Sistema Proveedor de Identidad (IdP) unificado para los servicios digitales de la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander

Luis Daniel Lemus Gonzalez

Cristian David Fuentes Duarte

Trabajo de Grado para Optar al Título de Ingeniero en Sistemas

Director

Juan Ramon Pernalete Maldonado

Ingeniero de Sistemas

Tutor

Luis Carlos Gomez Florez

Ingeniero de Sistemas

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2026

Dedicatoria

Dedicamos este trabajo de grado a nuestras familias, quienes han sido el pilar fundamental en nuestro proceso académico y personal, brindándonos su apoyo incondicional, motivación constante y confianza en cada etapa de este camino.

A todas aquellas personas que, de una u otra forma, contribuyeron a nuestro crecimiento y nos impulsaron a seguir adelante en los momentos de dificultad, les extendemos este logro como un reflejo del esfuerzo compartido.

Agradecimientos

Expresamos nuestro más sincero agradecimiento a nuestro director de trabajo de grado, por su orientación, acompañamiento y valiosos aportes durante el desarrollo de este proyecto.

A la Universidad y a la Escuela, por brindarnos las herramientas, conocimientos y espacios necesarios para nuestra formación profesional.

A nuestros compañeros y docentes, quienes contribuyeron a lo largo de nuestra carrera con su apoyo, enseñanzas y colaboración.

Finalmente, agradecemos a nuestras familias por su comprensión, paciencia y respaldo incondicional, fundamentales para la culminación de este logro académico.

Tabla de contenido

Índice de Figuras

Lista de Abreviaturas

IdP: Sistema Proveedor de Identidad

IAM: Gestión de Identidad y Acceso

SSO: Inicio de Sesión Único

MFA: Autenticación Multifactor

OIDC: OpenID Connect

OAuth 2.0: Open Authorization 2.0

NIST: National Institute of Standard and Technology

OWASP: Open Worldwide Application Security Protocol

API: Interfaz de Programación de Aplicaciones

JWT: JSON Web Token

UIS: Universidad Industrial de Santander

Glosario

Autenticación: Proceso de verificación de la identidad de un usuario, generalmente mediante credenciales como usuario y contraseña.

Autorización: Proceso de determinar qué permisos o niveles de acceso tiene un usuario ya autenticado dentro de un sistema.

Cliente: En este contexto, se refiere a cualquier aplicación que confía en el sistema central para autenticar usuarios.

Fragmentación de Identidad: Situación en la que los datos de un mismo usuario se encuentran dispersos en múltiples bases de datos o sistemas independientes, obligando a duplicar credenciales e información del usuario.

Token de Acceso: Credencial digital usualmente en formato JWT que permite a una aplicación acceder a recursos protegidos en nombre de un usuario.

Zero Trust: Modelo de seguridad informática que parte de la premisa de que no se debe confiar en nadie por defecto, exigiendo verificación constante

Resumen

Título: Sistema Proveedor de Identidad (IdP) unificado para los servicios digitales de la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander*

Autor(es): Luis Daniel Lemus Gonzalez, Cristian David Fuentes Duarte**

Palabras Clave: Proveedor de identidad (IdP), inicio de sesión único (SSO), autenticación multifactor (MFA), Gestión de identidad y acceso (IAM), ecosistema digital, OpenID Connect, OAuth 2.0.

Descripción:

El presente proyecto documenta la implementación de un sistema centralizado de gestión de identidad para mitigar la fragmentación de credenciales en un ecosistema digital. En la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander, una iniciativa de desarrollo distribuido permite a los estudiantes crear soluciones tecnológicas con autonomía en la selección de herramientas, metodologías y tecnologías. Esta dinámica ha contribuido al fortalecimiento de las competencias técnicas de los estudiantes y al crecimiento de una comunidad de desarrollo activa y colaborativa.

Sin embargo, la creación de múltiples aplicaciones de manera independiente ha generado plataformas aisladas con mecanismos propios de autenticación, gestión de usuarios y control de acceso. Como consecuencia, los usuarios deben administrar diferentes credenciales para acceder a los distintos servicios disponibles, lo que dificulta la experiencia de uso y puede incrementar los riesgos asociados a la seguridad de la información y la administración de identidades.

Para abordar esta problemática, se implementó un Sistema Proveedor de Identidad (IdP) que centraliza los procesos de autenticación y autorización para las aplicaciones participantes. Mediante la aplicación de la metodología RUP y el uso de estándares internacionales de gestión de identidad, se diseñó y desarrolló una arquitectura orientada a la integración de servicios y a la interoperabilidad entre plataformas. Como resultado, se promueve un entorno de acceso más unificado, seguro y eficiente, facilitando la administración de usuarios y mejorando la experiencia de acceso dentro del ecosistema digital de la institución.

* Trabajo de grado

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Juan Ramón Pernalete Maldonado. Tutor: Luis Carlos Gómez Flórez.

Abstract

Title: Identity Provider System (IdP) unified for the digital services of the School of Systems and Informatics Engineering at the Industrial University of Santander*

Author(s): Luis Daniel Lemus Gonzalez, Cristian David Fuentes Duarte**

Key Words: Identity Provider (IdP), Single Sign-On (SSO), Multi-Factor Authentication (MFA), Identity and Access Management (IAM), Digital Ecosystem, OpenID Connect, OAuth 2.0.

Description:

This project documents the implementation of a centralized identity management system to mitigate credential fragmentation in a digital ecosystem. At the School of Systems Engineering and Computer Science at the Industrial University of Santander, a distributed development initiative allows students to create technological solutions with autonomy in the selection of tools, methodologies, and technologies. This approach has contributed to strengthening students' technical skills and fostering the growth of an active and collaborative development community.
However, the development of multiple applications in isolation has resulted in siloed platforms, each with its own authentication, user management, and access control mechanisms. As a result, users must manage different sets of credentials to access the various available services, which complicates the user experience and can increase risks related to information security and identity management.
To address this issue, an Identity Provider (IdP) system was implemented to centralize authentication and authorization processes for participating applications. By applying the RUP methodology and using international identity management standards, an architecture was designed and developed to facilitate service integration and interoperability across platforms. As a result, a more unified, secure, and efficient access environment is promoted, simplifying user administration and enhancing the access experience within the institution's digital ecosystem.

* Final year project

** Physical-Mechanical Engineering Faculty. Systems and Informatics Engineering School. Advisor: Juan Ramón Pernalete Maldonado. Mentor: Luis Carlos Gómez Flórez.

Introducción

La Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander ha adoptado una estrategia de desarrollo tecnológico distribuido que fomenta la creación de sistemas especializados mediante equipos independientes. Este enfoque, alineado con las tendencias modernas de desarrollo ágil y arquitecturas de microservicios, facilita la autonomía técnica y acelera la innovación digital institucional.

El desarrollo distribuido ha transformado la construcción de ecosistemas digitales empresariales, facilitando que organizaciones desarrollen múltiples sistemas tecnológicos mediante equipos independientes. Esta evolución favorece que los sistemas operen de manera autónoma, reduce dependencias entre equipos de desarrollo y acelera los ciclos de entrega, características especialmente valoradas en entornos académicos donde la innovación y experimentación son prioritarias.

La combinación de metodologías ágiles, arquitecturas orientadas a servicios y equipos multidisciplinarios ha impulsado la adopción de soluciones de gestión de identidad unificada como elemento integrador de ecosistemas digitales diversos. Los sistemas de Single Sign-On (SSO) se han posicionado como infraestructura crítica para mantener la cohesión operativa mientras se preserva la autonomía de desarrollo, permitiendo que usuarios accedan de manera transparente a múltiples aplicaciones.

En este contexto, la implementación de un Sistema Proveedor de Identidad (IdP) representa una decisión estratégica que equilibra la autonomía de desarrollo con la integración operativa. La adopción de protocolos estándar como OAuth 2.0 y OpenID Connect permite que cada sistema opere de forma independiente mientras utilizan una infraestructura común de

autenticación y autorización.

El presente documento propone el desarrollo de un sistema proveedor de identidad para abordar la fragmentación de identidad en sistemas aislados del ecosistema digital de la Escuela de Ingeniería de Sistemas e Informática. Se describe la metodología RUP como estrategia para el desarrollo de sistemas críticos de seguridad.

1. Planteamiento y Justificación del Problema

La Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander presenta actualmente una fragmentación en la gestión de identidad digital derivada de su modelo de desarrollo tecnológico distribuído. Los sistemas desarrollados por diferentes equipos tienden a operar con sus propios mecanismos de autenticación independientes, por lo que estudiantes, profesores y personal administrativo deben crear y mantener credenciales distintas para cada plataforma. Esta situación puede generar problemáticas operativas y de seguridad, ampliamente documentadas en instituciones de educación superior (UK Department for science, innovation and technology, 2024).

La falta de un punto de acceso unificado puede resultar frustrante para los usuarios, quienes deben enfrentar ecosistemas digitales donde cada plataforma tiende a mantener mecanismos de acceso independientes. Esta dispersión implica interactuar con múltiples interfaces, cada una con sus propios formularios de registro que establecen distintos requisitos para la creación de contraseñas seguras. La variabilidad en estos criterios puede llevar a los usuarios a establecer contraseñas complejas que resultan difíciles de recordar.

Esta situación puede derivar en procesos frecuentes de recuperación de contraseñas que consumen tiempo, como lo evidencian estudios que reportan que estas solicitudes constituyen una porción significativa del soporte técnico (Forrester Research, 2024). El restablecimiento constante de credenciales podría representar pérdidas de tiempo tanto para los equipos de soporte como para los usuarios, interrumpiendo las actividades diarias que se ejecutan en las plataformas (HYPR & Beyond Identity, 2024).

Esta complejidad también puede inducir a prácticas inseguras por parte de los usuarios.

Ante la necesidad de recordar múltiples contraseñas para diferentes plataformas, los usuarios pueden optar por crear una única contraseña segura que cumpla con los requisitos establecidos, pero utilizarla en todos en todos los sistemas a los que necesitan acceder. Adicionalmente, podrían almacenar estos datos de acceso en medios inseguros para facilitar su recuperación. En entornos universitarios la reutilización de contraseñas puede representar un riesgo mayor que el uso de contraseñas débiles (Nisenoff et al., 2023).

Desde la perspectiva de seguridad, este panorama podría promover riesgos que deben considerarse. El Verizon Data Breach Investigations Report documenta que las credenciales robadas constituyen uno de los vectores de ataque más exitosos en el sector educativo (Verizon, 2024). Adicionalmente, existen evidencias sobre la disponibilidad masiva de información de autenticación universitaria en mercados ilícitos (Digital Citizens Alliance, 2017; FBI Cyber Division, 2022), situación que representa una amenaza persistente para las instituciones.

El problema tiende a amplificarse con el desarrollo continuo de nuevas aplicaciones. Cada plataforma debe implementar múltiples componentes de autenticación independientes, como formularios de acceso, recuperación de contraseñas, gestión de sesiones y validaciones. Esta descentralización puede extender los tiempos de desarrollo y llevar a que se omitan mecanismos esenciales de seguridad como la autenticación multifactor. Diversas investigaciones sugieren que este mecanismo añade una capa adicional de verificación de identidad y reduce significativamente el riesgo de suplantación por credenciales comprometidas (Meyer et al., 2023). Esta variedad de componentes no integrados aumenta la superficie de ataque a puntos que podrían ser vulnerables, dificulta el monitoreo centralizado de eventos y complica la administración del ecosistema tecnológico (GuidePoint Security, 2024).

Un enfoque centralizado facilita el monitoreo de eventos permitiría identificar incidentes de seguridad asociados a credenciales mediante políticas centralizadas que establezcan niveles de riesgo y monitoreen patrones anómalos de comportamiento. Esto permitiría tomar medidas correctivas inmediatas al detectar actividad anómala, bloqueando el acceso comprometido a todas las actividades simultáneamente de un único punto de control. Esta capacidad de detección y respuesta rápida es considerada esencial por el NIST Cybersecurity Framework 2.0 y fundamental para arquitecturas Zero Trust (NIST, 2024; NIST, 2020).

Ante esta necesidad de centralización, las organizaciones suelen adoptar soluciones comerciales como Curity, Auth0 u Okta, servicios de suscripción que ofrecen alta personalización pero representan costos significativos. También existen alternativas de código abierto como Keycloak, las cuales pueden presentar limitaciones para adaptarse a necesidades institucionales específicas y sistemas antiguos.

El sistema desarrollado aborda estas limitaciones mediante una solución personalizada que integre la identidad visual e institucional de la universidad, contemple roles específicos del entorno académico, y se adapte a las plataformas y procesos internos existentes. La implementación seguiría lineamientos internacionales de seguridad, conforme a las recomendaciones del OWASP Authentication Cheat Sheet y NIST SP 800-63B-4 . Esto incluiría mecanismos robustos para el almacenamiento seguro de credenciales, autenticación multifactor y validación de contraseñas. La arquitectura soportará protocolos estándar como OAuth 2.0 y OpenID Connect, facilitando la integración con aplicaciones actuales y futuras (OWASP Foundation, 2024; NIST, 2025).

