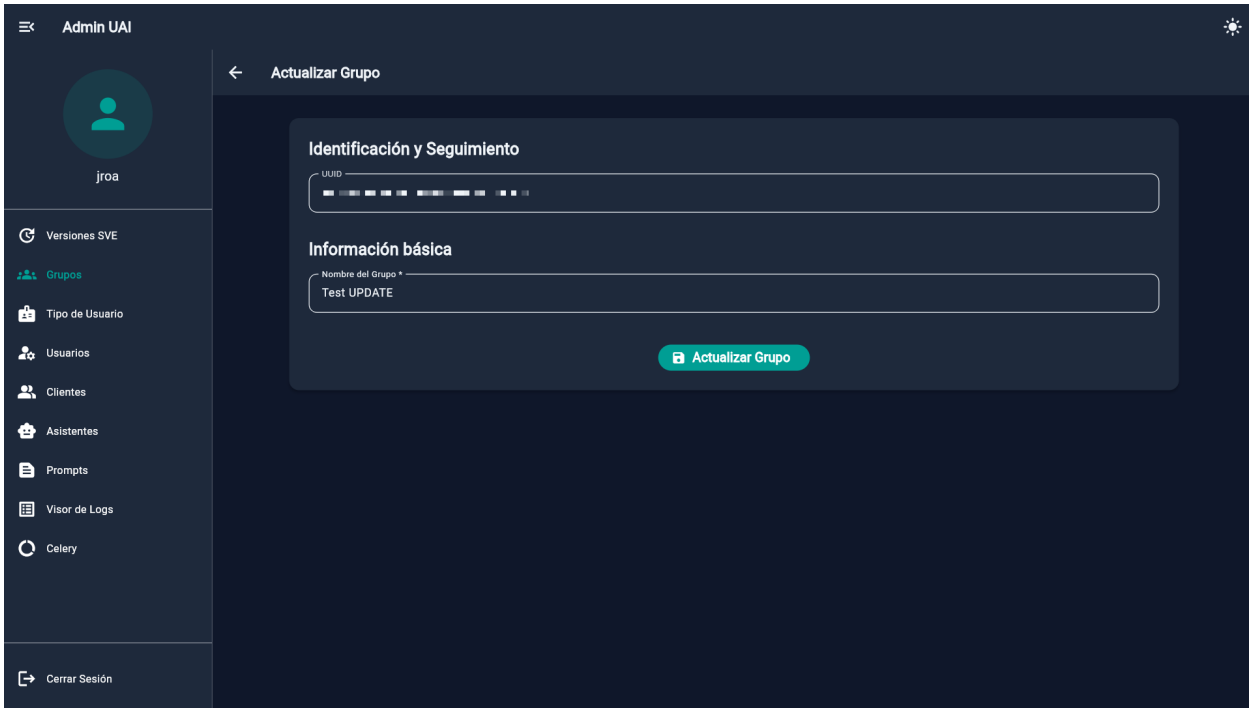
Formulario de creación de un grupo



En la Figura 17 se ilustra la actualización de un grupo, correspondiente al requerimiento RF-GRP-04. Esta vista reutiliza la estructura del detalle (RF-GRP-03), habilitando el campo del nombre. Este patrón de alternancia entre visualización y edición se mantiene de forma consistente en todo el sistema.

Figura 17

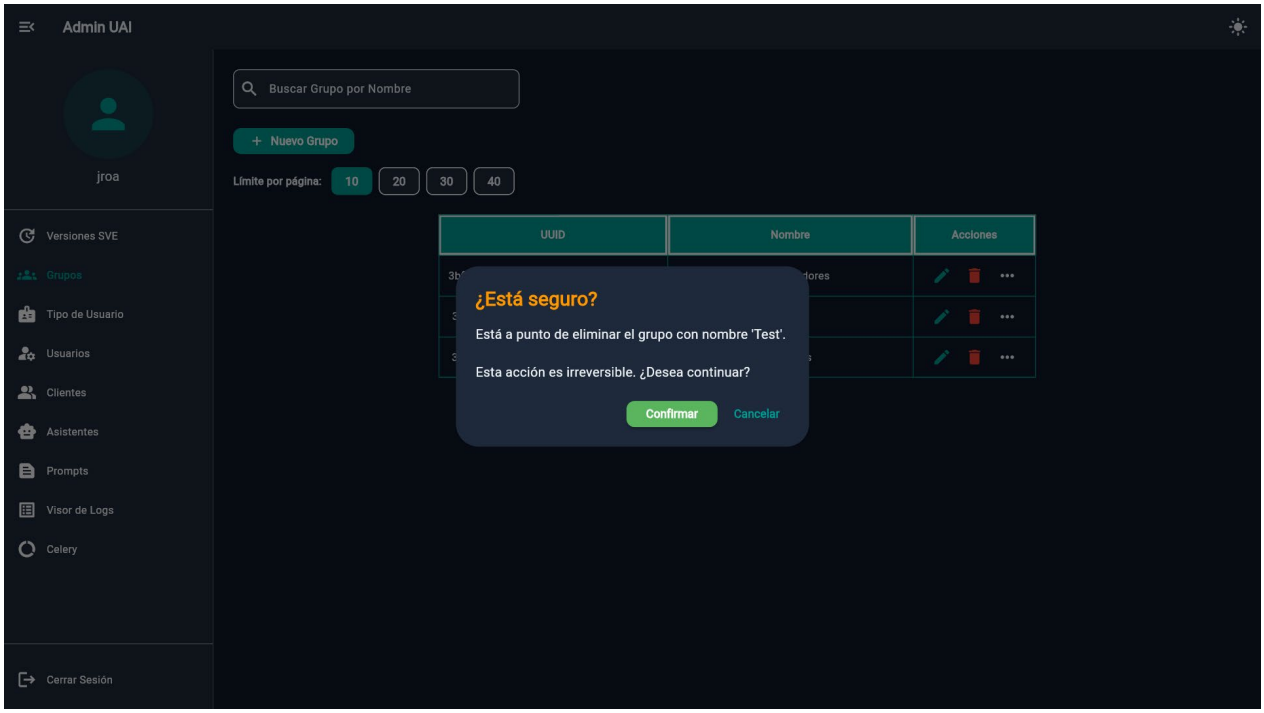
Actualización de un grupo



La Figura 18 presenta el modal de confirmación de eliminación de un grupo, cumpliendo el requerimiento RF-GRP-05 al solicitar validación explícita antes de ejecutar la acción. Este mecanismo de confirmación es transversal en todas las operaciones de eliminación del sistema, garantizando decisiones informadas por parte del usuario y alineándose con el requerimiento RNF-USA-01.

Figura 18

Confirmación de eliminación de grupo

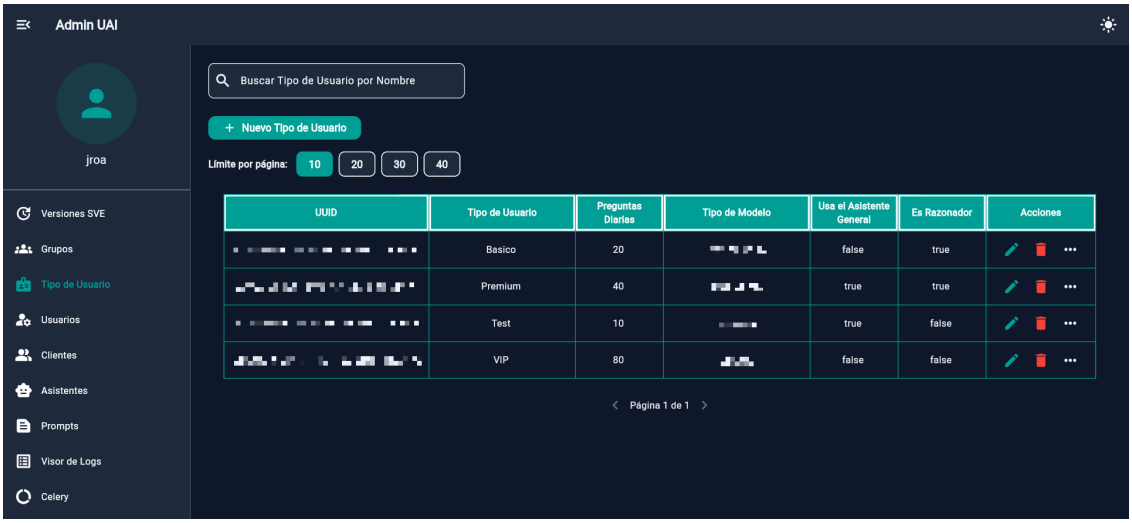


6.3.2.3 Módulo de tipos de usuario. A continuación, se presenta el módulo de tipos de usuario, el cual permite definir configuraciones operativas asociadas al rendimiento y comportamiento de PIA para usuarios específicos.

La Figura 19 muestra el listado de tipos de usuario, dando cumplimiento al requerimiento RF-UST-01, mediante una vista paginada con capacidades de filtrado.

Figura 19

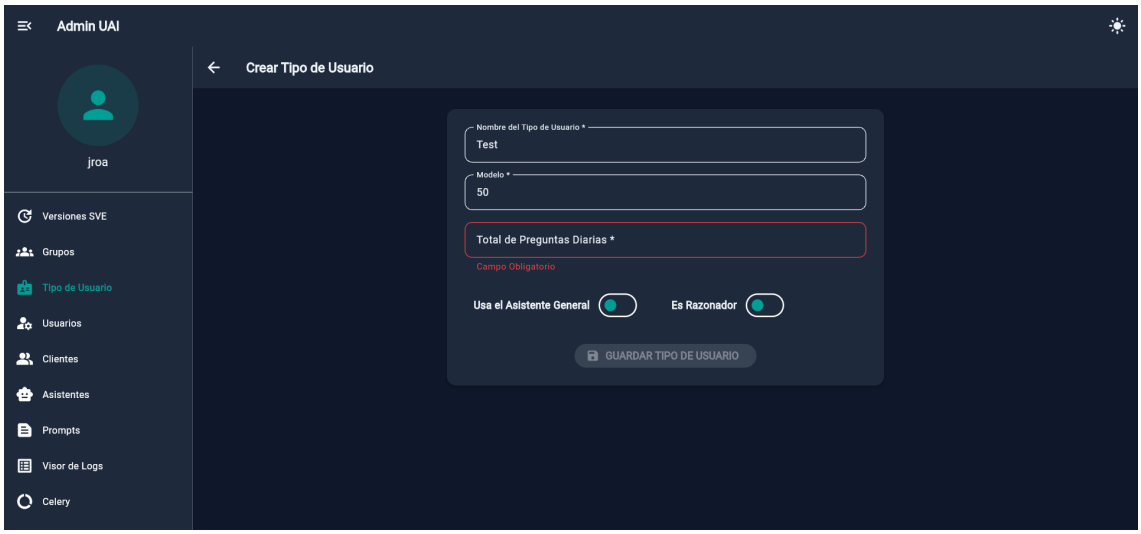
Listado de tipos de usuario



La Figura 20 presenta el formulario de creación de tipos de usuario, el cual implementa el requerimiento RF-UST-02, permitiendo definir parámetros como límites de uso, el modelo LLM que se va a utilizar y validando continuamente la información ingresada.

Figura 20

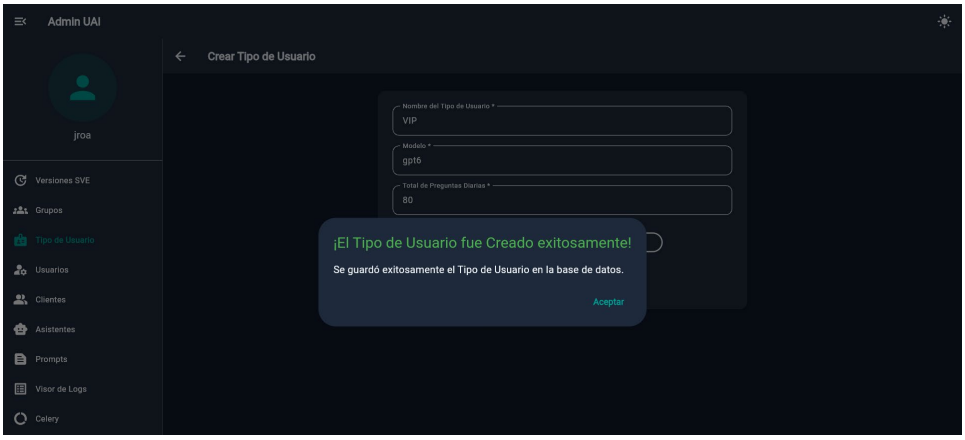
Formulario de creación de tipo de usuario



La Figura 21 evidencia el mensaje de confirmación de creación exitosa, asociado al mismo requerimiento RF-UST-02, proporcionando retroalimentación inmediata al usuario. Este tipo de mensajes se muestran de manera transversal en los diferentes procesos de creación del sistema.

Figura 21

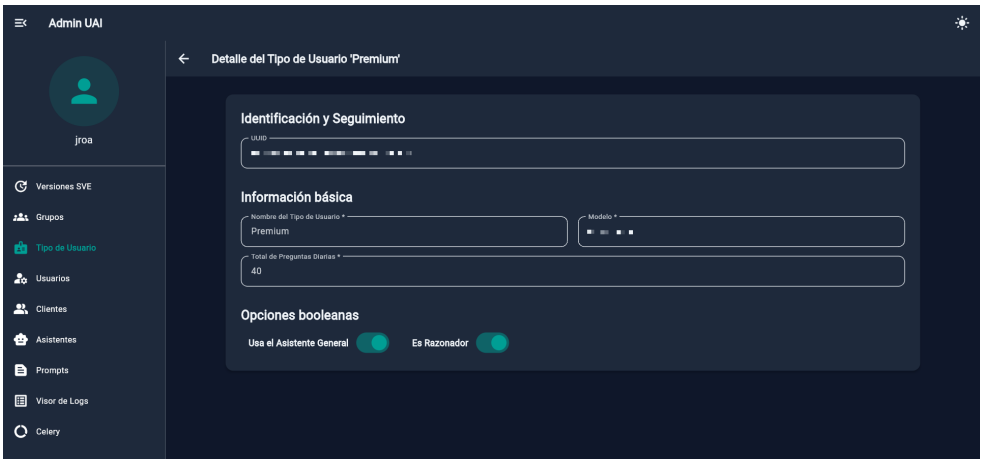
Mensaje de confirmación al crear un tipo de usuario



La Figura 22 corresponde al detalle de un tipo de usuario (RF-UST-03), permitiendo visualizar su configuración en modo de solo lectura.

Figura 22

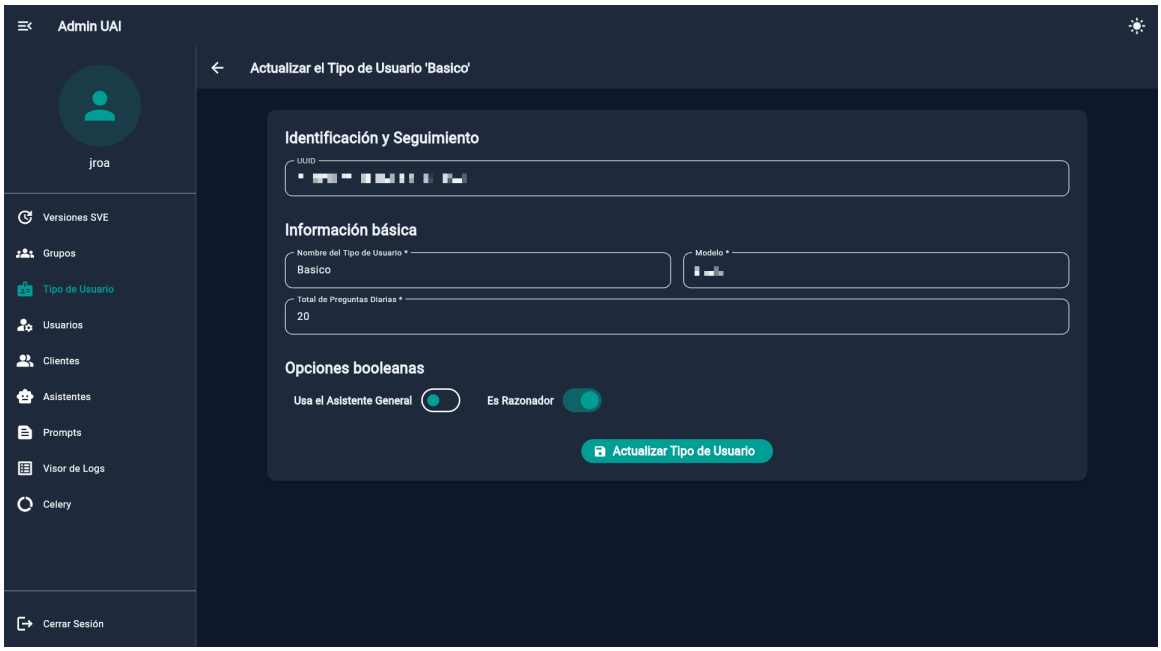
Detalle de tipo de usuario



Finalmente, la Figura 23 presenta la actualización de un tipo de usuario, cumpliendo el requerimiento RF-UST-04 mediante la modificación controlada de sus parámetros.

Figura 23

Actualización de un tipo de usuario

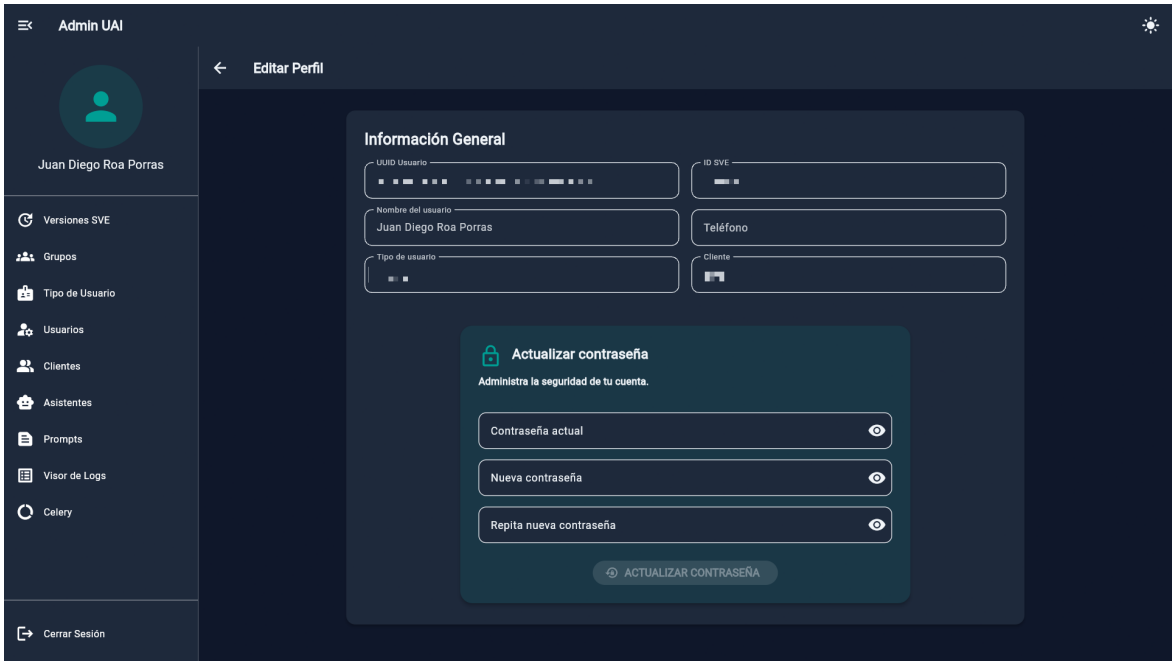


6.3.2.4 Módulo de usuarios. Posteriormente, se presenta el módulo de usuarios, el cual permite gestionar de manera individual el acceso y la configuración de los usuarios que consultan a PIA.

La Figura 24 muestra el perfil del usuario autenticado, accesible desde la parte superior del panel de navegación y cumpliendo el requerimiento RF-USR-03. En esta vista se presenta la información resumida del usuario en modo de solo lectura, junto con un formulario de actualización de contraseña que, para el alcance del proyecto, es únicamente visual.

Figura 24

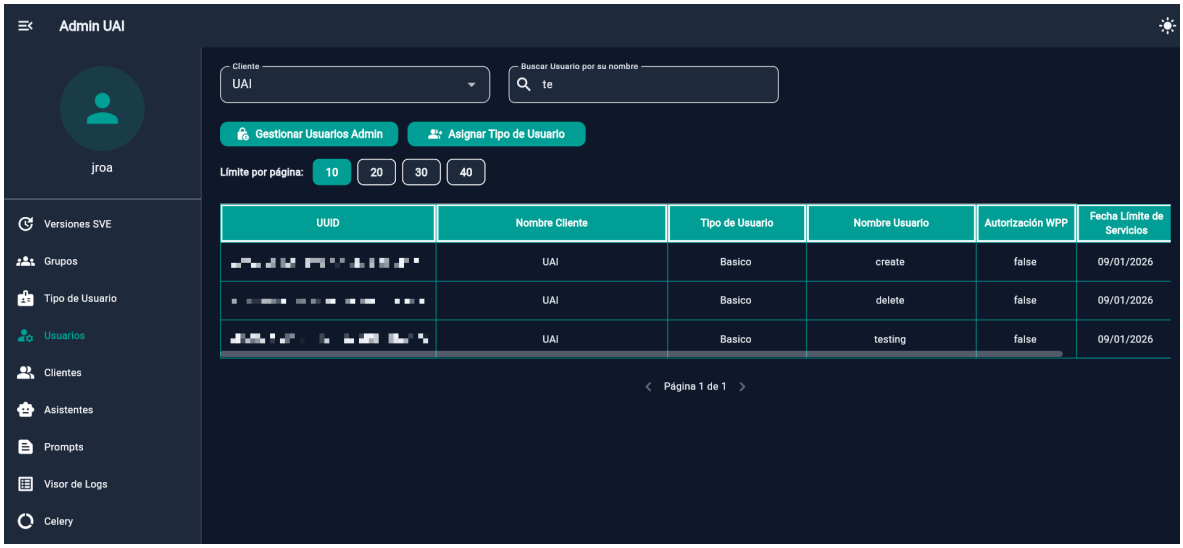
Perfil del usuario en el sistema AdminUAI



La Figura 25 presenta el listado de usuarios, dando cumplimiento al requerimiento RF-USR-01 mediante filtros y visualización paginada.

Figura 25

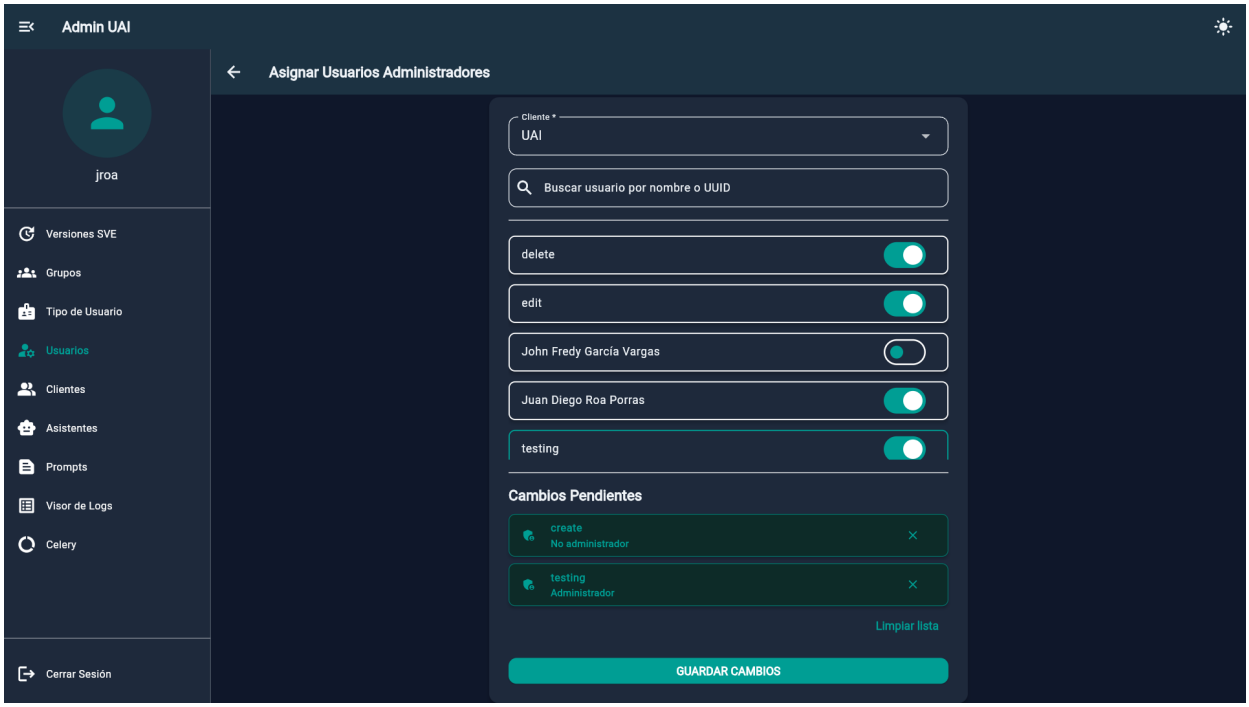
Listado de usuarios



La Figura 26 evidencia la gestión de usuarios administradores, cumpliendo el requerimiento RF-USR-04 al permitir habilitar o revocar accesos de forma controlada.