La necesidad de una solución integral se refuerza con las tendencias de transformación

digital en educación superior, que sugieren ecosistemas de aplicaciones interconectadas y experiencias de usuario mejoradas. El Educause Top 10 Id Issues identifica la gestión de identidad y acceso como prioridad crítica para la resiliencia institucional (EDUCAUSE, 2024). Sin un servicio centralizado de identidad, la UIS puede continuar enfrentando barreras técnicas y operativas que limitarían la innovación y la modernización de sus servicios académicos. La implementación de una plataforma unificada no sólo abordaría los problemas actuales sino que también proporciona bases arquitectónicas para un crecimiento sostenible del portafolio digital, facilitando que futuros desarrollos se integren con mayor eficiencia al ecosistema institucional.

2. Objetivos

2.1 Objetivo General

Desarrollar un sistema centralizado de gestión de identidad que elimine la fragmentación de credenciales mediante la implementación de inicio de sesión único, fortaleciendo la seguridad con autenticación multifactor y estableciendo una arquitectura base para la integración eficiente de los sistemas informáticos en uso y en desarrollo, así como todo otro recurso dispuesto en el futuro.

2.2 Objetivos Específicos

1. Definir los requerimientos del sistema de gestión de identidad, que permitan la selección de estrategias técnicas apropiadas para el contexto institucional del proyecto.
2. Diseñar la arquitectura del sistema y de comportamiento de sus módulos, considerando aspectos de seguridad e integración con los sistemas informáticos existentes, en desarrollo y futuros de la Escuela de Ingeniería de Sistemas e Informática.
3. Implementar los módulos de autenticación centralizada, autorización y los mecanismos de integración necesarios para el funcionamiento del sistema de gestión de identidad.
4. Validar la implementación del sistema en ambiente de producción probando el funcionamiento de sus componentes con usuarios finales e incluyendo la documentación necesaria para su operación y administración.

3. Marco de Referencia

3.1 Marco teórico

3.1.1 *Proveedor de Identidad*

Un proveedor de identidad es un sistema centralizado que verifica la identidad de los usuarios y gestiona su información dentro de un entorno organizacional (NIST, 2025). Este componente actúa como la autoridad única que mantiene las credenciales de autenticación y los atributos de cada usuario, funcionando como fuente de verdad para toda la infraestructura tecnológica.

El proveedor de identidad constituye el núcleo de las arquitecturas de identidades federadas, un modelo donde múltiples sistemas confían en las verificaciones de identidad realizadas por una autoridad común. Cuando un usuario intenta acceder a un sistema federado, esto lo redirige automáticamente al proveedor de identidad para que verifique su identidad. Una vez confirmada la autenticación, el proveedor de identidad genera un artefacto digital que contiene la confirmación de identidad y los atributos del usuario. El sistema que solicitó el acceso valida este artefacto y concede los permisos correspondientes, sin necesidad de mantener sus propias credenciales o realizar un proceso autenticación independiente.

Esta centralización mantiene coherencia en roles, permisos e información personal a través de todos los sistemas integrados, propagando automáticamente cualquier modificación sin intervención manual. El proveedor de identidad registra todas las transacciones de autenticación y autorización, proporcionando trazabilidad para cumplimiento normativo y detección de anomalías de seguridad. Adicionalmente permite establecer políticas de seguridad uniformes y simplifica la gestión del ciclo de vida de usuarios desde un punto único de control, desde su

provisión hasta su desactivación.

3.1.2 *Autorización*

La autorización es el proceso que determina qué acciones o recursos puede utilizar un usuario dentro de un sistema, definiendo sus permisos y nivel de acceso a información o funcionalidades específicas. Este control de acceso resulta fundamental en sistemas distribuidos donde múltiples aplicaciones necesitan compartir recursos de manera segura y controlada.

OAuth 2.0 es el estándar maps utilizado para implementar autorización en sistemas distribuidos (Hardt et al., 2023). Este protocolo hace posible que las aplicaciones accedan a recursos protegidos sin requerir las credenciales del usuario, estableciendo un modelo de delegación de permisos. OAuth 2.0 define cuatro roles fundamentales en su arquitectura, que incluyen el propietario del recurso que posee los datos, la aplicación cliente que solicita acceso, el servidor de autorización que emite permisos y el servidor de recursos que otorga acceso a la información protegida.

Uno de los flujos más seguros y utilizados para aplicaciones web que especifica OAuth 2.0 es el flujo de código de autorización. Este proceso funciona en dos etapas donde se involucran el usuario y la aplicación que requiere acceso. Cuando un usuario intenta acceder a un recurso en el sistema, es redirigido al servidor de autorización para dar su aprobación. Una vez aprobado, el servidor genera una confirmación temporal que la aplicación utiliza para obtener y guardar permisos de acceso a esos recursos a través de una comunicación directa y confidencial entre servidores, sin que esta información pase por el navegador del usuario. Este diseño minimiza riesgos de seguridad al mantener los datos de autorización protegidos durante todo el proceso.

Las aplicaciones cliente requieren registro previo en el servidor de autorización configurando direcciones de redirección permitidas y alcances de permisos autorizados. Este modelo proporciona control detallado sobre qué recursos puede acceder cada aplicación.

3.1.3 *Autenticación*

La autenticación es el proceso que verifica la identidad de un usuario, confirmando que quien intenta acceder a un sistema es realmente quien dice ser. Este proceso establece la confianza inicial necesaria antes de otorgar cualquier permiso o acceso a recursos protegidos.

OpenID Connect es el protocolo estándar para implementar autenticación en sistemas modernos, construido como una extensión de OAuth 2.0 (OpenID Foundation, 2023). A diferencia de OAuth 2.0 que únicamente gestiona autorizaciones, OpenID Connect añade capacidades de autenticación al especificar cómo transmitir y validar la información personal de manera segura y estandarizada. La información incluye atributos como nombres completos, direcciones de correo electrónico, nombres de usuario y otros datos relevantes según las necesidades de cada sistema. Esto facilita confirmar quién es el usuario y acceder a su perfil de forma confiable.

Adicionalmente, una funcionalidad principal de OpenID Connect es el inicio de sesión único. Este mecanismo permite a los usuarios autenticarse una vez en el proveedor de identidad y acceder posteriormente a múltiples aplicaciones sin repetir el proceso de verificación. El proveedor mantiene una sesión persistente que reconoce al usuario autenticado cuando solicita acceso a diferentes sistemas integrados, eliminando la necesidad de ingresar credenciales repetidamente.

3.1.4 *Autenticación de Múltiples Factores*

La autenticación de múltiples factores es un mecanismo que añade capas adicionales de seguridad al proceso de verificación de identidad proporcionando resistencia significativa contra vectores de ataques comunes (NIST, 2025). Este método requiere que el usuario presente más de una forma de verificación para acceder al sistema, reduciendo drásticamente el riesgo de acceso no autorizado incluso cuando las contraseñas están comprometidas. El enfoque más común implementa un segundo factor de verificación mediante contraseñas de un solo uso basadas en tiempo. Estos códigos temporales tienen una validez limitada de pocos minutos y se entregan al usuario a través de diferentes canales como correo electrónico, mensajes de texto o aplicaciones especializadas. Cuando un usuario ingresa su contraseña correctamente, el sistema solicita además el código temporal actual, asegurando una capa adicional de verificación antes de conceder acceso.

Esta arquitectura de seguridad en capas protege las cuentas incluso cuando las contraseñas son robadas, filtradas o descubiertas mediante ataques de fuerza bruta. Un atacante que obtenga la contraseña de un usuario no podrá acceder al sistema sin el segundo factor de verificación. Esta protección resulta especialmente valiosa considerando que las contraseñas pueden verse comprometidas de múltiples formas, desde filtraciones de bases de datos hasta técnicas de ingeniería social.

3.1.5 *Tokens Web en Formato JSON*

Un JSON Web Token (JWT) es un formato estandarizado para transmitir información de manera segura entre sistemas (Vertocci, B., 2021). Este tipo de token funciona como un contenedor que almacena datos verificables, garantizando su integridad y autenticidad mediante

mecanismos criptográficos que previenen su modificación o falsificación durante la transmisión a través de internet.

Los JWT están estructurados en tres secciones fundamentales. El encabezado especifica el tipo de token y el algoritmo criptográfico utilizado para su protección. La carga útil contiene la información relevante sobre el usuario, incluyendo datos estandarizados como el identificador del usuario, la audiencia destinataria del token, el emisor que lo generó y el tiempo de expiración. Esta sección también puede transportar información personalizada sobre roles organizacionales, permisos específicos o atributos adicionales del usuario según las necesidades de cada sistema. Finalmente, la firma digital garantiza que el contenido del token no ha sido alterado, utilizando encriptación asimétrica para asegurar su validez.

La validación de un JWT no requiere comunicación con el servidor que lo emitió. Las aplicaciones pueden verificar independientemente la autenticidad del token usando la clave pública del proveedor de identidad, confirmando que la firma corresponde al contenido y que no ha sido modificado. Esta arquitectura auto-contenida reduce significativamente la latencia al eliminar las consultas remotas de validación, permitiendo que los sistemas distribuidos verifiquen tokens de manera eficiente sin depender de conectividad constante con el emisor original.

3.1.6 *Seguridad en la Capa de Transporte (TLS)*

Transport Layer Security (TLS) es el protocolo que permite a los sistemas comunicarse de forma segura a través de internet, estableciendo un canal cifrado que protege la información durante su transmisión (Rescorla E., 2023). Este mecanismo resulta fundamental en sistemas de

identidad federada donde se intercambian credenciales, tokens de acceso y otra información sensible entre navegadores, aplicaciones y servidores de autenticación.

TLS garantiza tres propiedades esenciales en la comunicación. Primero, la confidencialidad asegura que solo el emisor y el receptor pueden leer el contenido transmitido, impidiendo que terceros intercepten información sensible. Segundo, la integridad verifica que los datos no han sido modificados durante su tránsito, detectando cualquier intento de alteración. Tercero, la autenticación confirma que el servidor con el que se está comunicando es realmente quien dice ser, mitigando ataques de suplantación.

Los sistemas de identidad también utilizan elementos de seguimiento de sesión que mantienen al usuario autenticado a través de múltiples solicitudes u aplicaciones. Estos elementos requieren configuración específica de seguridad para transmitirse exclusivamente por canales de cifrado con TLS, impidiendo su acceso mediante código malicioso y protegiéndolos contra ataques que intentan falsificar solicitudes en nombre del usuario. El cifrado adicional de información sensible dentro de estos elementos proporciona una capa extra de protección, asegurando la confidencialidad incluso si el almacenamiento local del navegador se ve comprometido.

3.2 Estado del arte

En la revisión de literatura, se identificaron trabajos relacionados con la implementación de sistemas de gestión de identidad y acceso, single sign-on y autenticación multifactor en entornos organizacionales y educativos.

Pome Ramírez y Zavaleta Reyes (2023) en su trabajo “Implementación de una arquitectura empresarial para mejorar la gestión de identidades y acceso en una empresa de seguros en Lima” como requisito para optar al título de Ingeniero de Sistemas de la Universidad

Tecnológica del Perú, desarrollaron una propuesta integral de arquitectura empresarial orientada a optimizar la gestión de identidades y acceso en el sector asegurador peruano- Su investigación implementó una solución práctica en una empresa de seguros de Lima utilizando el framework TOGAF como metodología base. El estudio empleó un enfoque cualitativo que abarcó el ciclo de vida completo del Método de Desarrollo de Arquitectura (ADM), proporcionando lecciones valiosas sobre la modernización de sistemas de identidad en entornos corporativos complejos (Poma Ramirez & Zavaleta Reyes, 2023).

La investigación destaca por su aplicación práctica en un entorno empresarial real donde los autores implementaron una solución que integra los principios de arquitectura empresarial con las necesidades específicas de gestión de identidades en organizaciones del sector financiero. Este trabajo proporciona evidencia empírica sobre los beneficios de adoptar marcos arquitectónicos estructurales para abordar la fragmentación de sistemas de autenticación y autorización en entornos corporativos complejos.

Serrano Mora (2023) En su trabajo “Diseño y desarrollo de una aplicación web que permita conocer el nivel de madurez en ciberseguridad para entidades pertenecientes al sector educativo” como requisito para optar al título de Ingeniería de Sistemas de la Universidad Industrial de Santander, desarrolló una aplicación web para evaluar el nivel de madurez en ciberseguridad específicamente para el sector educativo. Su investigación utilizó una metodología que combina los controles CIS Top 18 y las cinco funciones del NIST Cybersecurity Framework para identificar vulnerabilidades y riesgos de seguridad en instituciones educativas (Serrano Mora, 2023).

El trabajo de Serrano Mora valida la importancia crítica de implementar sistemas de

seguridad robustos en entornos universitarios y proporciona un marco de evaluación que complementa la propuesta del Sistema Proveedor de Identidad (IdP) unificado (Serrano Mora, 2023). Mientras su investigación se enfoca en la evaluación del nivel de madurez en ciberseguridad, el presente proyecto aborda la implementación práctica de un sistema unificado de identidad que busca mitigar algunas de las vulnerabilidades identificadas en su análisis del sector educativo.

Rodriguez Romero (2021) en su trabajo "Servicio de autenticación unificado para personas en el marco de los servicios ciudadanos digitales" como requisito para optar al título de Ingeniería de Sistemas y Computación de la Universidad de los Andes, desarrolló un servicio de autenticación unificado para personas naturales en el marco de los servicios ciudadanos digitales. Su investigación aborda la problemática de la fragmentación de credenciales en el acceso a servicios públicos digitales, proponiendo un sistema centralizado que permite la autenticación ciudadana mediante dispositivos móviles en aplicaciones empresariales (Rodriguez Romero, 2021).

El trabajo de Rodríguez Romero resulta particularmente relevante al abordar los desafíos de interoperabilidad entre múltiples proveedores de servicios digitales, estableciendo un marco conceptual para la identificación confiable por parte de terceros reconocidos (Rodriguez Romero, 2021). Su propuesta contempla aspectos críticos de seguridad y demanda, proporcionando lecciones valiosas sobre la implementación de arquitecturas de identidad en contextos donde la confiabilidad y el cumplimiento normativo son prioritarios.

4. Metodología

Para el desarrollo de este proyecto y el cumplimiento del marco propuesto, se adoptó la metodología Proceso Unificado Racional (RUP - Rational Unified Process) de IBM. Esta selección metodológica se fundamentó en las características particulares de los sistemas de identidad. Estos sistemas requieren un enfoque estructurado, con énfasis en la gestión de riesgos de seguridad, trazabilidad de decisiones arquitectónicas y validación continua de controles de acceso.

El RUP ha sido ampliamente adoptado por organizaciones líderes en el desarrollo de sistemas empresariales críticos, incluyendo IBM, Microsoft, Oracle y empresas de consultoría tecnológica como Accenture y Deloitte para proyectos de infraestructura de identidad. La metodología es especialmente reconocida por su efectividad en proyectos donde la seguridad, la integración con sistemas existentes y la documentación exhaustiva son requisitos fundamentales, características que definen precisamente el desarrollo de un sistema proveedor de identidad para una institución educativa.

Las cuatro fases del RUP (Inicio, Elaboración, Construcción y Transición) se adaptan naturalmente a los requerimientos de este proyecto, permitiendo una progresión lógica desde el establecimiento de fundamentos técnicos hasta la entrega de un sistema completamente operativo. Además, se integraron prácticas específicas del dominio de gestión de identidad y acceso, incluyendo la aplicación del NIST Cybersecurity Framework para la evaluación continua de riesgos de seguridad y recomendaciones del OWASP Identity Management Cheat Sheet para desarrollo seguro.

4.1 Primera fase: Fase de Inicio

Esta fase establece las bases técnicas y regulatorias del proyecto, incluyendo la selección fundamentada del enfoque de implementación más adecuado para el contexto institucional. Corresponde a la fase de Inception del RUP, enfocándose en establecer el caso de negocio y el alcance del proyecto.

Actividades:

- Especificación detallada de requerimientos del sistema de identidad, incluyendo aspectos funcionales, de seguridad y de integración
- Investigación y análisis de marcos regulatorios aplicables al tratamiento de datos personales en instituciones educativas públicas
- Capacitación técnica integral en protocolos de identidad, vulnerabilidades comunes y marcos de seguridad reconocidos
- Evaluación comparativa de soluciones tecnológicas disponibles
- Selección del enfoque de implementación más adecuado para el contexto

Objetivo de fase:

Establecer una base de conocimiento técnico y regulatorio que fundamente la toma de decisiones arquitectónicas, junto con la selección del enfoque técnico más apropiado para el desarrollo del sistema de identidad.

Resultados esperados:

- Especificación de requerimientos del sistema: Define qué necesita el sistema mediante requisitos funcionales y no funcionales específicos para la gestión de identidad
- Marco de referencia técnico: Establece el panorama de opciones disponibles mediante consolidación de protocolos, estándares de seguridad y mejores prácticas aplicables
- Registro de decisiones técnicas: Documenta las tecnologías seleccionadas, incluyendo enfoque técnicos, componentes arquitectónicos y herramientas de implementación con su justificación

4.2 Segunda fase: Fase de Elaboración

En esta fase se desarrolla la arquitectura técnica detallada del sistema, especificando componentes, interfaces y controles de seguridad según la alternativa seleccionada en la fase anterior. Esta fase corresponde a la Elaboración del RUP, donde se define y valida la arquitectura del sistema.

Actividades:

- Elaboración de especificación técnica completa del sistema con casos de uso detallados
- Diseño de arquitectura técnica del sistema, incluyendo componentes principales, modelo de datos e interfaces de programación
- Diseño de interfaces de usuario y experiencia de usuario para portales de gestión personal y administración
- Planificación detallada de iteraciones de desarrollo con configuración de entorno y herramientas

Objetivo de fase:

Obtener un diseño arquitectónico detallado que sirva como guía técnica para la fase de implementación, incluyendo especificaciones de los componentes principales del sistema.

Resultados esperados:

- Especificación de Requisitos de Software (ERS): Documentación detallada de requisitos técnicos con casos de uso y criterios de aceptación
- Documento de Arquitectura de Software (DAS): Especificación técnica de la arquitectura del ecosistema, incluyendo diagramas de infraestructura, flujos de interacción entre componentes y modelo de datos
- Prototipos de interfaces de usuario: Diseños validados de portales de gestión personal y administración

4.3 Tercera fase: Fase de Construcción

Esta fase se centra en la implementación iterativa e incremental del sistema de identidad, siguiendo los principios de desarrollo incremental de RUP con entregas funcionales validadas en cada iteración.

Actividades:

- Iteración 1: Implementación de infraestructura base, modelo de datos y servicios fundamentales de autenticación
- Iteración 2: Desarrollo de servicios de autorización, gestión de tokens y aplicación de controles de seguridad

- Iteración 3: Implementación de interfaces de usuario, portales de gestión y validación de seguridad de interfaces
- Iteración 4: Integración con al menos un sistema existente, configuración de inicio de sesión único y verificación de implementación de controles de seguridad definidos

Objetivo de fase:

Desarrollar un sistema de identidad funcional mediante iteraciones incrementales que incluyan validación continua de funcionalidades y controles de seguridad.

Resultados esperados:

- Sistema base operativo: Infraestructura, servicios de autenticación y monitoreo configurados
- Servicios de autorización completos: Gestión de tokens, sesiones y controles de acceso implementados
- Interfaces de usuario funcionales: Portales de gestión personal y administrativa operativos
- Sistema integrado y validado: Inicio de sesión único funcionando con sistema piloto e integración verificada sin interrupciones operativas

4.4 Cuarta fase: Fase de Transición

La fase final del proyecto corresponde a la Transición del RUP, con el objetivo de desplegar el sistema en ambiente de producción y realizar validación con usuarios finales.

Actividades:

- Despliegue del sistema en ambiente de producción con verificación de funcionamiento
- Ejecución de pruebas de aceptación con usuarios reales y recopilación de retroalimentación
- Elaboración de documentación técnica, manuales de usuario y procedimientos de administración

Objetivo de fase:

Entregar un sistema de identidad funcional, validado mediante pruebas con usuarios reales, alineado con los objetivos establecidos en el alcance del proyecto.

Resultados esperados:

- Sistema en producción: Sistema de identidad desplegado y operando de manera estable en ambiente productivo
- Validación funcional: Resultados de pruebas con usuarios reales con análisis de retroalimentación y recomendaciones de mejora
- Documentación técnica: Manuales de usuario, administración y documentación del sistema

El desarrollo seguirá estas fases de manera estructurada, manteniendo flexibilidad para realizar ajustes basados en hallazgos técnicos, retroalimentación de usuarios o cambios en requerimientos que surjan durante el proceso.

5. Resultados

5.1 Fase de Inicio

Esta fase estableció las bases técnicas y regulatorias del proyecto, incluyendo la selección fundamentada del enfoque de implementación más adecuado para el contexto institucional. Corresponde a la fase de Inception del RUP, orientada a la mitigación de riesgos tempranos y la delimitación del alcance del proyecto.

5.1.1 *Especificación de requerimientos del sistema*

Para entender con precisión las necesidades del ecosistema digital de la Escuela, se realizaron sesiones de trabajo en conjunto con los diferentes equipos de desarrollo de la Escuela de Ingeniería de Sistemas e Informática. Estas reuniones colaborativas permitieron identificar las necesidades de los usuarios, los niveles de acceso que cada uno requería y la arquitectura de flujo de la información para garantizar un entorno seguro y funcional.

5.1.1.1 Definición de Roles. En un entorno digital con múltiples aplicaciones, es necesario establecer políticas claras qué definan la capacidad de cada entidad en el sistema. Para esto se adoptó un modelo de autorización basado en roles.

Un rol funciona como una credencial o etiqueta digital que el Proveedor de Identidad (IdP) es decir, el sistema central encargado de verificar quién es cada usuario, asigna a una persona una vez que confirma su identidad. Cuando un sujeto intenta acceder a algún recurso o

funcionalidad, los sistemas evalúan su rol para decidir si le conceden o le niegan el acceso. Es el equivalente digital de mostrar un carné que dice "soy estudiante" o "soy docente" antes de entrar a un espacio determinado.

Para que este modelo sea ordenado y pueda crecer sin volverse difícil de administrar, los roles se organizaron en dos niveles:

Roles Universales (Alcance Global e Institucional): Estos roles funcionan como una identificación oficial dentro del ecosistema digital. Representan el cargo o función principal del usuario dentro de la institución y son reconocidos por todas las aplicaciones integradas al sistema de identidad.

Su carácter universal significa que cualquier sistema conectado al proveedor de identidad (IdP) puede leer y utilizar estos roles sin necesidad de configuraciones adicionales. Por ejemplo, tanto el sistema de biblioteca como el de calificaciones pueden reconocer automáticamente a un usuario como "Estudiante" sin que sea necesario registrar este dato dos veces. Esto reduce la duplicación de información y garantiza coherencia en todo el ecosistema.

Los roles universales definidos son:

- Director de escuela
- Estudiante
- Profesor
- Secretaria de escuela
- Coordinador de escuela

Roles Específicos (Alcance Local por Aplicación): A diferencia de los roles universales, estos funcionan como llaves de acceso a áreas restringidas dentro de una sola aplicación. Están pensados para usuarios que necesitan permisos administrativos, técnicos u operativos muy concretos en un sistema particular, y que no tienen sentido fuera de ese contexto.

Al limitar su alcance a una sola aplicación, se mitiga el riesgo de otorgar permisos innecesarios por error, un principio de seguridad conocido como mínimo privilegio: cada usuario debe tener acceso únicamente a lo que necesita para cumplir su función, y nada más.

Los roles específicos definidos para las aplicaciones actuales son:

- Administrador de SAAM
- Administrador de SAC
- Administrador de SGPP
- Laboratorista

5.1.1.2 Atributos de Usuario. En un sistema de identidad centralizado, no basta con saber qué puede hacer un usuario dentro del sistema, eso lo definen los roles, sino también quién es exactamente esa persona. Para esto se definieron los Atributos de Usuario: el conjunto de datos que conforman el perfil digital único de cada persona dentro del Proveedor de Identidad.

Cuando un usuario inicia sesión exitosamente, el Proveedor de Identidad empaqueta estos atributos y los envía a la aplicación que está solicitando el acceso. Gracias a esto, cada aplicación puede personalizar la experiencia del usuario: mostrar su nombre, enviarle notificaciones a su correo o filtrar la información según su programa académico, sin necesidad de pedirle esos datos nuevamente.

Para cubrir a toda la población que interactúa con los servicios de la Escuela, el modelo de datos se organizó en dos perfiles:

Perfil del Usuario Institucional: Este perfil corresponde a las personas con un vínculo formal y activo con la Universidad Industrial de Santander: estudiantes, profesores y personal administrativo. Dado que su información es de carácter oficial, se definió un conjunto de atributos estandarizado y compartido en todo el ecosistema digital. Esto garantiza que si un usuario actualiza, por ejemplo, su número de teléfono, ese cambio se refleja automáticamente en todas las aplicaciones de la Escuela, sin necesidad de actualizarlo en cada una por separado.

Los atributos que componen este perfil son:

Username: Identificador principal del usuario en el sistema. Para estandarizar el acceso, se homologó para que sea idéntico a la dirección de correo electrónico institucional (ej. usuario@correo.uis.edu.co).

Email: Dirección de correo electrónico oficial provista por la universidad, utilizada como canal principal de comunicación segura.

First name: Campo destinado a almacenar el primer y segundo nombre del usuario, según su documento de identidad.

Last name: Campo destinado a almacenar el primer y segundo apellido.