Figura 26

Gestión de usuarios administradores

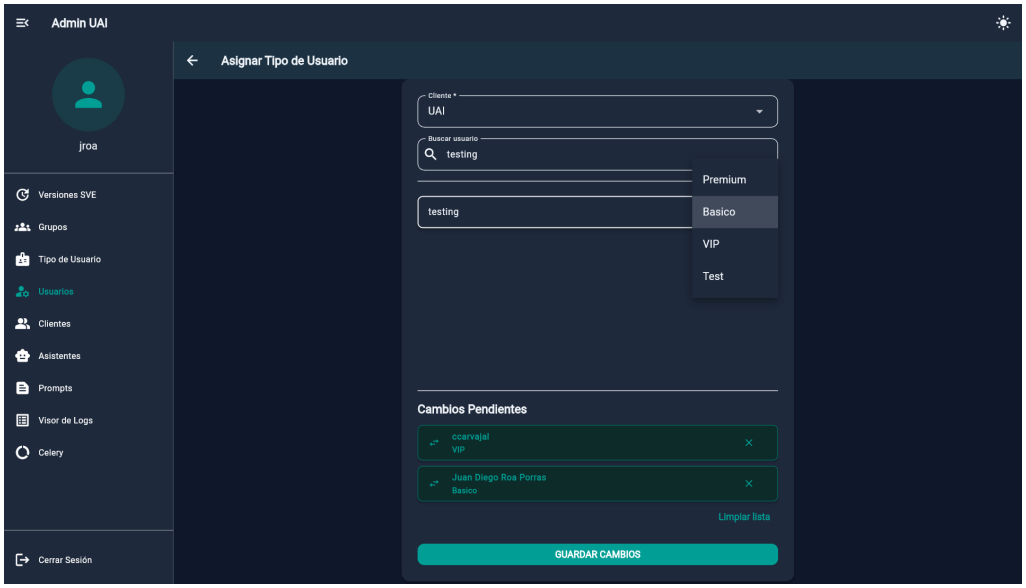


La Figura 27 presenta la asignación de tipos de usuario, cumpliendo el requerimiento RF-USR-05 y manteniendo la misma interfaz de interacción que en la gestión de usuarios administradores, pero cambiando el switch por un desplegable para seleccionar el tipo de usuario correspondiente.

En estas dos vistas cabe resaltar que se pueden dar casos específicos donde el usuario no tenga nombre, para ello se implementó como medida preventiva el uso del UUID del usuario, tanto para el filtrado como para la identificación del usuario al que se le va a hacer la asignación.

Figura 27

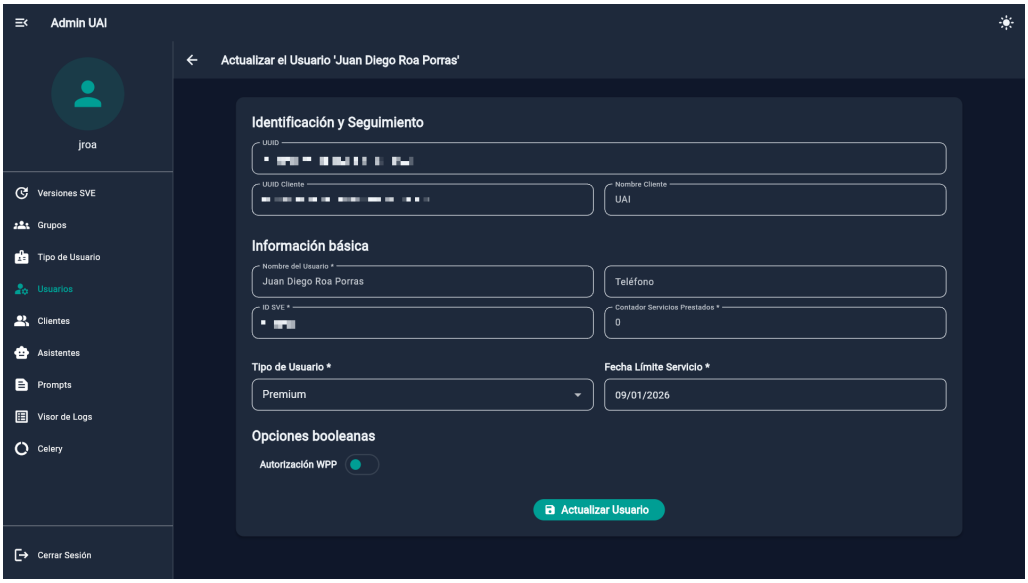
Asignación de tipos de usuario



Finalmente, la Figura 28 muestra la actualización de un usuario, donde, de acuerdo con las reglas de negocio, únicamente es posible modificar el tipo de usuario asignado.

Figura 28

Actualización de un usuario

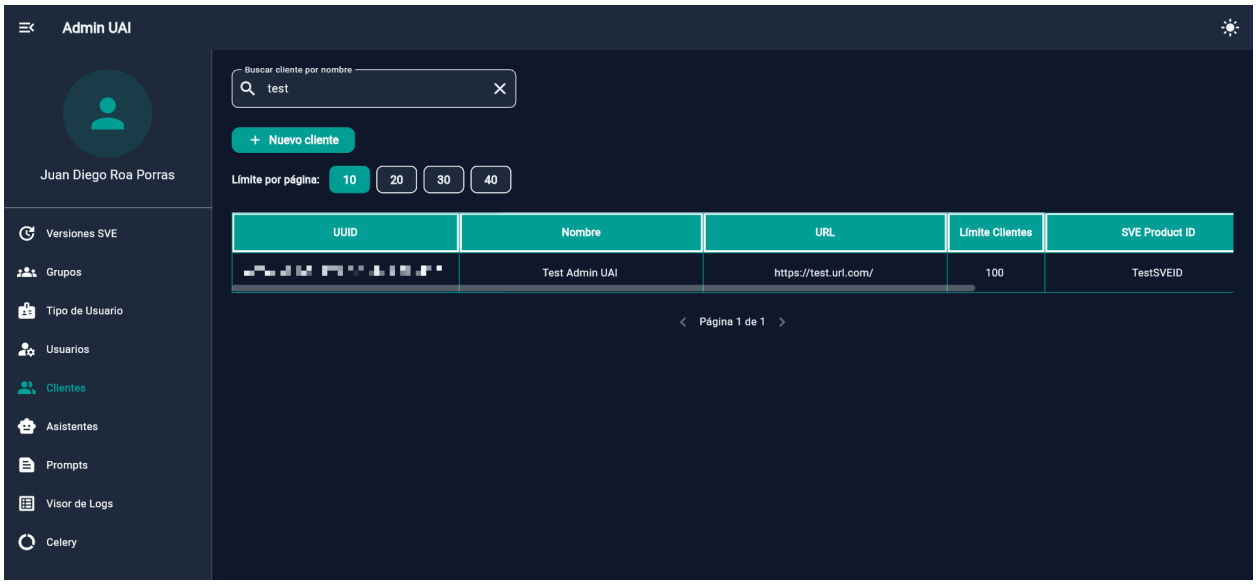


6.3.2.5 Módulo de clientes. Seguidamente, se presenta el módulo de clientes, que constituye uno de los componentes centrales del sistema, ya que define quiénes tienen acceso a PIA.

La Figura 29 presenta el listado de clientes, cumpliendo el requerimiento RF-CLI-01 al permitir su visualización y filtrado.

Figura 29

Listado de clientes



La Figura 30 muestra el formulario de creación de clientes, mediante el cual se da cumplimiento a los requerimientos RF-CLI-02 a RF-CLI-07, permitiendo no solo registrar un cliente, sino también automatizar la creación de asistentes, prompts e índices asociados.

Por su parte, la Figura 31 muestra la vista de actualización de un cliente, correspondiente al requerimiento RF-CLI-09. donde se podrán crear nuevos asistentes para un cliente existente si se requiere.

Figura 30

Formulario de creación de clientes

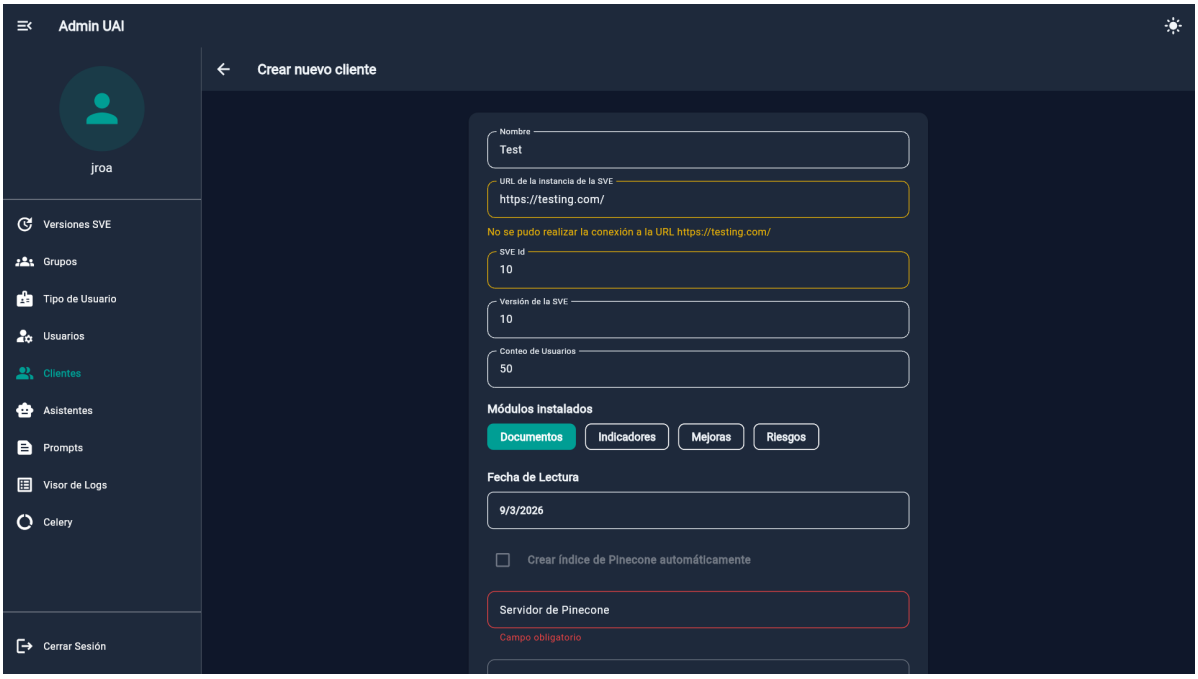
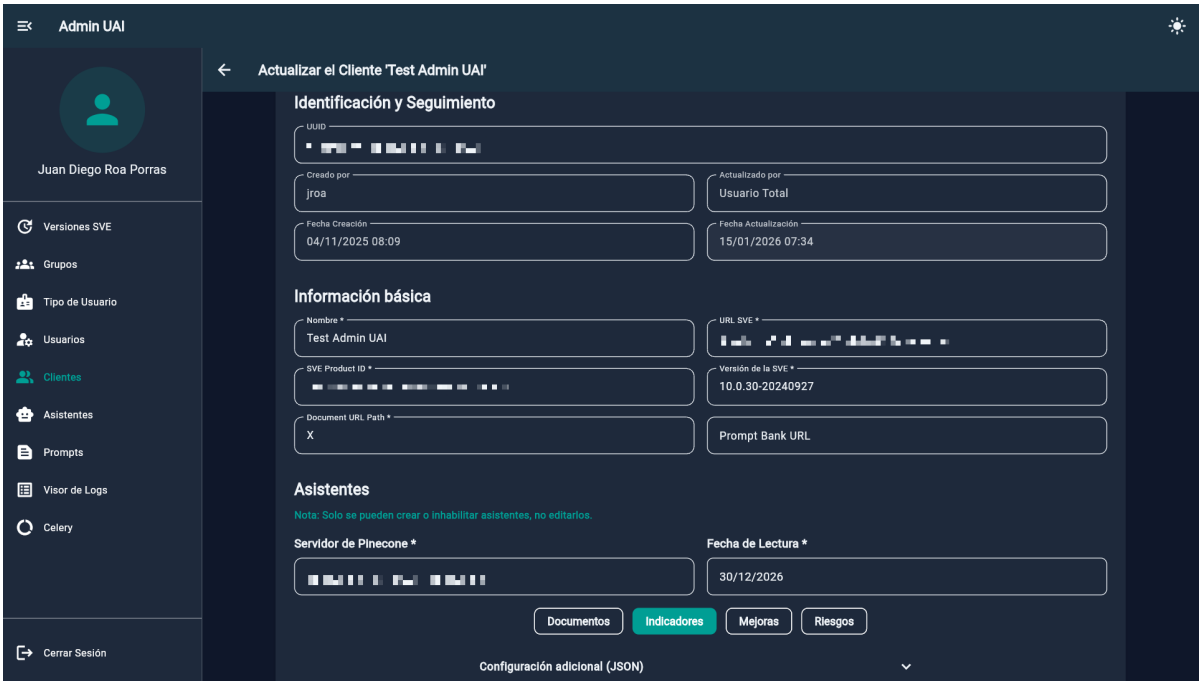