Número de documento: Identificador legal (Cédula de Ciudadanía, Tarjeta de Identidad o Pasaporte) utilizado para validaciones en procesos oficiales o legales dentro de las aplicaciones.

Email personal: Dirección de correo electrónico alternativa, proporcionada por el usuario para procesos de recuperación de cuenta o comunicaciones de emergencia si pierde acceso a su correo institucional.

Código estudiantil: Identificador numérico único asignado por la universidad a los estudiantes matriculados, fundamental para el cruce de datos con los sistemas académicos centrales.

Teléfono: Número de contacto móvil o fijo.

Programa académico: Atributo fundamental que identifica la escuela o facultad a la que pertenece el usuario. Este dato permite a las aplicaciones segmentar la información (por ejemplo, mostrar noticias solo a estudiantes de Ingeniería de Sistemas). El catálogo de programas soportados por el IdP se definió de la siguiente manera:

- 10: Biología
- 11: Ingeniería de Sistemas
- 14: Química
- 16: Licenciatura en matemáticas
- 21: Ingeniería Civil
- 23: Ingeniería Industrial
- 24: Ingeniería Mecánica
- 27: Diseño Industrial
- 30: Licenciatura en música
- 32: Ingeniería de Petróleos
- 33: Ingeniería Química

- 39: Matemáticas
- 40: Física
- 47: Ingeniería en Inteligencia Artificial
- 50: Ingeniería en Ciencia de Datos
- 56: Fisioterapia
- 57: Nutrición y dietética
- 58: Microbiología y bioanálisis
- 69: Ingeniería Biomédica

Perfil del Usuario Externo: El ecosistema digital de la Escuela también contempla aplicaciones que requieren la participación de personas que no pertenecen a la institución: aspirantes aún no matriculados, egresados sin correo institucional activo, invitados a eventos o representantes de empresas externas, entre otros. Para estos casos se definió el perfil de Usuario Externo.

Este perfil se diseñó excluyendo intencionalmente los datos de carácter académico, como el código estudiantil o el programa académico, ya que estos campos simplemente no aplican para esta población. En cambio, su estructura se centra en los datos básicos de identificación y contacto, suficientes para garantizar la trazabilidad de las acciones del usuario dentro de los sistemas a los que tiene acceso permitido.

Los atributos de este perfil son:

- Username: Dirección de correo electrónico personal utilizada para el registro.
- Email: Dirección de correo electrónico personal para notificaciones.
- First name: Primer y segundo nombre.

- Last name: Primer y segundo apellido.
- Número de documento: Identificador legal.
- Teléfono: Número de contacto.

5.1.1.3 Operaciones en el Proveedor de Identidad. Por razones de seguridad y de cumplimiento de normativas, las aplicaciones del ecosistema no almacenan directamente la información de los usuarios en sus propias bases de datos. En su lugar, cada aplicación guarda únicamente el UUID (Universally Unique Identifier), un código único e inmutable que el Proveedor de Identidad asigna a cada persona. Cuando una aplicación necesita información de un usuario, utiliza este código para consultarla directamente al Proveedor de Identidad, garantizando integridad y que siempre trabaje con datos actualizados.

Para hacer facilitar esto, se definió la necesidad de un componente intermediario que centralice las consultas al Proveedor de Identidad y las ponga a disposición de todas las aplicaciones. Las operaciones requeridas para este componente son: búsqueda de un usuario por su UUID, búsqueda por atributos del perfil (nombre, correo, programa académico, entre otros) y búsqueda por rol.

5.1.1.4 Servicio Transversal de Mensajería. Se identificó la necesidad de centralizar el envío de correos electrónicos del ecosistema en un único servidor de correo institucional. Para esto, se definió la creación de un componente compartido al que todas las aplicaciones puedan acudir cuando sea necesario enviar una notificación, evitando que cada sistema debe gestionar su propia configuración de correo y garantizando que todas las comunicaciones se realicen bajo una identidad institucional uniforme.

Esto evita que cada sistema gestione credenciales de forma independiente, reduciendo la superficie de ataque.

5.1.1.5 Laboratorio de Desarrollo de Integración. Conectar una aplicación a un nuevo sistema de identidad es una operación crítica: los equipos de desarrollo necesitan un espacio donde puedan probar esa conexión de forma segura, sin afectar el entorno real donde los usuarios trabajan. Para cubrir esta necesidad, se requirió la creación de un entorno de laboratorio basado en estándares internacionales de la industria.

Este entorno funciona como una "pista de pruebas" aislada del sistema productivo, donde los equipos pueden verificar que su aplicación se comunica correctamente con el Proveedor de Identidad, que los roles y atributos llegan como se espera y que la integración es estable antes de salir a producción. Adicionalmente, el laboratorio incorpora flujos de Integración y Despliegue Continuo (CI/CD), un mecanismo automatizado que permite que cada cambio en el código sea compilado, probado y desplegado de forma automática, reduciendo errores humanos y acelerando los ciclos de desarrollo.

5.1.1.6 Manejo de Secretos. Las aplicaciones de software necesitan credenciales para funcionar: contraseñas de bases de datos, llaves de acceso a servicios externos, certificados de seguridad, entre otros. A estos datos sensibles se les denomina secretos. Una práctica común pero peligrosa es almacenarlos directamente en el código fuente de la aplicación, lo que los expone a cualquier persona que tenga acceso al repositorio.

Para eliminar este riesgo en el ecosistema de la Escuela, se requirió la implementación de un Vault, una bóveda digital de secretos. En lugar de que cada equipo gestione sus credenciales de forma independiente, el Vault actúa como una fuente de confianza centralizada: las aplicaciones solicitan sus secretos de forma dinámica y bajo demanda en el momento en que los necesitan, y el Vault los entrega de forma cifrada y controlada, registrando cada acceso. Esto no solo reduce el riesgo de filtraciones, sino que también centraliza la administración de la seguridad en un único punto de control.

5.1.1.7 Estándar de Integración Transversal. Sin una directriz clara, cada equipo de desarrollo podría implementar la autenticación de usuarios a su manera: unos con pantallas de inicio de sesión propias, otros con validaciones distintas, generando una experiencia inconsistente y multiplicando los puntos donde un error de seguridad podría ocurrir.

Para evitar esto, se estableció que la autenticación en todo el ecosistema debe ser completamente transversal: ninguna aplicación implementa su propia lógica de inicio de sesión. En su lugar, un componente de infraestructura compartido actúa como una puerta de seguridad principal que intercepta cada solicitud antes de que llegue a la aplicación. Este componente verifica con el Proveedor de Identidad que el usuario tiene una sesión válida y, de ser así, le

entrega a la aplicación la información de identidad ya verificada —quién es el usuario y qué roles tiene—, sin que la aplicación haya tenido que hacer ninguna validación por su cuenta.

Este enfoque tiene dos beneficios directos: simplifica significativamente el desarrollo de nuevas aplicaciones, que ya no necesitan preocuparse por la seguridad del inicio de sesión, y garantiza que todos los sistemas del ecosistema apliquen exactamente el mismo estándar de autenticación.

5.1.2 *Marco de referencia técnico*

5.1.2.1 Análisis de Marcos Regulatorios. Como parte de las fases iniciales del proyecto, se realizó una investigación sobre los marcos legales y regulatorios que aplican al desarrollo de aplicaciones digitales expuestas al público en el contexto de una universidad pública colombiana. El objetivo de esta revisión fue identificar las obligaciones legales que el ecosistema debe cumplir al recolectar, almacenar y procesar información personal de sus usuarios, evitando así riesgos legales y garantizando un manejo ético y responsable de los datos.

Ley 1581 de 2012 — Régimen General de Protección de Datos Personales: Esta es la norma principal en materia de protección de datos en Colombia. Fue expedida para garantizar el derecho de todas las personas a conocer, actualizar y rectificar la información recopilada sobre ellas en bases de datos, ya sea por entidades públicas o privadas (Congreso de la República de Colombia, 2012). Su base está en el Artículo 15 de la Constitución Política, que protege la intimidad personal, y aplica a todas las entidades que recolectan, almacenan o procesan datos personales en Colombia, incluyendo universidades (TI Rescue, 2025). Los artículos de mayor relevancia para este proyecto son:

Art. 1 — Objeto: Establece el derecho de toda persona a conocer, actualizar y rectificar su información en bases de datos. Este principio es el fundamento de por qué el sistema de identidad diseñado permite a los usuarios gestionar su propio perfil.

Art. 3 — Definiciones: Introduce los términos clave del marco legal. El Titular es la persona dueña de los datos; el Responsable es quien decide para qué se usan; el Encargado es quien los trata por cuenta del responsable; y el Tratamiento engloba cualquier operación sobre los datos, como recolectarlos, almacenarlos o compartirlos. En el contexto del proyecto, la Universidad Industrial de Santander actúa como Responsable del Tratamiento, y los sistemas del ecosistema digital como Encargados.

Art. 8 — Derechos de los Titulares: Los usuarios tienen derecho a conocer qué datos suyos están almacenados, actualizarlos, rectificarlos, solicitar su eliminación y revocar en cualquier momento el permiso que otorgaron para su uso. Estos derechos orientaron el diseño del perfil de usuario en el Proveedor de Identidad, que centraliza la información para que cualquier actualización se refleje de forma inmediata en todo el ecosistema.

Art. 9 — Autorización Es el pilar de todas las leyes: Ninguna entidad puede usar los datos de una persona sin haber obtenido previamente su consentimiento explícito. Esto implica que las aplicaciones del ecosistema deben informar claramente al usuario para qué se van a usar sus datos antes de solicitarlos.

Art. 12 y 13 — Deber de informar: Al momento de recolectar datos, se debe comunicar al titular la finalidad del tratamiento y a qué terceros podría compartir su información. Este artículo fundamenta la necesidad de contar con políticas de privacidad claras en las aplicaciones.

Art. 26 — Transferencia Internacional: Se prohíbe el envío de datos personales a países que no ofrezcan niveles adecuados de protección, salvo excepciones específicas contempladas en la ley. Este artículo es relevante en la medida en que el proyecto evalúa el uso de servicios en la nube o herramientas de terceros con servidores en el exterior (Congreso de la República de Colombia, 2012).

CONPES 3995 de 2020 — Política Nacional de Confianza y Seguridad Digital: Este documento de política pública establece lineamientos para que las entidades del Estado adopten medidas de ciberseguridad que protejan su infraestructura digital. Las universidades públicas, al ser parte del Estado colombiano, están cobijadas por esta política, lo que implica que sus sistemas deben implementar controles técnicos para proteger sus servidores y plataformas frente a amenazas digitales. Este marco sustenta decisiones de diseño como el uso del Vault para el manejo de secretos y la centralización de la autenticación en un único Proveedor de Identidad.

Ley 30 de 1992 — Educación Superior: Esta ley otorga autonomía a las universidades públicas en la gestión de sus asuntos internos. Sin embargo, es importante precisar que dicha autonomía no exime a las instituciones del cumplimiento del régimen general de protección de datos personales. El Consejo de Estado ha sido claro en este sentido: la autonomía universitaria opera en el ámbito académico y administrativo, pero no constituye una excepción frente a las obligaciones legales que aplican a cualquier entidad que trate datos personales de ciudadanos colombianos.

5.1.2.2 Capacitación Técnica Integral. El diseño e implementación de un ecosistema de identidad digital no es una tarea que pueda abordarse únicamente con conocimiento general en desarrollo de software. Implica comprender y aplicar estándares, protocolos y tecnologías especializadas que, en muchos casos, el equipo de trabajo no había utilizado previamente. Ante esta realidad, se reconoció desde una etapa temprana que la capacitación técnica no era un complemento opcional, sino una condición necesaria para procurar que las decisiones de diseño fueran fundamentadas, coherentes y alineadas con las mejores prácticas de la industria.

Un criterio transversal que orientó toda la investigación fue el uso exclusivo de tecnologías de código abierto (open source): soluciones cuyo código fuente es público, libre de usar, modificar y distribuir sin necesidad de adquirir licencias comerciales ni suscripciones. Esta restricción no es una limitación, sino una decisión estratégica que favorece la sostenibilidad del proyecto, su independencia tecnológica y su viabilidad dentro del presupuesto de una institución pública. Cada tecnología investigada y seleccionada cumple con este criterio.

Los resultados de esta capacitación fueron documentados como entregable formal de esta fase. A continuación se presentan los conceptos y tecnologías estudiados.

Cuando una organización crece y sus usuarios necesitan acceder a múltiples sistemas distintos, surge de manera natural una pregunta: ¿debe cada sistema conocer, verificar y almacenar por separado la identidad de cada persona? La respuesta intuitiva sería que sí, pero esa solución escala mal. Con el tiempo, los usuarios acumulan múltiples contraseñas, los administradores multiplican los puntos de gestión y los riesgos de seguridad se dispersan en lugar de concentrarse donde pueden controlarse.

SISTEMA PROVEEDOR DE IDENTIDAD

La Federación de Identidades es la respuesta que la industria tecnológica ha desarrollado para este problema. Describe las tecnologías, estándares y casos de uso que permiten la portabilidad de la información de identidad a través de dominios de seguridad autónomos, con el objetivo de que los usuarios de un dominio puedan acceder de forma segura a los datos o sistemas de otro dominio sin necesidad de administración de usuarios redundante. En términos cotidianos, es el principio que hace posible que una persona use sus credenciales de un sistema para acceder a otro completamente diferente, del mismo modo en que un pasaporte emitido por un país es reconocido y aceptado en las fronteras de otros países sin necesidad de tramitar una identificación nueva en cada uno (UNAD, 2025).

Dentro de este modelo, los sistemas participantes se distribuyen en dos roles bien definidos:

Proveedor de Identidad (Identity Provider): Es la entidad central que almacena las identidades, verifica las credenciales y emite certificados de autenticación. La responsabilidad de revisar y autenticar las credenciales del usuario recae sobre el Proveedor de Identidad, no sobre las aplicaciones individuales. En el contexto de este proyecto, el Proveedor de Identidad es el sistema central que se implementa para la Escuela (TI Rescue, 2025).

Proveedor de Servicios (Service Provider): Son las aplicaciones que confían en el Proveedor de Identidad para verificar a sus usuarios. En lugar de gestionar sus propios mecanismos de autenticación, delegan esa responsabilidad. Cuando dos dominios están federados, un usuario puede autenticarse en uno y luego acceder a recursos del otro sin necesidad de realizar un proceso de inicio de sesión separado (OpenID Foundation, 2023).

A medida que las aplicaciones web y móviles se multiplicaron, surgió una necesidad cotidiana pero técnicamente compleja: permitir que una aplicación acceda a los datos de un usuario en otro servicio, sin que ese usuario tenga que revelar su contraseña. La solución que la industria adoptó como estándar para este problema es OAuth 2.0 (Open Authorization 2.0).

OAuth 2.0 es un marco de autorización de estándar abierto que permite a las aplicaciones solicitar acceso delegado y limitado a recursos protegidos, utilizando tokens de acceso de corta duración en lugar de contraseñas (Baquero Giraldo, 2019). La analogía más precisa es la de un pase de acceso temporal: en lugar de entregar las llaves de su casa a un visitante, el propietario le entrega un pase que solo abre una puerta específica y que expira después de cierto tiempo. La aplicación nunca conoce la contraseña del usuario; solo recibe un token que representa un permiso concreto.

Es importante precisar que OAuth 2.0 es un marco de autorización, no un protocolo de autenticación. Su propósito principal es permitir que una aplicación acceda a recursos en nombre de un usuario sin manejar sus credenciales directamente (UNAD, 2025). Dicho de otro modo: OAuth 2.0 responde a la pregunta "¿qué puede hacer esta aplicación?", pero no necesariamente a "¿quién es este usuario?". Esta distinción, aunque sutil, es fundamental y motivó el surgimiento del protocolo complementario que se describe más adelante.

Para adaptarse a los distintos tipos de aplicaciones que existen —web, móvil, servidor a servidor—, OAuth 2.0 define el concepto de grant type, que es la manera en que una aplicación obtiene su token de acceso. Cada tipo está optimizado para un caso de uso particular(Hardt et al., 2023). Los flujos de mayor relevancia para este proyecto son:

Authorization Code Flow: Es el flujo más utilizado para aplicaciones que involucran interacción con el usuario y cuentan con un servidor de respaldo. Separa el paso de autenticación del proceso de intercambio del token, lo que procura que los tokens de acceso nunca queden expuestos al navegador. Es el flujo recomendado para las aplicaciones web del ecosistema de la Escuela.

Authorization Code Flow con PKCE (Proof Key for Code Exchange): Es una extensión del flujo anterior diseñada para clientes que no pueden almacenar secretos de forma segura, como aplicaciones móviles o de una sola página (Single Page Applications). Protege contra la interceptación del código de autorización vinculando la solicitud inicial al intercambio del token (UNAD, 2025).

Client Credentials Flow: Se utiliza para comunicación entre servidores, en escenarios donde no hay un usuario involucrado. Los servicios usan este flujo para intercambiar credenciales de forma segura y obtener tokens de acceso (Ordoñez D., s.f.). Es el flujo apropiado para la comunicación entre los componentes internos del ecosistema.

OAuth 2.0 resolvió el problema de la autorización delegada, pero dejó abierta una pregunta que muchas aplicaciones también necesitan responder: ¿quién es exactamente la persona que acaba de autenticarse? Para llenar este vacío, se desarrolló OpenID Connect, un protocolo de autenticación interoperable construido sobre el marco de OAuth 2.0, que simplifica la forma de verificar la identidad de los usuarios basándose en la autenticación realizada por un servidor de autorización (Bhargava R., 2024).

La relación entre ambos protocolos puede entenderse con una analogía: si OAuth 2.0 es el sistema que le permite a alguien entrar a un edificio con un pase temporal, OpenID Connect es

la capa adicional que hace que ese pase también lleve una fotografía e información de identidad. Mientras OAuth 2.0 emite tokens de acceso que representan permisos, OpenID Connect introduce el concepto del ID token, un documento digital que contiene información verificable sobre la identidad del usuario (Arulmmozhi K., 2025).

Este ID token adopta el formato de un JWT (JSON Web Token), que es esencialmente un documento compacto y firmado digitalmente. Los JWT contienen claims, que son declaraciones sobre el usuario —como su nombre o dirección de correo electrónico— junto con metadatos adicionales sobre la sesión de autenticación. Gracias a la firma digital, cualquier aplicación que reciba este token puede verificar que fue emitido legítimamente por el Proveedor de Identidad y que su contenido no ha sido alterado, sin necesidad de consultar al servidor en cada solicitud.

OpenID Connect extiende OAuth 2.0 con funcionalidades de autenticación de usuarios e inicio de sesión único (Single Sign-On), permitiendo recuperar y almacenar información de autenticación sobre los usuarios finales (Fatenbberg J., 2024). Esto significa que una persona puede iniciar sesión una única vez en el Proveedor de Identidad y acceder sin interrupciones a todas las aplicaciones del ecosistema, sin necesidad de ingresar sus credenciales en cada una.

En síntesis, OAuth 2.0 y OpenID Connect son protocolos complementarios que trabajan en conjunto: el primero gestiona los permisos de acceso a recursos, y el segundo se encarga de verificar y comunicar la identidad del usuario. Esta combinación constituye la base técnica sobre la que se construye el sistema de identidad centralizado del ecosistema de la Escuela.

El desarrollo de software rara vez ocurre en un único paso lineal. En la práctica, múltiples desarrolladores trabajan simultáneamente sobre distintas partes de un mismo sistema, introducen cambios con frecuencia y necesitan verificar constantemente que esos cambios no rompan lo que

ya funcionaba. A medida que un proyecto crece en complejidad, coordinar este flujo de trabajo de forma manual se vuelve progresivamente inmanejable. La respuesta que la industria ha consolidado para este desafío es la adopción de prácticas de Integración Continua y Entrega Continua, conocidas por sus siglas en inglés CI/CD (Continuous Integration / Continuous Delivery).

La Integración Continua hace referencia a la práctica de incorporar automáticamente y con frecuencia los cambios de código en un repositorio de código fuente compartido. La Entrega Continua, por su parte, es un proceso de dos partes que abarca la integración, las pruebas y la preparación de esos cambios para su despliegue (NIST, 2024). Juntas, estas prácticas conforman lo que se conoce como un pipeline de CI/CD: una cadena automatizada de pasos que el código debe superar antes de considerarse apto para llegar a los usuarios.

Una analogía útil es la de una línea de ensamblaje en una fábrica. Cada pieza que entra a la línea pasa por estaciones de verificación automáticas: se compila, se prueba, se analiza en busca de vulnerabilidades y, finalmente, se empaqueta para su despliegue. Al automatizar estos procesos, los equipos de desarrollo y operaciones pueden minimizar el error humano y mantener un proceso consistente para la forma en que el software es publicado (Sánchez Castillo V., 2025).

Los tres conceptos que componen este enfoque tienen alcances distintos y progresivos:

Integración Continua (CI): Cada vez que un desarrollador introduce un cambio en el repositorio, se activa automáticamente un proceso que compila el código y ejecuta pruebas. El objetivo es detectar errores lo antes posible, cuando aún son pequeños y fáciles de corregir. Esto facilita que las organizaciones mitiguen errores y fallas en el código mientras mantienen un ciclo continuo de desarrollo y actualizaciones (NIST, 2024).

Entrega Continua (CD - Continuous Delivery): Es una extensión de la integración continua que despliega automáticamente todos los cambios de código a un entorno de pruebas después de la fase de construcción. Esto significa que, además de las pruebas automatizadas, existe un proceso de publicación automatizado, y la aplicación puede desplegarse en cualquier momento con solo confirmar la acción (Ordoñez D., s.f.). La decisión final de pasar a producción recae en una persona.

Despliegue Continuo (CD - Continuous Deployment): Es el paso final en un pipeline maduro. Una vez que el código supera todas las etapas de prueba, su despliegue al entorno de producción ocurre de forma completamente automática, sin necesidad de intervención humana (Ordoñez D., s.f.).

Para el ecosistema de la Escuela, la adopción de una estrategia de CI/CD no fue simplemente una buena práctica de ingeniería: fue una condición casi obligatoria para que el proyecto fuera viable.

Un Proveedor de Identidad centralizado es, por naturaleza, un sistema del que dependen todas las demás aplicaciones. Intentar construir y conectar cada aplicación directamente en un entorno de producción —el entorno real donde trabajan los usuarios— habría representado un riesgo inaceptable. Cualquier error en la configuración de un protocolo, en la definición de un rol o en la estructura de un token podría haber afectado a todos los sistemas del ecosistema simultáneamente, sin posibilidad de revertir los cambios de forma controlada.

La estrategia adoptada consistió en estructurar el trabajo en ambientes progresivos, cada uno con un propósito definido:

Ambiente de desarrollo (dev): El espacio de trabajo cotidiano de los equipos. Aquí se construyen y prueban las integraciones por primera vez, se identifican incompatibilidades entre protocolos, se verifican los flujos de autenticación y se depuran los errores sin consecuencias para los usuarios reales. Detectar defectos en esta etapa temprana es significativamente menos costoso que encontrarlos en producción (TI Rescue, 2025).

Ambiente de pruebas (staging): Un entorno que replica las condiciones del sistema productivo. Una vez que la construcción supera las pruebas iniciales, puede empaquetarse y enviarse a este entorno, donde es posible validarla en condiciones cercanas a las reales antes de tomar la decisión de desplegarla (MEN, 2020).

Ambiente de producción (prod): El entorno real. Solo los cambios que han superado exitosamente los ambientes anteriores llegan aquí, lo que procura que las actualizaciones sean predecibles, estables y de bajo riesgo.

Este esquema de ambientes, orquestado por el pipeline de CI/CD, transformó el proceso de integración de las aplicaciones al Proveedor de Identidad en un flujo estandarizado y reproducible. Cada equipo de desarrollo pudo incorporar su aplicación siguiendo el mismo camino verificado, reduciendo la fricción técnica y facilitando que los errores de integración fueran identificados y resueltos en etapas tempranas, antes de comprometer el entorno productivo.

A medida que el ecosistema digital crece e incorpora nuevas aplicaciones, surge una pregunta de diseño fundamental: ¿quién es el responsable de verificar que un usuario esté autenticado antes de dejarlo entrar? La respuesta más intuitiva sería que cada aplicación se encargue de esa verificación por su cuenta. Sin embargo, este enfoque distribuido presenta un

problema serio: cada equipo implementa la autenticación a su manera, con distintos niveles de rigor, distintas experiencias de inicio de sesión y múltiples puntos donde un error de seguridad podría ocurrir de forma independiente.

La alternativa que la industria ha consolidado para este problema es el patrón de autenticación por proxy de borde. Este enfoque consiste en colocar un proxy inverso consciente de la identidad frente a las aplicaciones, de modo que se encargue de forma centralizada de interceptar las solicitudes, manejar la autenticación con el Proveedor de Identidad e inyectar la información de identidad verificada hacia el sistema de destino, todo sin que las aplicaciones necesiten modificar su código (NIST, 2025).

La analogía más precisa es la de la recepción de un edificio corporativo: ningún visitante llega directamente a las oficinas internas. Primero debe pasar por la recepción, donde se verifica su identidad y se le entrega un carné de acceso. Sólo entonces puede subir. Las oficinas no necesitan preguntar quién es cada persona; confían en que quien llegó hasta ellas ya fue verificado en la entrada.

Cuando un usuario intenta acceder a cualquier aplicación del ecosistema, el proxy de borde actúa como único punto de control. El proceso ocurre de la siguiente manera:

1. Interceptación de la solicitud: El proxy intercepta todas las solicitudes dirigidas a las aplicaciones protegidas y verifica si el usuario cuenta con una sesión válida, representada normalmente por una cookie de sesión cifrada almacenada en el navegador (NIST, 2025).