Figura 31

Actualización de un cliente



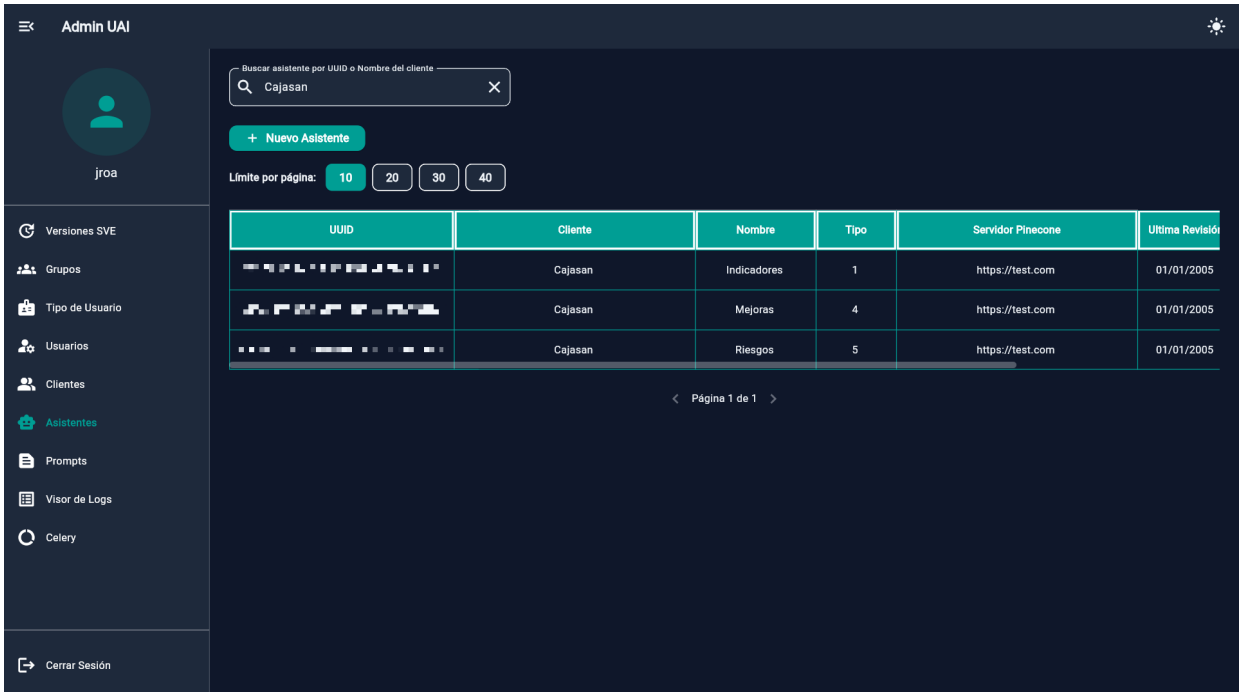
Es importante resaltar que, durante el proceso de creación de un cliente, el sistema ejecuta de manera automática la creación de asistentes y prompts asociados, así como la creación del índice de Pinecone cuando sea necesario. Aunque estos procesos no son visibles directamente en la interfaz, su correcta ejecución se puede evidenciar posteriormente al consultar los registros en los módulos de asistentes y de prompts.

6.3.2.6 Módulo de asistentes. Posteriormente se presenta el módulo de asistentes, el cual permite gestionar de manera personalizada el comportamiento que tendrá PIA para un cliente específico.

La Figura 32 presenta el listado de asistentes, dando cumplimiento al requerimiento RF-AST-01.

Figura 32

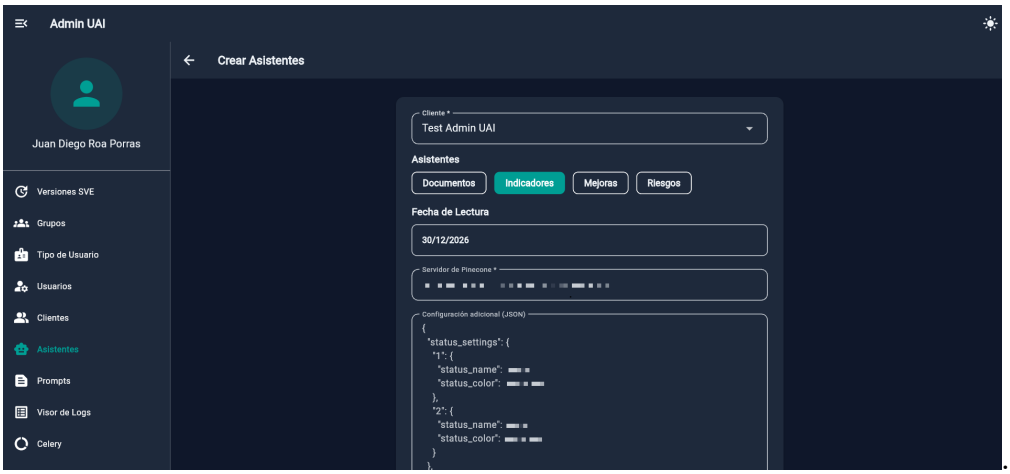
Listado de asistentes



En la Figura 33 se muestra el formulario de creación de asistentes para clientes existentes, correspondiente al requerimiento RF-AST-02, el cual validará que se va a crear por lo menos un nuevo asistente antes de habilitar el botón de guardar asistentes.

Figura 33

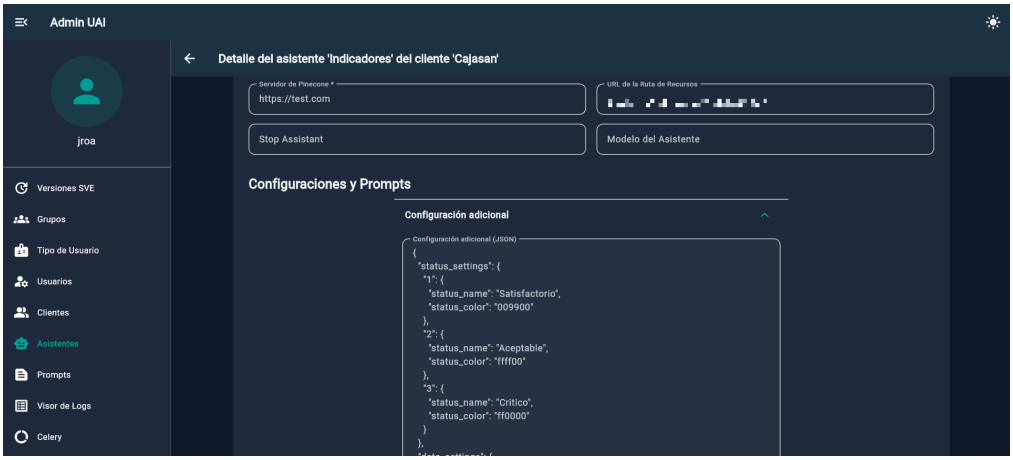
Formulario de creación de asistentes para clientes existentes



En la Figura 34 se observa el detalle de un asistente, cumpliendo el requerimiento RF-AST-03, cabe resaltar que para optimizar el espacio, el usuario podrá desplegar la información de la configuración adicional y otros campos solo cuando la necesite.

Figura 34

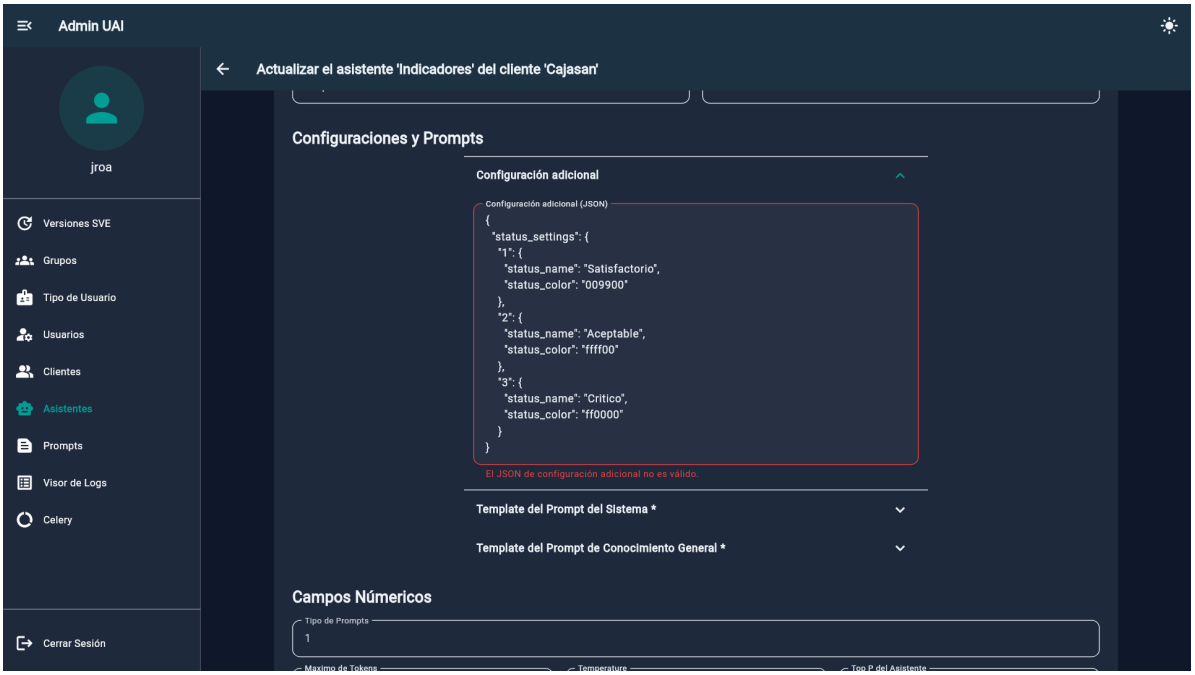
Detalle de un asistente



Finalmente, la Figura 35 presenta la retroalimentación de valor inválido en un formulario, comportamiento transversal que responde al requerimiento RNF-USA-01 y se implementa en todos los formularios del sistema.