2. Redirección al Proveedor de Identidad: Si el usuario no tiene una sesión activa, el proxy lo redirige automáticamente al Proveedor de Identidad, donde ingresa sus credenciales.

Esta redirección es transparente: el usuario simplemente ve la pantalla de inicio de sesión institucional, sin importar a qué aplicación intentaba acceder originalmente.

3. Recepción del token: Una vez que el Proveedor de Identidad verifica las credenciales, redirige al usuario de regreso al proxy con un código de autorización. El proxy intercambia ese código por los tokens de sesión correspondientes —el access token y el ID token— y establece la sesión del usuario mediante una cookie segura en el navegador.

4. Inyección de identidad hacia el backend: Con la sesión validada, el proxy reenvía la solicitud original hacia la aplicación de destino, adjuntando la información del usuario autenticado en las cabeceras HTTP de la petición, incluyendo datos como el correo electrónico, el nombre y los roles asignados (UNAD, 2025). La aplicación recibe esta información ya verificada y puede utilizarse directamente, sin haber participado en ningún paso del proceso de autenticación.

La adopción de este patrón tiene implicaciones prácticas significativas para el desarrollo y la seguridad del ecosistema:

Separación total de responsabilidades: Las aplicaciones se concentran exclusivamente en su lógica de negocio. La autenticación es una preocupación del proxy, no de cada equipo de desarrollo. Esto reduce la complejidad de cada proyecto y mitiga el riesgo de implementaciones incorrectas.

Inicio de Sesión Único (SSO) transparente: Una vez que el usuario se autentica a través del proxy, puede acceder a cualquier otra aplicación protegida del ecosistema sin necesidad de

iniciar sesión nuevamente, ya que la cookie de sesión es reconocida por el proxy en cada solicitud subsiguiente (UNAD, 2025).

Protección de aplicaciones sin capacidad nativa de autenticación: Este modelo es especialmente valioso para sistemas que no fueron construidos con soporte para protocolos modernos de autenticación, ya que el proxy añade esa capa de seguridad de forma externa sin requerir modificaciones en el código de la aplicación (NIST, 2020).

Punto único de auditoría: Al centralizar el control de acceso en un único componente, todas las solicitudes de autenticación pasan por el mismo lugar, lo que facilita el registro y monitoreo de accesos en todo el ecosistema desde un solo punto.

Construir un sistema de identidad centralizado implica asumir una responsabilidad mayor que la de una aplicación común: si el Proveedor de Identidad es comprometido, todas las aplicaciones que dependen de él quedan expuestas simultáneamente. Esta realidad exigió una fase de estudio rigurosa sobre las amenazas y vulnerabilidades más relevantes para el tipo de sistema que se estaba desarrollando.

La investigación se organizó en dos bloques complementarios: el catálogo de vulnerabilidades del estándar OWASP Top 10:2025, y un conjunto de vectores de ataque específicos que apuntan directamente a los mecanismos de autenticación e identidad.

El OWASP Top 10 es una lista de las diez categorías de vulnerabilidades más críticas en aplicaciones web, actualizada periódicamente con base en datos de la industria. Sirve como documento de referencia para desarrolladores y equipos de seguridad, y es frecuentemente utilizado como línea base en auditorías de seguridad y procesos de cumplimiento (Ordoñez D.,

s.f.). La edición 2025 incorpora dos categorías nuevas que reflejan la evolución del panorama de amenazas moderno.

A01:2025 — Control de Acceso Roto (Broken Access Control): Sigue siendo el riesgo más prevalente. Cubre situaciones en las que los usuarios logran actuar fuera de los permisos que les fueron asignados, lo que puede derivar en divulgación no autorizada de información, modificación de datos o ejecución de funciones restringidas (Baquero Giraldo, 2019). Esta categoría es de máxima relevancia para el ecosistema, dado que el modelo de roles y el control de acceso son componentes centrales del sistema diseñado. En la edición 2025, se incorporó a esta categoría el ataque de tipo SSRF (Server-Side Request Forgery), reconociendo que muchas vulnerabilidades de este tipo son, en esencia, fallas de control de acceso (UNAD, 2025).

A02:2025 — Mala Configuración de Seguridad (Security Misconfiguration): Ascendió al segundo puesto. Esta categoría abarca situaciones en las que un sistema, aplicación o servicio en la nube fue configurado de forma incorrecta desde el punto de vista de la seguridad, creando vulnerabilidades explotables. Ejemplos comunes incluyen cuentas por defecto no eliminadas, servicios innecesarios activos, permisos excesivos y cabeceras de seguridad ausentes (UNAD, 2025).

A03:2025 — Fallas en la Cadena de Suministro de Software (Software Supply Chain Failures): Es una categoría nueva en 2025, que amplía el concepto anterior de "componentes vulnerables" para abarcar todo el ciclo de vida del software. Reconoce que las dependencias de terceros, las herramientas de construcción y los paquetes de código abierto pueden introducir vulnerabilidades desconocidas o incluso código malicioso sin que el equipo de desarrollo lo advierta (UNAD, 2025).

A04:2025 — Fallas Criptográficas (Cryptographic Failures): Agrupa las debilidades relacionadas con ausencia de cifrado, uso de algoritmos criptográficos insuficientemente robustos y exposición de llaves criptográficas (Baquero Giraldo, 2019). En el contexto de un sistema de identidad, esta categoría es crítica: los tokens, las contraseñas y los secretos del sistema deben estar protegidos mediante algoritmos modernos y bien configurados.

A05:2025 — Inyección (Injection): Agrupa las vulnerabilidades en las que un atacante logra insertar código o comandos maliciosos en los campos de entrada de una aplicación, induciendo al sistema a ejecutarlos como si fueran instrucciones legítimas. SQL e inyección de comandos de shell son los ejemplos más conocidos (Baquero Giraldo, 2019).

A06:2025 — Diseño Inseguro (Insecure Design): Se refiere a fallas en la arquitectura o la lógica del sistema, no en su implementación. Flujos de recuperación de contraseña débiles, ausencia de modelado de amenazas o pasos de autorización omitidos son manifestaciones de esta categoría (Ordoñez D., s.f.). Su inclusión en el Top 10 refleja un giro importante en la industria: la seguridad debe considerarse desde el diseño, no añadirse como un parche posterior.

A07:2025 — Fallas de Autenticación (Authentication Failures): Ocurren cuando un sistema no verifica correctamente la identidad de sus usuarios, permitiendo que un atacante se haga pasar por una persona legítima. Esta categoría engloba desde contraseñas débiles y sesiones sin expiración hasta mecanismos de recuperación de cuenta vulnerables (Baquero Giraldo, 2019). Es la categoría de mayor impacto directo sobre el sistema diseñado en este proyecto.

A08:2025 — Fallas en la Integridad del Software o los Datos (Software or Data Integrity Failures): Agrupa situaciones en las que el sistema no verifica la integridad de actualizaciones,

datos críticos o pipelines de despliegue, permitiendo que código o información modificada de forma maliciosa sea procesada como válida.

A09:2025 — Fallas en el Registro y las Alertas de Seguridad (Security Logging and Alerting Failures): Esta categoría destaca la importancia no solo de registrar los eventos del sistema, sino de contar con mecanismos de alerta que induzcan acciones oportunas ante eventos relevantes. Un sistema con excelentes registros pero sin alertas tiene un valor mínimo para la detección de incidentes de seguridad (Guidepoint Security, 2024).

A10:2025 — Manejo Inadecuado de Condiciones Excepcionales (Mishandling of Exceptional Conditions): Es una categoría nueva en 2025, centrada en cómo los sistemas fallan. Un manejo deficiente de errores puede filtrar información sensible —como trazas de pila o llaves de configuración— o generar condiciones en las que el sistema falla de forma abierta, concediendo acceso en lugar de denegarlo (Ordoñez D., s.f.).

Más allá del catálogo general de OWASP, se estudiaron en profundidad un conjunto de vectores de ataque que apuntan específicamente a los mecanismos de sesión, cookies y tokens que conforman el núcleo de cualquier sistema de autenticación.

CSRF — Falsificación de solicitudes entre Sitios (Cross-Site Request Forgery): Este ataque ocurre cuando un sitio malicioso induce al navegador del usuario a realizar acciones no deseadas en un sitio de confianza donde ya tiene una sesión activa. Dado que el navegador incluye automáticamente las cookies de sesión en cada solicitud, el sitio destino no puede distinguir si la acción fue iniciada voluntariamente por el usuario o fabricada por el atacante (Bhargava R., 2024). El mecanismo de defensa estándar consiste en incluir en cada solicitud un token secreto e impredecible que el sitio malicioso no puede conocer.

SSRF — Falsificación de Solicitudes del Lado del Servidor (Server-Side Request Forgery): Este ataque explota la capacidad del servidor para realizar solicitudes hacía recursos internos. El atacante manipula la aplicación para que el propio servidor haga peticiones hacia servicios internos que no deberían ser accesibles desde el exterior —como bases de datos, servicios de configuración o metadatos de infraestructura en la nube—, utilizando la confianza que la red interna deposita en el servidor.

Secuestro de Cookies (Cookie Hijacking): Las cookies de sesión son el equivalente digital de una llave de acceso temporal. Si un atacante logra interceptar o robar esta cookie —mediante tráfico no cifrado, scripts maliciosos u otras técnicas—, puede suplantar la identidad del usuario sin necesidad de conocer su contraseña. Las defensas más efectivas incluyen el uso obligatorio de HTTPS, marcar las cookies como HttpOnly —impidiendo que scripts de JavaScript puedan leerlas— y aplicar el atributo SameSite para restringir en qué contextos el navegador las envía.

Ataque de Tabla Arcoíris (Rainbow Table Attack): Cuando una base de datos de contraseñas es comprometida, los atacantes intentan revertir los valores almacenados para obtener las contraseñas originales. Una tabla arcoíris es una estructura precalculada que mapea miles de millones de contraseñas comunes a sus valores cifrados, permitiendo encontrar coincidencias de forma casi instantánea. La defensa contra este tipo de ataque es el uso de sal criptográfica (salt): un valor aleatorio único que se añade a cada contraseña antes de realizar el cifrado, haciendo que dos contraseñas idénticas produzcan valores almacenados completamente distintos e inutilizan cualquier tabla precalculada.

Robo y Reutilización de Tokens (Token Theft & Replay): Los tokens de acceso son credenciales digitales de corta duración. Si un atacante logra interceptar un token válido antes de que expire, puede utilizarlo para acceder a los recursos protegidos sin necesidad de autenticarse. A diferencia de otros ataques que requieren una sesión activa, los ataques de reutilización simplemente reenvían datos capturados previamente, por lo que las defensas se centran en el uso de marcas de tiempo, valores de un solo uso (nonces) y tokens de corta duración.

AiTM — Adversario en el Medio (Adversary-in-the-Middle): Este ataque representa una evolución sofisticada del clásico ataque de intermediario. El atacante despliega un servidor proxy transparente entre la víctima y el servicio legítimo, retransmitiendo las credenciales y los factores de autenticación en tiempo real. Una vez que el usuario completa el proceso —incluso si utiliza autenticación de múltiples factores—, el atacante captura la cookie de sesión resultante y la utiliza para acceder al sistema con una identidad completamente válida (Fortinet, s.f.). Este vector es particularmente relevante porque es capaz de eludir mecanismos de seguridad que se consideraban robustos, como la autenticación de dos pasos.

5.1.2.3 Registro de decisiones técnicas. Para concluir la Fase de Inicio, y tomando como base los requerimientos levantados y la capacitación integral recibida, se procedió a evaluar diferentes enfoques de arquitectura y soluciones disponibles en el mercado. El objetivo fue seleccionar un conjunto de herramientas tecnológicas que permitieran materializar el ecosistema digital, procurando un equilibrio entre seguridad, rendimiento y sostenibilidad institucional.

Como premisa fundamental para la Selección del enfoque de implementación, se determinó que toda la infraestructura debía basarse en software de código abierto (Open Source). Esta decisión estratégica garantiza que la Escuela de Ingeniería de Sistemas e Informática pueda

utilizar, auditar, modificar y extender las herramientas sin depender del pago de licencias o suscripciones a terceros, favoreciendo la autonomía tecnológica y la viabilidad financiera del proyecto a largo plazo.

El resultado de esta evaluación se consolida en el presente registro de decisiones técnicas, donde cada herramienta seleccionada cumple un rol específico dentro de la arquitectura general:

5.1.2.3.1 ***Núcleo de Identidad y Autenticación.*** Keycloak (Proveedor de Identidad - IdP): Actúa como la "oficina de pasaportes" central de la institución. Se seleccionó por ser una solución Open Source robusta y ampliamente respaldada por la industria. Su principal ventaja radica en su alta capacidad de extensión y personalización mediante plugins, lo que permite adaptarlo a las reglas de negocio específicas de la universidad.

Redis (Gestión de Sesiones): Funciona como la "memoria a corto plazo" del ecosistema. Es un motor de base de datos en memoria (RAM) de código abierto, seleccionado para manejar las sesiones activas de los usuarios de forma ultrarrápida, procurando que la validación continua de credenciales no degrade el rendimiento de las aplicaciones.

5.1.2.3.2 ***Control de Tráfico y Seguridad Perimetral.*** OPNsense (Firewall): Actúa como la "muralla perimetral" del entorno del laboratorio. Es una plataforma de seguridad de código abierto que inspecciona y filtra el tráfico entrante y saliente, procurando mitigar intentos de acceso no autorizados desde el exterior.

OpenVPN (Red Privada Virtual): Funciona como un "túnel blindado". Se eligió por su alta efectividad y naturaleza abierta, permitiendo a los desarrolladores y administradores

conectarse de forma segura a la infraestructura interna desde ubicaciones remotas, sin exponer los servidores directamente a internet.

Nginx (Edge Proxy y Servidor Web): Opera como el "agente de tránsito" principal. Se ubica en el borde de la red para recibir todas las peticiones de los usuarios y permite establecer la ruta a la aplicación correcta. A diferencia de otras soluciones más limitadas, Nginx permite una extensibilidad profunda mediante módulos y programación en lenguaje Lua, lo que facilita implementar lógicas de validación de identidad transversales antes de que el tráfico toque las aplicaciones.

5.1.2.3.3 ***Observabilidad y Monitoreo.*** Grafana con Loki (Monitoreo y Bitácoras): Como se evidenció en la capacitación sobre vulnerabilidades, la falta de registro de eventos (OWASP A09) es un riesgo crítico. Estas herramientas de código abierto actúan como las "cámaras de seguridad y el libro de actas" del sistema. Loki centraliza los registros (logs) de todas las aplicaciones, y Grafana permite visualizarlos en tiempo real, facilitando la detección de anomalías y la mitigación proactiva de ataques.

5.1.2.3.4 ***Infraestructura, Almacenamiento y Despliegue.*** Docker (Contenerización): Se adoptó para empaquetar las aplicaciones en "contenedores estandarizados". Esto garantiza que el software funcione de manera idéntica independientemente del entorno (desarrollo, pruebas o producción), facilitando el despliegue concurrente de múltiples sistemas y aislando sus procesos por seguridad.

MinIO (Almacenamiento de Objetos): Actúa como una "bodega de alta seguridad" para archivos. Dado que la infraestructura está basada en contenedores (Docker), no es una práctica segura guardar documentos directamente en los discos de los servidores mediante volúmenes

estáticos. MinIO ofrece un almacenamiento de binarios compatible con el estándar S3, mitigando riesgos de acceso no autorizado al sistema de archivos del servidor host.

GitLab (Laboratorio CI/CD): Se seleccionó como el entorno central para gestionar el código fuente y automatizar el despliegue de las aplicaciones. Al ser una solución Open Source que permite instalación local (On-Premise), la institución mantiene el control total sobre la propiedad intelectual de sus desarrollos.

5.2 Fase de Elaboración

En esta fase se desarrolla la arquitectura técnica detallada del sistema, especificando componentes, interfaces y controles de seguridad según la alternativa seleccionada en la fase anterior. Esta fase corresponde a la Elaboración del RUP, donde se define y valida la arquitectura del sistema.

5.2.1 *Especificación de Requisitos de Software (ERS)*

Para procurar que el sistema de identidad responda fielmente a las necesidades de la institución, se elaboró la Especificación de Requisitos de Software (ERS). Este documento traduce las reglas de negocio en Casos de Uso concretos. Un caso de uso funciona como un guión: describe paso a paso cómo un actor (un usuario o un sistema) interactúa con la plataforma para lograr un objetivo específico, estableciendo Criterios de Aceptación claros que sirven como lista de verificación para dar por válida la funcionalidad.

A continuación, se detallan los requerimientos centrales del ecosistema:

5.2.1.1 Caso de Uso 1: Inicio de Sesión Único (Single Sign-On). Actor: Usuario Institucional / Usuario Externo.

- Descripción: Este flujo actúa como la "recepción central" del edificio digital. Cuando un usuario intenta acceder a cualquier aplicación protegida del ecosistema, el sistema detiene su paso y lo redirige al Proveedor de Identidad (IdP). Allí, el usuario ingresa sus credenciales una única vez. Si son correctas, el IDP le otorga un "pase de acceso" (Token) válido para ingresar a esa y otras aplicaciones sin tener que volver a identificarse.
- Criterios de Aceptación:
    - El sistema debe redirigir al usuario a la pantalla de inicio de sesión centralizada de Keycloak al detectar un intento de acceso no autenticado.
    - El sistema debe validar las credenciales ingresadas. Si son incorrectas, debe mostrar un mensaje de error genérico (procurando no revelar si falló el usuario o la contraseña para mitigar ataques de enumeración).

5.2.1.2 Caso de Uso 2: Validación Transversal de Sesión (Edge Proxy). Actor: Nginx (Proxy de Borde) y Aplicación Satélite.

- Descripción: Para evitar que cada equipo de desarrollo programe su propia puerta de seguridad, este caso de uso define cómo el Proxy (Nginx) intercepta el tráfico. Funciona como un "guardia de seguridad invisible" que revisa el pase del usuario en milisegundos. Si el pase es válido, Nginx inyecta la información de identidad en la petición y la deja pasar hacia la aplicación.
- Criterios de Aceptación:

- El componente transversal debe interceptar todas las peticiones entrantes dirigidas a las aplicaciones del ecosistema.
- El sistema debe verificar la validez del token de sesión contra el servidor de Redis o el IdP.
- Si la sesión es válida, el proxy debe inyectar un token firmado.
- Si la sesión es inválida o ha expirado, el sistema debe bloquear el paso y redirigir al usuario al flujo de inicio de sesión (Caso de Uso 1).

5.2.1.3 Caso de Uso 3: Consumo del Servicio Transversal de Mensajería. Actor: Aplicación Satélite (ej. Sistema de Matrícula o Laboratorio).

- Descripción: Define el estándar mediante el cual cualquier sistema de la Escuela puede solicitar el envío de un correo electrónico institucional. En lugar de tener múltiples sistemas enviando correos desordenados, todos entregan su "carta" a un buzón central que se encarga del despacho oficial.
- Criterios de Aceptación:
  - El micro servicio de mensajería debe exponer un contrato de comunicación (API) que reciba los parámetros: email de destino, subject (asunto) y body (cuerpo del mensaje en formato HTML).
  - El sistema debe validar que la petición provenga de una aplicación autorizada dentro de la red del laboratorio usando credenciales solo para esa app.
  - Tras el envío exitoso, el servicio debe retornar un código de confirmación (ej. Success: true).
  - En caso de fallo en el servidor de correos, el sistema debe registrar el error en la bitácora central (Loki) para facilitar su depuración.

5.2.1.4 Caso de Uso 4: Acceso Transparente a Múltiples Aplicaciones. Actor: Usuario Institucional / Usuario Externo.

- Descripción: Este flujo representa el verdadero beneficio del Inicio de Sesión Único (SSO). Es el equivalente a recibir una "manilla VIP" al entrar a un parque de diversiones: una vez que el usuario se autentica en la primera aplicación (ej. el sistema de matrícula), si abre una segunda aplicación (ej. el sistema de biblioteca) en la misma sesión de su navegador, el sistema debe reconocerlo instantáneamente sin pedirle sus credenciales de nuevo.
- Criterios de Aceptación:
    - Si un usuario posee una sesión activa en el IdP e intenta acceder a una nueva aplicación del ecosistema, el sistema debe evaluar el token existente.
    - El componente transversal debe dejar pasar al usuario a la nueva aplicación de forma transparente, inyectando sus roles correspondientes sin mostrar la pantalla de inicio de sesión.

5.2.1.5 Caso de Uso 5: Verificación de Identidad por Doble Factor (MFA vía OTP). Actor: Usuario Institucional / IdP (Keycloak).

- Descripción: Para mitigar ataques donde un tercero obtenga la contraseña del usuario, se establece una segunda barrera de seguridad. Funciona como el cajero automático que, además de la tarjeta, exige un PIN. En este caso, tras ingresar la contraseña correctamente, el sistema envía una Clave Única Temporal (OTP numérica) al correo del usuario para confirmar que realmente es él quien intenta acceder.
- Criterios de Aceptación:

- El sistema debe generar un código numérico temporal y enviarlo mediante el servicio transversal de mensajería al correo registrado.
- El IDP debe bloquear el acceso final hasta que el usuario ingrese el código correcto.
- El código OTP debe tener un tiempo de expiración corto (ej. 5 minutos) para procurar la seguridad de la transacción.

5.2.1.6 Caso de Uso 6: Gestión Autónoma del Ciclo de Vida del Usuario. Actor: Usuario Institucional / Usuario Externo.

- Descripción: Agrupa las funcionalidades de autoservicio delegadas por el sistema (gestionadas a través del motor de Keycloak). Funciona como una "taquilla de autoatención" donde el usuario puede gestionar su propia cuenta sin requerir soporte técnico constante, reduciendo la carga operativa de la institución. Incluye el registro inicial, la asignación de contraseñas temporales y la actualización de perfil.
- Criterios de Aceptación:
  - El sistema debe permitir la creación de cuentas de usuarios externos solicitando únicamente los atributos definidos en su perfil.
  - Para cuentas nuevas o restablecimientos, el sistema debe emitir una contraseña temporal que fuerce al usuario a cambiarla en su primer inicio de sesión exitoso.
  - El usuario debe poder actualizar datos de contacto no críticos (como su teléfono o correo personal) directamente desde su panel de control en el IdP.

5.2.1.7 Caso de Uso 7: Consulta Centralizada de Atributos de Usuario (Directorio). Actor: Aplicaciones Satélites del Ecosistema.

- Descripción: Dado que las aplicaciones solo almacenan el identificador único (UUID), este flujo describe cómo los sistemas consultan el "directorio telefónico" central. Si un sistema necesita saber el nombre o el programa académico asociado a un UUID para generar un reporte, le "pregunta" al IdP.
- Criterios de Aceptación:
    - El IdP debe exponer un servicio (API) protegido para la consulta de información de usuarios.
    - La aplicación solicitante debe enviar el UUID válido y el IdP debe retornar los atributos autorizados del perfil correspondiente.

Adicional a los flujos de usuario final, se definieron requisitos operativos para el equipo técnico, mediante casos de uso de administración y operación:

5.2.1.8 Caso de Uso 8: Consulta de Bitácoras de Seguridad (Monitoreo). Actor: Administrador del Sistema.

- Descripción: Para mantener la observabilidad del ecosistema, el administrador utiliza Grafana como una "torre de control". Desde allí, puede consultar los registros (logs) centralizados por Loki para auditar inicios de sesión fallidos, errores de red o comportamientos anómalos.
- Criterios de Aceptación:
    - El administrador debe poder filtrar los registros por aplicación, tipo de evento o UUID del usuario.

- El sistema de monitoreo debe proveer paneles visuales (dashboards) que faciliten la identificación de picos inusuales de tráfico.

5.2.1.9 Caso de Uso 9: Integración de Nuevas Aplicaciones en Laboratorio. Actor: Equipo de Desarrollo Estudiantil.

- Descripción: Define el proceso estandarizado para que un equipo conecte una nueva herramienta al IdP utilizando el entorno de pruebas. Funciona como una "pista de ensayo" antes de salir al público.
- Criterios de Aceptación:
  - El equipo debe poder registrar el "Cliente" (aplicación) en el entorno de laboratorio de Keycloak, obteniendo sus llaves de integración de forma segura (mediante el Vault).
  - Las pruebas de inicio de sesión no deben afectar las sesiones ni los datos del entorno de producción.

5.2.1.10 Caso de Uso 10: Flujo de autenticación Centrado en la Identidad. Actor: Usuario Institucional / Usuario Externo.

- Descripción: A diferencia de los sistemas tradicionales que piden usuario y contraseña en la misma pantalla, este flujo funciona como un "recepcionista inteligente" que primero pregunta únicamente: *"¿Quién eres?"*. Al ingresar solo el correo electrónico, el IdP evalúa el estado del usuario para decidir el siguiente paso. Esto facilita enormemente la experiencia del usuario, ya que el sistema lo guía dinámicamente dependiendo de si su cuenta es nueva, si ya está activa, o si fue migrada de un sistema antiguo y necesita activación.

- Criterios de Aceptación:
    - El sistema debe presentar una pantalla inicial que solicite exclusivamente el identificador del usuario (correo electrónico).
    - Si el usuario existe y su cuenta está activa, el sistema debe avanzar a una segunda pantalla para solicitar la contraseña y/o el código OTP.
    - Si el usuario fue migrado de un sistema anterior pero no ha inicializado su cuenta, el sistema debe detectar este estado y redirigirlo a un flujo de verificación de correo y asignación de contraseña nueva.
    - Si el correo ingresado no existe en la base de datos, el sistema debe habilitar visiblemente la opción o el botón de "Registrarse" para iniciar la creación de un nuevo usuario externo.