Figura 35

Retroalimentación de valor inválido en un formulario

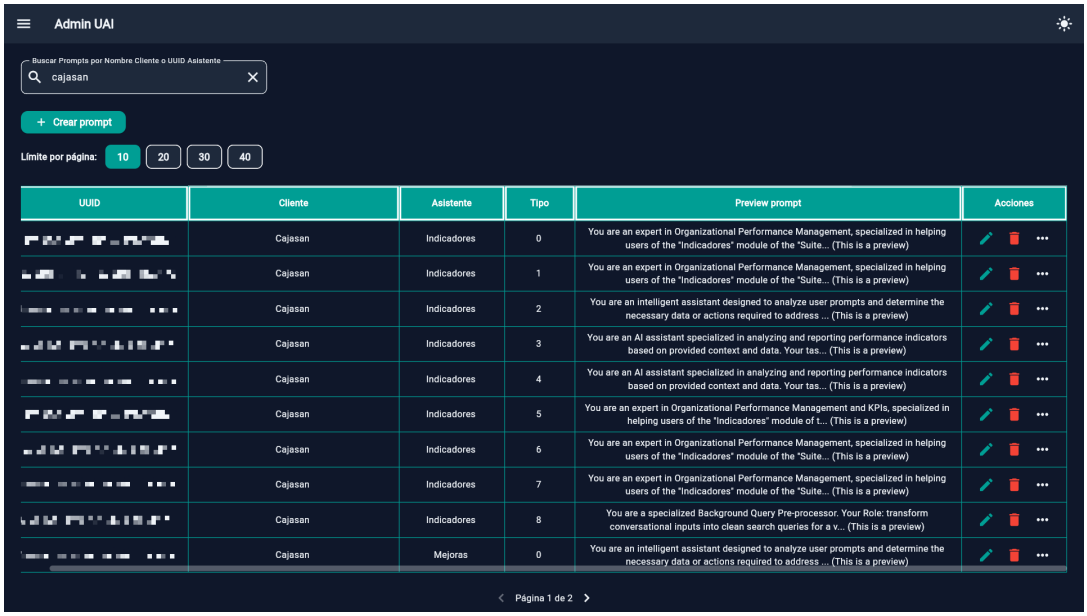


6.3.2.7 Módulo de Prompts. Siguiendo el flujo de navegación del sistema, se presenta el módulo de prompts, encargado de establecer los lineamientos básicos que rigen el comportamiento de cada asistente. De este modo, incluso cuando el usuario proporcione instrucciones sin contexto, PIA mantiene una conducta uniforme y un estándar mínimo de calidad.

La Figura 36 muestra el listado de prompts, dando cumplimiento al requerimiento RF-PRO-01 y donde se destaca la columna de previsualización del prompt para reducir la carga sobre la API al hacer consultas que retornen múltiples resultados.

Figura 36

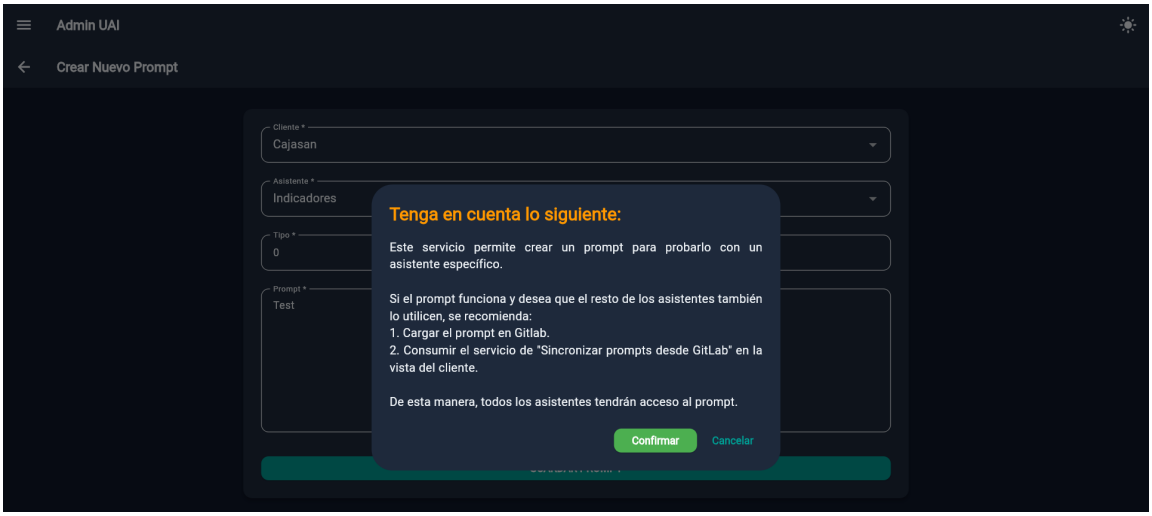
Listado de prompts



La Figura 37 presenta un modal informativo asociado a la gestión de prompts, el cual se utiliza en operaciones de creación, actualización y sincronización, alineándose con los requerimientos RF-PRO-02 a RF-PRO-04.

Figura 37

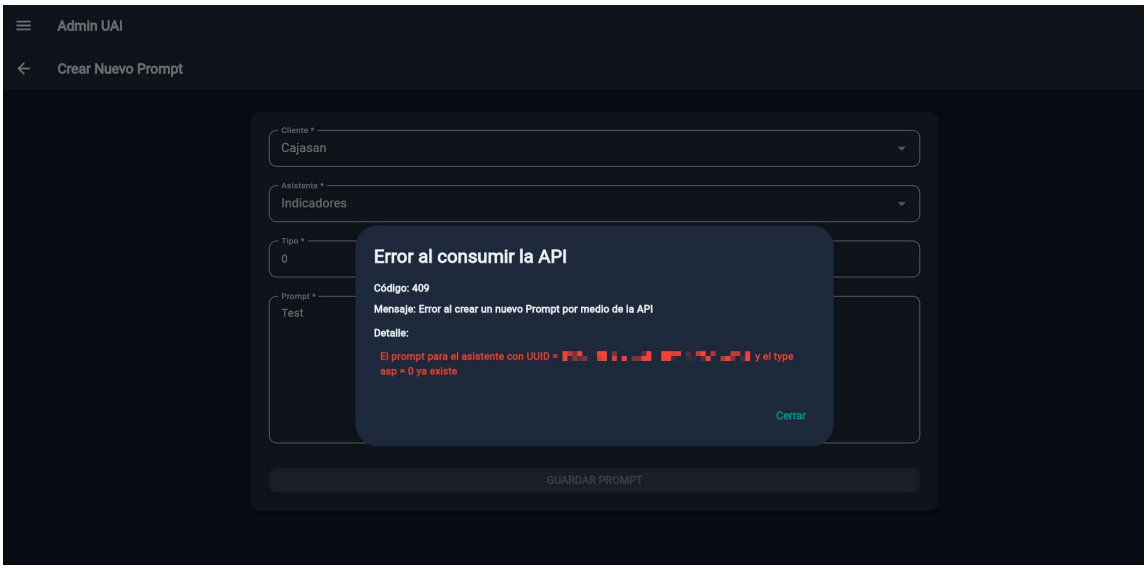
Modal informativo de gestión de prompts



Por su parte, la Figura 38 evidencia un modal informativo de error, utilizado cuando no es posible ejecutar una operación, comportamiento transversal que responde tanto a los requerimientos funcionales como al RNF-FIA-01 y RNF-USA-01.

Figura 38

Modal informativo de error



6.3.2.8 Visor de logs. Finalmente, se presenta el módulo de visor de logs, el cual permite consultar los registros de funcionamiento de los servicios del ecosistema tecnológico de PIA.

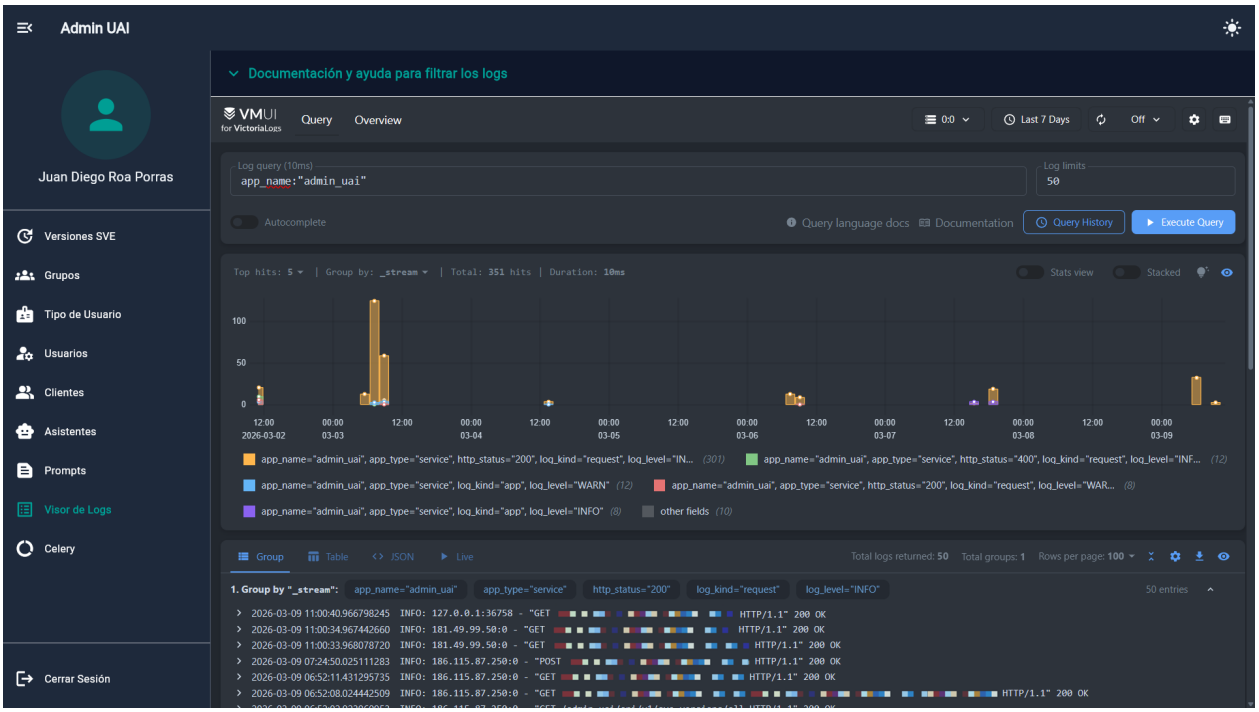
La Figura 39 muestra el visor de logs embebido en AdminUAI, dando cumplimiento al requerimiento RF-LOG-01, permitiendo visualizar la información de manera centralizada.

Es importante destacar que durante la implementación de esta funcionalidad, se identificó que la integración del visor requería exponerlo mediante una ruta específica en el servidor de Nginx. No obstante, esta decisión implicaba un posible riesgo de seguridad, ya que las herramientas utilizadas para la gestión de logs no requieren autenticación por defecto, lo que

permitiría el acceso directo a la información proporcionada por el visor mediante el conocimiento de la URL correspondiente.

Figura 39

Visor de logs embebido en AdminUAI

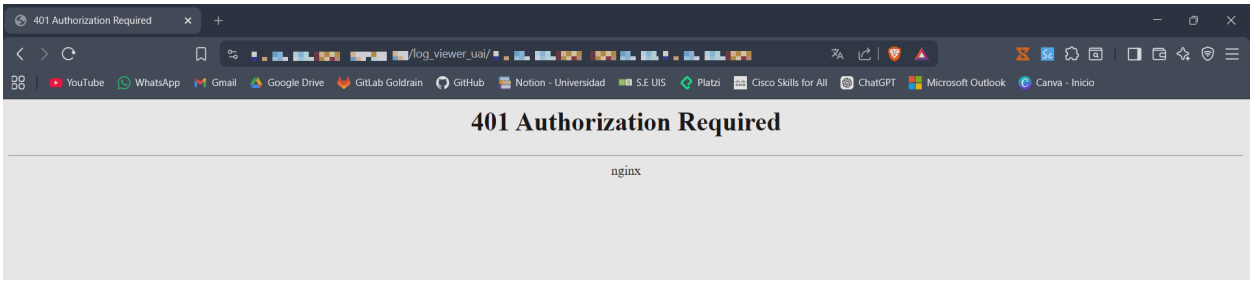


Con el fin de mitigar este riesgo, se implementó un mecanismo de validación basado en una cookie que contiene el token JWT del usuario autenticado. De esta manera, el acceso al visor de logs queda condicionado a la existencia de una sesión válida dentro del sistema, en concordancia con el requerimiento RNF-SEC-01.

La Figura 40 muestra el intento de acceso directo al visor de logs sin una cookie válida, evidenciando la respuesta de acceso no autorizado (401), lo cual confirma la correcta implementación de este mecanismo de seguridad.

Figura 40

Acceso no autorizado al visor de logs



En general, las interfaces presentadas en esta subsección constituyen la evidencia visual de la implementación de los requerimientos funcionales definidos en la sección 6.1, permitiendo apreciar el comportamiento del sistema AdminUAI en los diferentes módulos desarrollados. Asimismo, se destacan los mecanismos transversales de validación, retroalimentación y seguridad, los cuales contribuyen a la consistencia y usabilidad de la solución.

En la siguiente sección se presentan las pruebas unitarias y funcionales realizadas, con el fin de verificar el correcto funcionamiento del sistema tanto en un entorno controlado como en el ambiente de despliegue definido para el proyecto.

6.4 Validación de la solución

De acuerdo con lo planteado en la metodología, específicamente en la sección 5.1 y en la fase de implementación y validación, se realizó el desarrollo del sistema siguiendo un enfoque incremental. Bajo este enfoque, se definió como objetivo la entrega de un producto mínimo viable (MVP) al finalizar cada incremento de desarrollo, permitiendo contar desde etapas tempranas con una solución funcional que aportara valor al proceso de administración de PIA.

En este contexto, cada ciclo de desarrollo se orientó a la implementación incremental de funcionalidades específicas. El primer incremento estuvo enfocado en la creación de clientes, asistentes y prompts de manera encadenada; el segundo en la integración con la SVE para la obtención de información de licencias; el tercero en la interacción con la API de Pinecone y la gestión automatizada de índices; el cuarto en la implementación del mecanismo de autenticación mediante un token JWT; el quinto en la sincronización de prompts, y así sucesivamente hasta finalizar la implementación de todas las funcionalidades acordadas.

Un aspecto relevante dentro del proceso de validación es que, desde el primer incremento, el sistema fue desplegado en el servidor de pruebas del equipo de innovación. Inicialmente, este despliegue contó con el acompañamiento de un miembro del equipo; sin embargo, en las siguientes iteraciones, el proceso de actualización, despliegue y corrección de errores fue realizado de manera autónoma, permitiendo un ciclo continuo de mejora sobre el entorno de pruebas.

Este enfoque permitió validar la solución de forma continua a lo largo del desarrollo. En términos generales, la validación se organizó en cuatro etapas. En primer lugar, durante la implementación de cada funcionalidad, se realizaron pruebas unitarias para verificar el comportamiento individual de los componentes. Posteriormente, previo al cierre de cada incremento, se efectuaba el despliegue en el entorno de pruebas y se validaba nuevamente el funcionamiento del sistema de manera integral.

En una tercera etapa, al finalizar cada incremento, se realizaban sesiones de revisión con el tutor del proyecto y el equipo de innovación, en las cuales se presentaban los avances alcanzados. Durante estas sesiones, el sistema era probado en tiempo real, validando no solo el

cumplimiento de las funcionalidades implementadas, sino también aspectos relacionados con la usabilidad e intuitividad de las interfaces.

Finalmente, se llevó a cabo una validación en un contexto de uso real, donde el sistema fue utilizado para la ejecución de tareas propias del equipo, como la creación de clientes. Esta etapa permitió identificar algunos ajustes necesarios, tales como la corrección de parámetros de configuración, la actualización de estructuras JSON por defecto utilizadas por los asistentes y pequeñas mejoras en el manejo de información sensible. No obstante, gracias a la claridad de los requerimientos definidos y al seguimiento continuo mediante reuniones diarias, se logró completar la primera versión del prototipo sin contratiempos relevantes.

Una vez descrito el proceso de validación, a continuación, se presentan las pruebas realizadas.

6.4.1 Pruebas unitarias

Siguiendo los lineamientos establecidos por el equipo de innovación, se emplearon dos herramientas principales para la validación del código desarrollado.

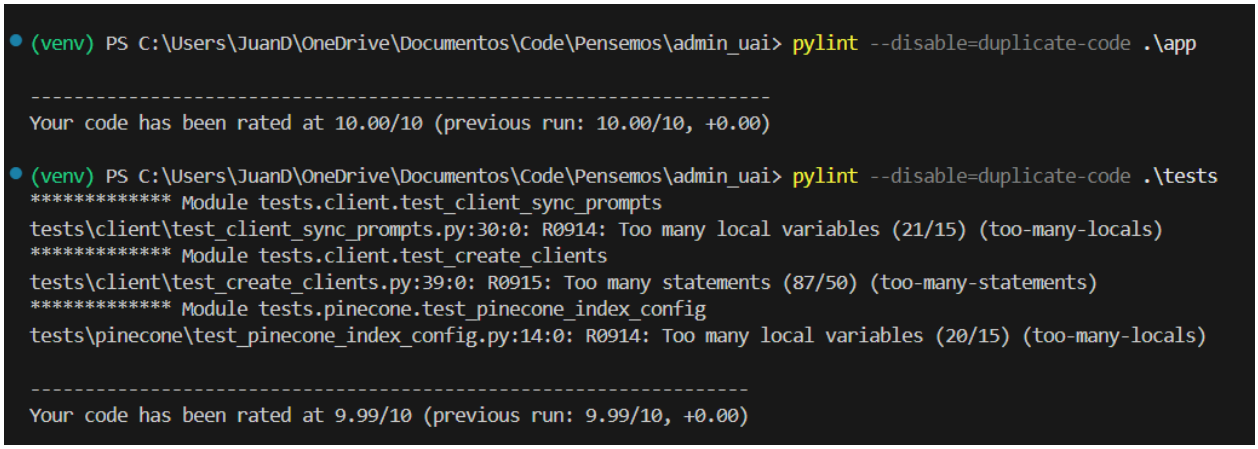
En primer lugar, se utilizó Pylint como herramienta de análisis estático, con el propósito de evaluar la calidad del código en aspectos como convenciones de nombrado, complejidad de funciones, número de variables locales y organización general del código. Este análisis se ejecutó de manera recurrente durante el desarrollo mediante el comando:

```
pylint --disable=duplicate-code .\app
```

El objetivo fue mantener una calificación cercana a 10, asegurando un código legible, mantenible y alineado con buenas prácticas de desarrollo de software.

Figura 41

Evaluación de calidad del código mediante Pylint



En la figura 41 se observa que el código del sistema alcanza una calificación de 10.00/10 en el módulo principal, mientras que en el conjunto de pruebas se identifican advertencias relacionadas principalmente con el número de variables locales y la cantidad de sentencias por función. Estas observaciones son consistentes con la naturaleza de las pruebas unitarias, donde es común agrupar múltiples escenarios en un mismo caso de prueba para garantizar cobertura.

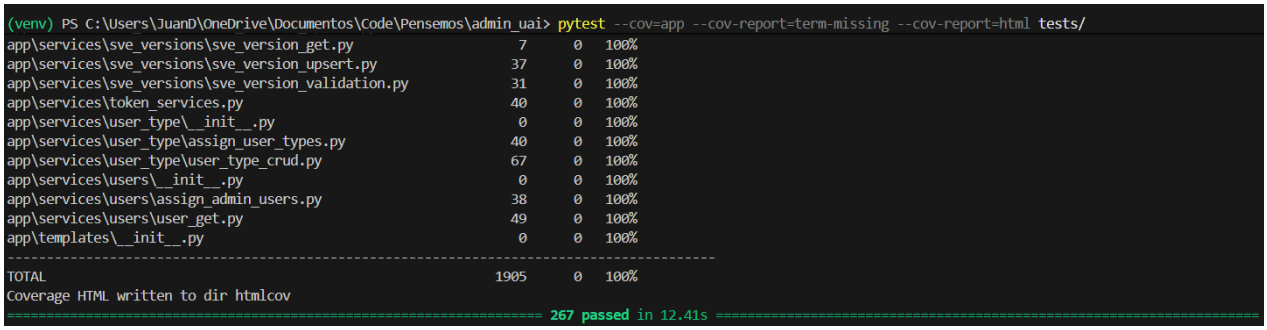
En segundo lugar, se utilizó Pytest como framework principal para la ejecución de pruebas unitarias, empleando el siguiente comando:

```
pytest --cov=app --cov-report=term-missing --cov-report=html tests/
```

Este enfoque permitió no solo validar el comportamiento individual de las funciones, sino también medir el nivel de cobertura del código, siguiendo el lineamiento del equipo de innovación de alcanzar un 100% de coverage.

Figura 42

Ejecución de pruebas unitarias mediante Pytest



Como se evidencia en la figura 42, se ejecutaron un total de 267 pruebas unitarias en un tiempo aproximado de 12 segundos, alcanzando un 100% de cobertura sobre el código del aplicativo. Este resultado se logró mediante el uso de técnicas como el mockeo de servicios externos, evitando dependencias directas con APIs como Pinecone durante la ejecución de las pruebas, lo cual permite mantener tiempos de ejecución bajos y resultados consistentes.

6.4.2 Pruebas funcionales

Las pruebas funcionales se realizaron de manera iterativa a lo largo del desarrollo, ejecutándose al menos una vez por incremento en diferentes momentos del proceso. En primer lugar, eran realizadas por el desarrollador al finalizar la implementación de cada funcionalidad; posteriormente, eran validadas por el tutor en el cierre del ciclo de desarrollo; y finalmente, se reforzaban en escenarios de uso real por parte del equipo de innovación.

Este proceso permitió validar que el sistema cumpliera con los requerimientos funcionales definidos en la sección 6.1, verificando el comportamiento de las diferentes operaciones bajo condiciones reales de uso.

En la Tabla 11 se presenta un resumen de las pruebas funcionales ejecutadas, organizadas por módulo y asociadas a sus respectivos requerimientos funcionales. El detalle completo de los casos de prueba, incluyendo pasos, resultados esperados y observaciones, se encuentra documentado en el **Apéndice D** del presente documento.

Tabla 12

Pruebas funcionales asociadas a los requerimientos del sistema

Identificador	Módulo	Requerimientos Evaluados	Descripción del caso de prueba	Resultado
PF-SEC-01	Seguridad	RF-SEC-01	Acceso al sistema mediante token JWT válido	Exitosa
PF-SEC-02	Seguridad	RF-SEC-02	Validación continua del token JWT	Exitosa
PF-SEC-03	Seguridad	RF-SEC-03	Cierre de sesión local del usuario	Exitosa
PF-CLI-01	Clientes	RF-CLI-01	Listado de clientes registrados	Exitosa
PF-CLI-02	Clientes	RF-CLI-02, RF-CLI-05, RF-CLI-06, RF-CLI-07	Crear cliente de la SVE y procesos automáticos	Exitosa

PF-CLI-03	Clientes	RF-CLI-03, RF-CLI-04	Validación de conexión e info de licencia SVE	Exitosa
PF-CLI-04	Clientes	RF-CLI-08	Consultar detalle de un cliente	Exitosa
PF-CLI-05	Clientes	RF-CLI-09	Actualizar información de un cliente	Exitosa
PF-CLI-06	Clientes	RF-CLI-10	Sincronizar prompts de un cliente con GitLab	Exitosa
PF-AST-01	Asistentes	RF-AST-01	Listado de asistentes registrados	Exitosa
PF-AST-02	Asistentes	RF-AST-02	Crear asistentes para un cliente existente	Exitosa
PF-AST-03	Asistentes	RF-AST-03	Consultar detalle de un asistente	Exitosa
PF-AST-04	Asistentes	RF-AST-04	Actualizar información de un asistente	Exitosa

PF-AST-05	Asistentes	RF-AST-05	Sincronizar prompts de un asistente específico	Exitosa
PF-PRO-01	Prompts	RF-PRO-01	Listado de prompts registrados	Exitosa
PF-PRO-02	Prompts	RF-PRO-02	Crear un prompt individual	Exitosa
PF-PRO-03	Prompts	RF-PRO-03	Consultar detalle de un prompt	Exitosa
PF-PRO-04	Prompts	RF-PRO-04	Actualizar la información de un prompt	Exitosa
PF-PRO-05	Prompts	RF-PRO-05	Eliminar un prompt del sistema	Exitosa
PF-SVE-01	Versiones de la SVE	RF-SVE-01	Listado de versiones de la SVE	Exitosa
PF-SVE-02	Versiones de la SVE	RF-SVE-02	Crear una versión de la SVE y cargar manuales	Exitosa

PF-SVE-03	Versiones de la SVE	RF-SVE-03	Consultar detalle de una versión de la SVE	Exitosa
PF-SVE-04	Versiones de la SVE	RF-SVE-04	Actualizar los datos y manuales de una versión	Exitosa
PF-GRP-01	Grupos	RF-GRP-01	Listado de grupos registrados	Exitosa
PF-GRP-02	Grupos	RF-GRP-02	Crear un grupo de permisos	Exitosa
PF-GRP-03	Grupos	RF-GRP-03	Consultar detalle de un grupo	Exitosa
PF-GRP-04	Grupos	RF-GRP-04	Actualizar información de un grupo	Exitosa
PF-GRP-05	Grupos	RF-GRP-05	Eliminar un grupo del sistema	Exitosa
PF-UST-01	Tipos de Usuario	RF-UST-01	Listado de tipos de usuario	Exitosa

PF-UST-02	Tipos de Usuario	RF-UST-02	Crear un nuevo tipo de usuario	Exitosa
PF-UST-03	Tipos de Usuario	RF-UST-03	Consultar detalle de un tipo de usuario	Exitosa
PF-UST-04	Tipos de Usuario	RF-UST-04	Actualizar información de un tipo de usuario	Exitosa
PF-UST-05	Tipos de Usuario	RF-UST-05	Eliminar un tipo de usuario huérfano	Exitosa
PF-USR-01	Usuarios	RF-USR-01	Listado de usuarios registrados	Exitosa
PF-USR-02	Usuarios	RF-USR-02	Consultar detalle de un usuario	Exitosa
PF-USR-03	Usuarios	RF-USR-03	Consultar perfil del usuario autenticado	Exitosa
PF-USR-04	Usuarios	RF-USR-04	Gestionar usuarios con acceso a AdminUAI	Exitosa

PF-USR-05	Usuarios	RF-USR-05	Gestionar asignación de tipos de usuario	Exitosa
PF-LOG-01	Logs	RF-LOG-01	Filtrar y buscar registros en el visor de logs	Exitosa

De manera global, se ejecutaron 39 casos de prueba funcionales, cubriendo la totalidad de los módulos implementados. En todos los casos documentados, las pruebas fueron exitosas, validando el cumplimiento de los requerimientos funcionales definidos para el sistema.

En general, los resultados obtenidos evidencian que el sistema cumple con los requerimientos establecidos, tanto a nivel de componentes individuales como en su operación dentro del entorno de pruebas.

Asimismo, el proceso iterativo de validación permitió identificar y corregir oportunamente ajustes menores, fortaleciendo la estabilidad y confiabilidad del prototipo antes de su uso en escenarios reales.

A continuación, en la sección 6.5, se presentan las evidencias correspondientes al proceso de capacitación realizado con el equipo de innovación de Pensemos S.A., en cumplimiento del último objetivo específico del proyecto.

6.5 Capacitación del equipo de innovación en el uso de AdminUAI

Como se evidenció en la sección anterior, la validación del sistema se llevó a cabo de manera paralela al desarrollo del prototipo, involucrando activamente al equipo de innovación de Pensemos S.A. durante los distintos ciclos de desarrollo. En este contexto, la capacitación en el uso del sistema no se limitó a sesiones formales, sino que se desarrolló de manera continua a lo largo del proyecto. En la práctica, cada vez que se identificaban inconsistencias o imperfectos en el sistema, estos eran reportados para su respectivo ajuste, y cuando surgían dudas sobre el funcionamiento de alguna funcionalidad, se brindaba la explicación correspondiente, permitiendo así una apropiación progresiva del sistema por parte del equipo.

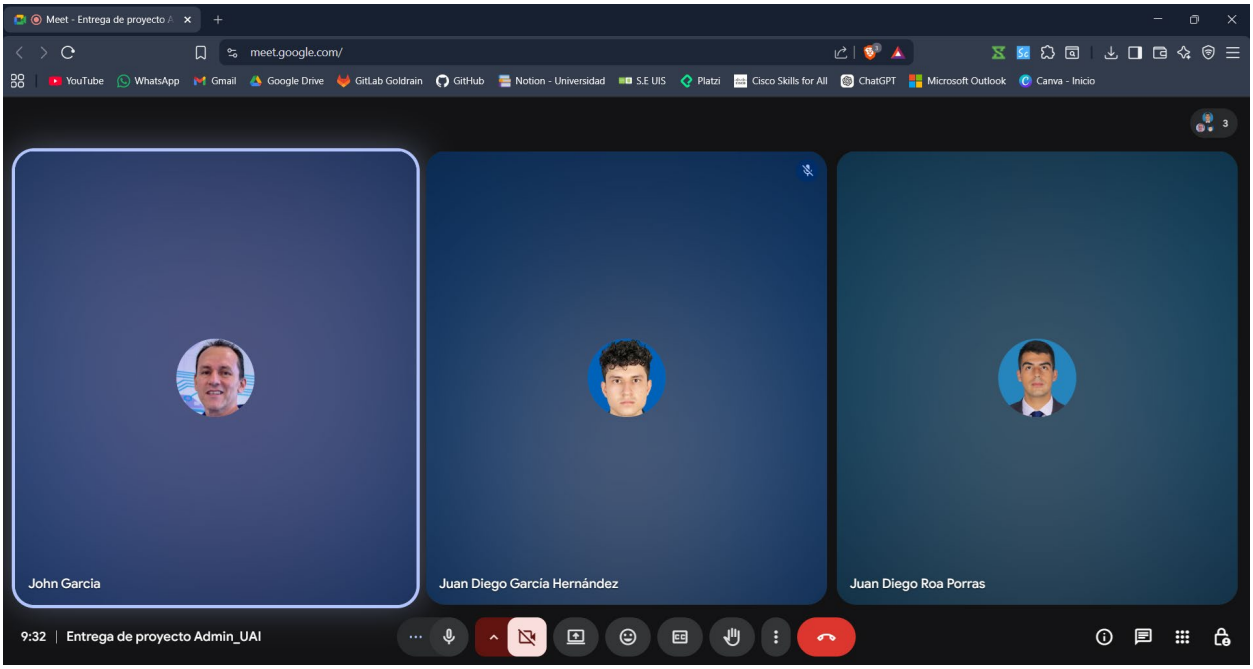
Aun así, y siguiendo las políticas de trabajo de la organización, las capacitaciones formales se realizaron de manera remota. Inicialmente, y con el fin de no interferir con otros proyectos internos que se ejecutaban en paralelo, se acordó realizar una única sesión de capacitación general dirigida a todo el equipo de innovación, conformado por John García, Juan Diego García, Juan Diego Roa y Camilo Andrés Carvajal. Esta sesión se llevó a cabo el día 3 de marzo del presente año, en un espacio de hora y media comprendido entre las 8:30 a.m. y las 10:00 a.m.

Durante esta primera sesión se presentó un recuento general del alcance del proyecto, así como de las funcionalidades implementadas con base en la especificación de requerimientos funcionales y no funcionales, previamente validadas mediante el plan de pruebas descrito en la sección 6.4. Asimismo, se realizó una demostración del sistema en funcionamiento, permitiendo evidenciar el flujo completo de las principales operaciones administrativas soportadas por la aplicación.

La Figura 43 presenta la evidencia correspondiente a esta primera sesión de capacitación, en la cual participaron tres integrantes del equipo de innovación.

Figura 43

Sesión de capacitación inicial del sistema AdminUAI



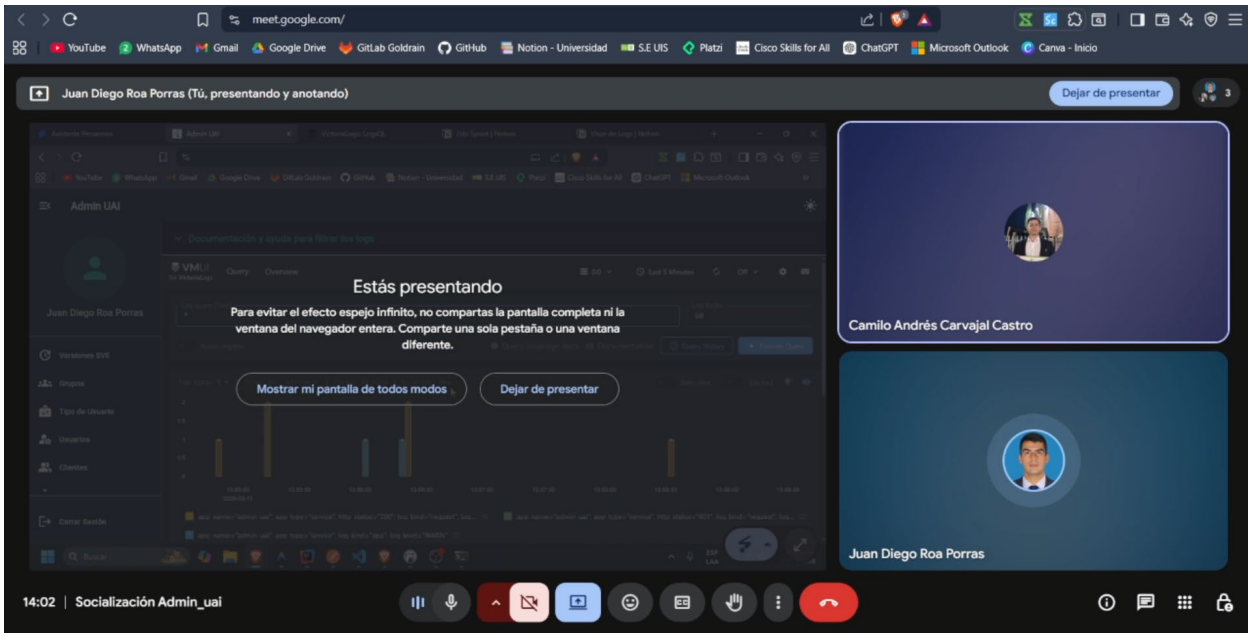
Posteriormente, debido a que uno de los integrantes no pudo asistir a la sesión inicial, se programó una segunda sesión de capacitación el día 11 de marzo del presente año, en un espacio de 1:00 p.m. a 2:30 p.m., con el objetivo de garantizar que todos los miembros del equipo contaran con el contexto necesario para el uso y evolución del sistema.

En esta segunda sesión se abordaron nuevamente las funcionalidades principales del sistema, haciendo mayor énfasis en aspectos técnicos relacionados con el despliegue, particularidades de implementación y posibles escenarios de uso. De igual manera, se abrió un espacio para la retroalimentación, en el cual se recibieron sugerencias de mejora orientadas principalmente a la flexibilidad de las vistas del sistema.

La Figura 44 presenta la evidencia de la segunda sesión de capacitación realizada con los miembros restantes del equipo.

Figura 44

Sesión de capacitación complementaria del sistema AdminUAI



Es importante destacar que, dado que el equipo de innovación de Pensemos S.A. está conformado por desarrolladores con amplio conocimiento tanto del ecosistema tecnológico de PIA como de las tecnologías utilizadas en la implementación del sistema, no fue necesario establecer un programa de capacitación extenso. En su lugar, las sesiones realizadas se enfocaron en socializar el alcance del prototipo desarrollado durante el periodo de prácticas, explicar las principales decisiones de diseño adoptadas y resolver inquietudes puntuales relacionadas con el funcionamiento y la evolución del sistema.

En este sentido, el uso del mismo stack tecnológico definido en los requerimientos no funcionales, en conjunto con el conocimiento previo del equipo sobre la arquitectura de PIA, permitió sentar las bases para que el sistema AdminUAI pueda ser mantenido, actualizado y ampliado de manera autónoma, de acuerdo con las necesidades futuras de la organización

Finalmente, con el fin de complementar la evidencia del proceso de capacitación, se presentan indicadores asociados a las sesiones realizadas. La Figura 45 muestra el número de personas capacitadas por sesión, mientras que la Figura 46 presenta la duración de cada una de las sesiones de capacitación. Estos resultados permiten evidenciar que, a pesar de la duración reducida del proceso formativo, el equipo adquirió el conocimiento necesario para continuar con la evolución del sistema de manera autónoma.

Figura 45

Número de personas capacitadas por sesión

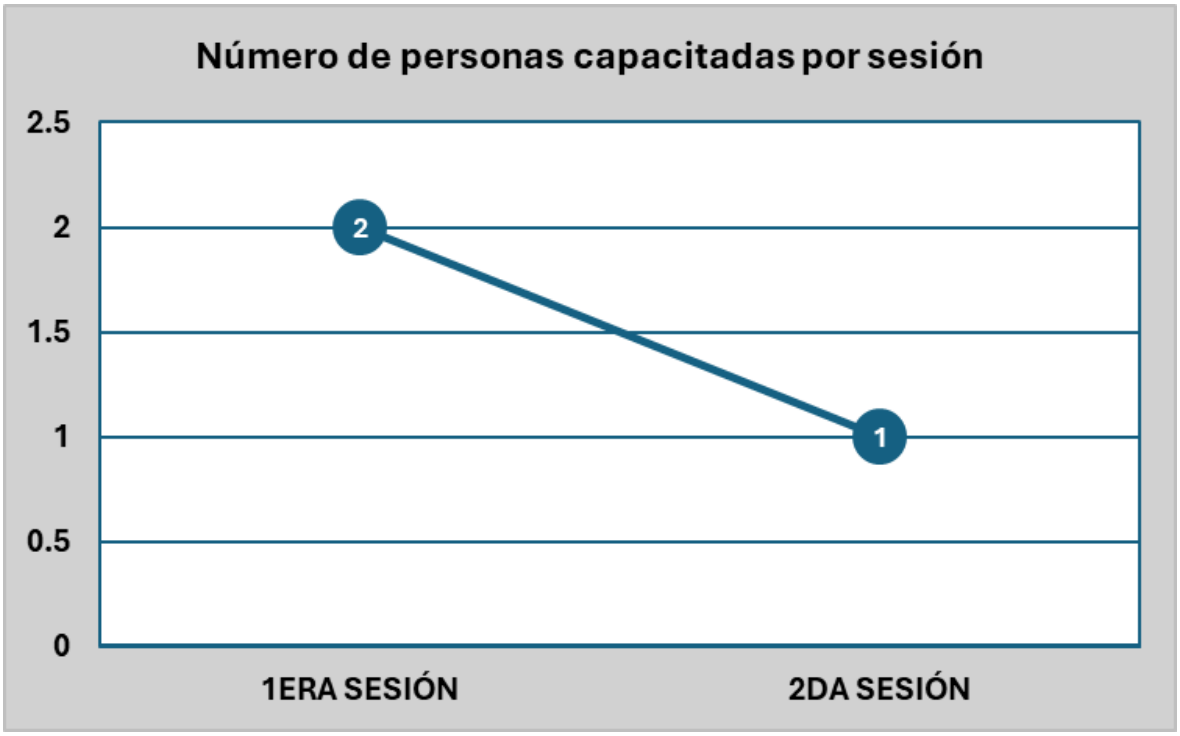
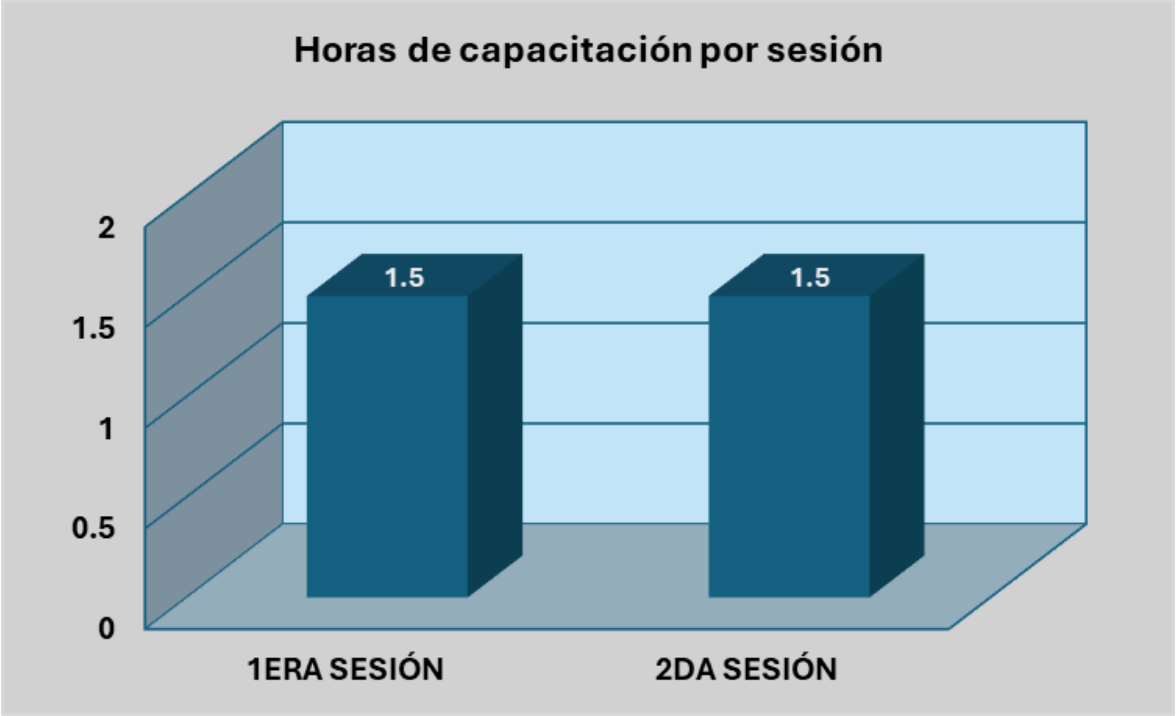


Figura 46

Horas de capacitación por sesión



7. Conclusiones

El desarrollo del aplicativo AdminUAI permitió atender las principales problemáticas identificadas en la sección de planteamiento y justificación del problema, particularmente aquellas relacionadas con la ejecución manual de tareas de configuración, la ausencia de trazabilidad y la alta probabilidad de errores en la administración de la herramienta PIA. A partir de la implementación realizada, fue posible centralizar y automatizar procesos como la creación de clientes, asistentes y prompts, reduciendo la dependencia de intervenciones manuales y facilitando la puesta en funcionamiento de nuevas instancias en menor tiempo y con mayor consistencia.

De manera complementaria, la incorporación de mecanismos como el visor de logs integrado al sistema representa un primer acercamiento a la mejora en la supervisión operativa de PIA, permitiendo consultar información relevante sin necesidad de acceder directamente al servidor. Esto contribuye a disminuir la curva de aprendizaje para nuevos integrantes del equipo de innovación y facilita la identificación de comportamientos del sistema en escenarios reales de uso.

En relación con los objetivos específicos planteados, se logró en primer lugar la definición de los requerimientos funcionales y no funcionales que delimitan el alcance de la solución, estableciendo una base clara para su diseño e implementación. Posteriormente, se diseñó la arquitectura del sistema considerando las restricciones tecnológicas y la integración con el ecosistema existente de PIA. A partir de este diseño, se desarrolló una primera versión funcional del aplicativo, evidenciada en el capítulo de resultados, donde se implementan las principales operaciones de gestión requeridas por el equipo de innovación.

Asimismo, la solución fue validada mediante pruebas unitarias y funcionales ejecutadas de manera iterativa durante el desarrollo, así como en el entorno de pruebas dispuesto por la organización, lo que permitió verificar su correcto funcionamiento bajo diferentes escenarios. Finalmente, se realizó la capacitación al equipo de innovación, con el propósito de facilitar la adopción del sistema y transferir el conocimiento necesario para su uso y evolución.

En términos generales, el proyecto permitió obtener un entendimiento integral del ecosistema tecnológico de PIA y materializarlo en una solución que se integra con los componentes existentes, aportando una base sólida para la gestión del sistema. Si bien esta primera versión no cubre la totalidad de posibles escenarios de configuración, sí establece un

punto de partida funcional que puede ser extendido y adaptado de acuerdo con las necesidades futuras de Pensemos S.A., apoyando la mejora continua de sus procesos internos.

8. Recomendaciones

En primera instancia, se recomienda realizar una validación del diseño actual del sistema con el apoyo de un especialista en experiencia de usuario, con el fin de identificar posibles mejoras en términos de usabilidad, navegación y consistencia visual. Si bien durante el desarrollo se procuró mantener una interfaz clara y funcional, este tipo de evaluación permitiría ajustar de manera más precisa aspectos relacionados con la interacción del usuario, priorizando cambios que aporten valor sin implicar una reestructuración completa del aplicativo. De igual forma, podría considerarse la alineación del estilo visual del sistema con la identidad de los demás productos de Pensemos S.A., como la SVE y PIA, con el propósito de mantener coherencia a nivel organizacional.

Por otra parte, se considera pertinente tener en cuenta la arquitectura y el funcionamiento actual del sistema ante cualquier modificación en el modelo de datos o en las reglas de negocio asociadas a PIA. Dado que el aplicativo AdminUAI se integra directamente con este ecosistema, es importante que cualquier cambio estructural o funcional sea reflejado también en el panel administrativo, garantizando su vigencia y evitando inconsistencias en la configuración de nuevos clientes.

Finalmente, se sugiere promover un proceso continuo de mantenimiento, actualización y ampliación del sistema, de manera que este pueda adaptarse al crecimiento del equipo de innovación y a la evolución de PIA dentro de la organización. La incorporación progresiva de nuevas funcionalidades y ajustes permitirá que el aplicativo conserve su utilidad en el tiempo,

evitando que se convierta en una herramienta obsoleta o limitada frente a las dinámicas del entorno. En este sentido, abordar estas mejoras de forma oportuna contribuirá a reducir costos futuros y a maximizar el valor generado por la solución implementada.

Referencias Bibliográficas

Amazon Web Services. (s.f.-a). *¿Qué es Python?*. Recuperado de https://aws.amazon.com/es/what-is/python/

BrowserStack. (2025a). *Regression Testing vs Unit Testing: Key Differences*. Recuperado de https://www.browserstack.com/guide/regression-testing-vs-unit-testing

BrowserStack. (2025b). *Functional Testing: A Detailed Guide*. Recuperado de https://www.browserstack.com/guide/functional-testing

BrowserStack. (2025c). *What is Flutter: Definition, Benefits, and Limitations*. Recuperado de https://www.browserstack.com/guide/what-is-flutter

Gupta, L. (2025). *What is REST?*. Recuperado de https://restfulapi.net/

Hillard, D. (2024). *Effective Python Testing With pytest*. Recuperado de https://realpython.com/pytest-python-testing/

JWT.io. (s.f.). *Introduction to JSON Web Tokens*. Recuperado de https://www.jwt.io/introduction

Management Solutions. (s.f.). *LLM: definición, contexto y regulación*. Recuperado de https://www.managementsolutions.com/sites/default/files/minisite/static/72b0015f-39c9-4a52-ba63-872c115bfbd0/llm/pdf/auge-de-los-llm-03.pdf

Microsoft. (s.f.). *Estilo de arquitectura de microservicios*. Recuperado de https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

Oracle. (2021). *What is a Relational Database (RDBMS)?*. Recuperado de https://www.oracle.com/latam/database/what-is-a-relational-database/

Papertrail. (s.f.). *Nginx*. Recuperado de

https://www.papertrail.com/solution/guides/nginx/

Pensemos S.A. (s.f.-a). *PENSEMOS SA - Partner*. Recuperado de https://www.softwareseleccion.com/implantador/pensemos+sa-i-1251

Pensemos S.A. (s.f.-b). *Pensemos S.A. LinkedIn*. Recuperado de https://www.linkedin.com/company/pensemos-s-a-/about

Pensemos S.A. (s.f.-c). *Suite Visión Empresarial*. Recuperado de https://pensemos.com/suite-vision-empresarial

Pensemos S.A. (s.f.-d). *Inteligencia artificial en la Suite Visión Empresarial*. Recuperado de https://pensemos.com/inteligencia-artificial-ia-para-gestion-estrategia-y-calidad

Pittet, S. (s.f.). *Los distintos tipos de pruebas de software*. Recuperado de https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing

Pressman, R. S., & Maxim, B. R. (2010). *Ingeniería de software: Un enfoque práctico* (7.ª ed.).

Ramírez, S. (s.f.). *FastAPI*. Recuperado de https://fastapi.tiangolo.com/

Susnjara, S., & Smalley, I. (2025*). ¿Qué son las pruebas de software?*. Recuperado de https://www.ibm.com/es-es/think/topics/software-testing

Visure Solutions. (s.f.). *Requisitos funcionales y no funcionales (con ejemplos).* Recuperado de https://visuresolutions.com/es/alm-guide/functional-vs-non-functional-requirements/