5.2.1.11 Caso de Uso 11: Migración de Cuentas de Usuarios Existentes. Actor: Administrador del Sistema / IdP (Keycloak).

- Descripción: Un ecosistema universitario no nace desde cero; posee bases de datos previas. Este caso de uso actúa como el traslado cuidadoso de "archivos históricos" hacia la nueva "bóveda digital central". Define el mecanismo para importar usuarios desde las aplicaciones antiguas de la Escuela hacia el nuevo Proveedor de Identidad, procurando que la comunidad no pierda su acceso ni su identidad digital durante la transición tecnológica.
- Criterios de Aceptación:
    - El sistema debe permitir la importación masiva de perfiles de usuario (con sus respectivos atributos como nombre, correo y programa académico) mediante el consumo de las APIs de administración de Keycloak.

- Por razones de seguridad criptográfica, las contraseñas antiguas no deben migrar. El sistema debe crear las cuentas migradas en un estado de "acción requerida".
- Al intentar ingresar por primera vez (detectado por el flujo *Identity-First*), el sistema debe obligar al usuario migrado a verificar su identidad mediante su correo y establecer una nueva contraseña segura bajo las nuevas políticas del IDP.

5.2.1.12 Caso de Uso 12: Cierre de Sesión Único (Single Logout - SLO) en el Borde

Actor: Usuario Institucional / Nginx (Edge Proxy).

- Descripción: Si el SSO es la "llave maestra" para entrar a todas las oficinas del ecosistema, el SLO es el mecanismo central para "apagar las luces y cerrar el edificio" al salir. Cuando un usuario finaliza su jornada en una aplicación (ej. SAAM), el sistema debe invalidar su acceso no solo en esa plataforma, sino en todo el entorno. Para lograrlo, el Proxy de borde expone una ruta específica por aplicación encargada de destruir la sesión central, mitigando el riesgo de que sesiones huérfanas sean secuestradas en equipos compartidos.
- Criterios de Aceptación:
  - El Edge Proxy debe exponer un *endpoint* o ruta estandarizada de cierre de sesión para cada aplicación satélite.
  - Al invocar esta ruta, el sistema debe comunicarse con el IdP y/o el gestor de sesiones (Redis) para invalidar el token actual.
  - Tras el cierre exitoso, el Proxy debe denegar cualquier petición subsecuente que intente usar el token revocado, redirigiendo al usuario a la pantalla de inicio.

5.2.1.13 Caso de Uso 13: Autorización Granular por Capacidades en el Proxy. Actor: Nginx (Edge Proxy) / Aplicación Satélite.

- Descripción: Este flujo actúa como un "control de aduanas avanzado". No basta con saber que el usuario es el “Director de Escuela” (su rol); el sistema debe saber exactamente qué acciones puede ejecutar dentro de un sistema específico. Se definió un modelo donde el Proxy evalúa "capacidades" (permisos específicos) antes de que la petición toque la aplicación. Por ejemplo, permite al Director listar usuarios en el sistema SAAM, pero le bloquea esa misma acción en otras aplicaciones donde no tiene competencia.
- Criterios de Aceptación:
    - El componente transversal debe poder inspeccionar el token interno de cada petición para identificar al usuario, su rol y la aplicación destino.
    - El Proxy debe contrastar esta información contra una matriz de acceso (Access Control List) configurada previamente.
    - Si el rol posee la capacidad requerida para la ruta solicitada, el tráfico se permite; de lo contrario, el Proxy debe rechazar la petición con un código de error de autorización (HTTP 403 Forbidden), protegiendo al backend de peticiones no autorizadas.

5**.2.1.14 Caso de Uso 14: Enrutamiento Unificado y Mitigación de CORS.** Actor: Nginx (Edge Proxy) / Equipos de Desarrollo.

- **Descripción:** En arquitecturas web modernas, la separación entre la interfaz visual (frontend) y la lógica de negocio (backend) suele generar vulnerabilidades y errores de comunicación cruzada (CORS - Cross-Origin Resource Sharing). Para facilitar la

integración, el Proxy actúa como un "conmutador telefónico central". Permite a los equipos configurar un único dominio oficial (ej. saam.uis.edu.co) y enrutar el tráfico internamente (ej. enviando las peticiones visuales al servidor frontend y las peticiones de datos a /api en el backend).

- **Criterios de Aceptación:**
    - El Proxy debe permitir la configuración dinámica de reglas de enrutamiento por dominio, separando el tráfico de frontend y backend bajo un mismo nombre de host.
    - Esta unificación debe procurar la mitigación nativa de errores CORS en los navegadores de los usuarios.
    - El enrutamiento debe ocultar la topología interna de la red y los puertos reales de los servidores satélite, reduciendo la superficie de ataque expuesta a internet.

5.2.1.15 Caso de Uso 15: Renovación Silenciosa de Sesión (Auto-refresco de Tokens).

Actor: Nginx (Edge Proxy mediante Lua) / IdP (Keycloak).

- Descripción: Por principios de ciberseguridad, los "pases de acceso" (Tokens) deben tener una vida útil muy corta (ej. 15 minutos) para mitigar el riesgo de robo. Sin embargo, si un profesor pasa dos horas registrando calificaciones, sería inaceptable interrumpirlo cada quince minutos para pedirle su contraseña. Para solucionar esto, el Proxy actúa como un "escolta discreto". Utilizando scripts avanzados en lenguaje Lua, Nginx detecta cuándo el pase está a punto de caducar y se comunica silenciosamente con Keycloak ("la oficina central") para solicitar una extensión del tiempo, todo esto en fracciones de segundo y sin interrumpir el trabajo del usuario.
- Criterios de Aceptación:

- El componente transversal (Nginx) debe ser capaz de evaluar el tiempo de expiración del Token de Acceso en cada petición entrante.
- Si el token ha expirado, pero la sesión general sigue siendo válida, el script en Lua debe interceptar la petición y consumir internamente el servicio de renovación (Refresh Token) del IdP.
- Al obtener el nuevo Token de Acceso, el Proxy debe actualizar la sesión en la memoria (Redis) e inyectar las nuevas credenciales en la petición original.
- El proceso completo debe ser imperceptible para el usuario final y para la aplicación satélite, procurando una experiencia de uso fluida y segura.

5.2.2 *Documento de Arquitectura de Software (DAS)*

5.2.2.1 Análisis Integral

Este documento tiene como objetivo presentar una propuesta de diseño de arquitectura de resultado de la evaluación técnica realizada al servicio que se está prestando, es un artefacto que permite a su audiencia:

- Entender quién es y qué hace el cliente.
- Dar un contexto de la solución.
- Dar un contexto técnico de la propuesta a presentar.
- Entender la arquitectura definida.
- Certificar que la estrategia arquitectónica definida se encuentra alineada a los objetivos establecidos en la iniciativa.
- Hay que asegurar que la arquitectura implementada satisfaga las necesidades del cliente y se cumpla la oferta de valor.

- Identificar Riesgos técnicos.

Descripción del cliente: La Escuela de Ingeniería de Sistemas e Informática - EISI- se encuentra adscrita a la Facultad de Ingenierías Fisicomecánicas de la Universidad Industrial de Santander.

Comprometida con la Misión Institucional, la EISI tiene como propósitos: la formación de personas autónomas, creativas, que actúen según principios éticos universalmente aceptados, de alta calidad ciudadana y comprometidos con el desarrollo regional y nacional; así como la construcción, innovación y mejoramiento del conocimiento, que permita disponer de la fundamentación teórica, tecnológica e instrumental para administrar y tratar los sistemas de información, las comunicaciones y la automatización industrial.

Para dar cumplimiento a lo anterior, la Escuela de Ingeniería de Sistemas e Informática ofrece a la comunidad académica los siguientes programas:

- Pregrado: Ingeniería de Sistemas
- Posgrado: Maestría en Ingeniería de Sistemas e Informática (investigación)
- Posgrado: Maestría en Informática para la Educación (profundización)
- Posgrado: Doctorado en Ciencias de la Computación

A través de su historia, los programas de la EISI se han caracterizado por responder a las necesidades del medio formando profesionales de alta calidad en este campo. Es así, como sus egresados se encuentran laborando en el sector público, educativo, empresarial y en sus propias empresas de informática. La Escuela, en su compromiso con la región, contribuye al progreso aportando análisis y solución de problemas del entorno, mediante la realización de proyectos de investigación y extensión (proyectos de grado).

La Escuela realiza análisis, diseño, programación, documentación, prueba e

implementación de sistemas software en diferentes campos de aplicación como la robótica, bioinformática, procesamiento de lenguaje natural, visión, sistemas expertos, redes neuronales, algoritmos genéticos, modelos, pedagogía y educación. Así mismo, desarrolla sistemas de información en todas las áreas como manejo administrativo ajustado al tipo de empresa, administración de eventos (deportivos, culturales, etc.), sistemas orientados a la web, desarrollo de material educativo para todos los niveles de educación, tutoriales inteligentes, simuladores, montaje de plataformas para la enseñanza virtual, entre otros.

5.2.2.2 Misión. La Escuela de Ingeniería de Sistemas e Informática (EISI) es una comunidad académica agrupada en torno al conocimiento científico y tecnológico en disciplinas de sistemas, computación, informática y afines. El propósito de la Escuela es desarrollar actividades de formación, investigación científica, innovación tecnológica y extensión social, que permitan la apropiación, aplicación, creación, transferencia y divulgación de conocimientos que impacten el entorno local, nacional e internacional. Teniendo como eje articulador la investigación, la Escuela está comprometida con la formación de profesionales integrales con capacidad de concebir y desarrollar proyectos que involucren las disciplinas mencionadas, y de emprender, insertarse y desempeñarse efectivamente en organizaciones generadoras de bienes o servicios (Universidad Industrial de Santander, 2025).

5.2.2.3 Objetivos funcionales. Laboratorio de Desarrollo: Crear un laboratorio de desarrollo que facilite la programación, la formación y la entrega de valor en los proyectos estudiantiles de la Escuela de Ingeniería de Sistemas e Informática, proporcionando un entorno estructurado que potencie las capacidades técnicas y la productividad del equipo de trabajo.

Sistema Centralizado de Gestión de Usuarios: Desarrollar e implementar un sistema centralizado para la gestión de todos los usuarios pertenecientes a la Escuela de Ingeniería de

Sistemas e Informática y a la Universidad Industrial de Santander, de manera que cada aplicación desarrollada pueda acceder a una fuente de información única, universal y consistente.

Estándares de Calidad para el Desarrollo de Software: Definir e implementar estrategias, lineamientos y estándares que orienten el desarrollo de soluciones de software con criterios de calidad, seguridad y escalabilidad, garantizando que cada producto satisfaga plenamente las necesidades y requerimientos para los cuales fue concebido.

5.2.2.4 Objetivos Técnicos. Infraestructura Segura del Laboratorio: Implementar una infraestructura para el laboratorio de desarrollo que cumpla con estándares de seguridad en el acceso, mediante el uso de VPN y cifrado de comunicaciones, sobre los servidores de la Escuela de Ingeniería de Sistemas e Informática.

Repositorio de Código con CI/CD: Implementar el laboratorio de desarrollo como repositorio centralizado de código para todos los proyectos, habilitando capacidades de integración continua, despliegue continuo y automatización para el aprovisionamiento de infraestructura.

Preservación del Desempeño del Sistema: Asegurar que el desempeño óptimo del sistema se mantenga durante la integración de nuevas características, garantizando que las mejoras incorporadas no comprometan la eficiencia ni la calidad de los servicios existentes.

Proveedor Centralizado de Identidad: Desarrollar e implementar un sistema centralizado de gestión y autenticación de usuarios, basado en un proveedor de identidad que pueda integrarse de forma sencilla y estandarizada en todas las aplicaciones desarrolladas dentro del laboratorio.

Coexistencia Armoniosa de Componentes: Garantizar que el sistema mantenga su operatividad y eficiencia durante la integración de nuevas características, asegurando una

coexistencia funcional y armoniosa entre los componentes en producción y los que se encuentran en desarrollo.

5.2.2.5 Contexto de negocio y su descripción funcional. La Escuela de Ingeniería de Sistemas e Informática impulsa iniciativas de desarrollo de software orientadas a resolver necesidades administrativas internas. Estas iniciativas presentan un doble beneficio: por un lado, permiten atender requerimientos reales de la organización mediante soluciones funcionales; por otro, ofrecen a los estudiantes próximos a graduarse la oportunidad de afianzar sus conocimientos aplicándolos en un contexto real, bajo la dirección de la Escuela, y de utilizar dicho desarrollo como proyecto de trabajo de grado.

En este esquema, la Escuela asume el rol de cliente y dirección funcional, definiendo los requisitos de negocio, estableciendo qué debe hacer cada solución y cómo debe comportarse. Sin embargo, la naturaleza académica de estos desarrollos hace necesario complementar esa dirección funcional con un conjunto de estrategias técnicas que orienten a los equipos de estudiantes hacia buenas prácticas de programación, patrones de diseño y estándares de calidad vigentes en la industria tecnológica.

Para ello, la Escuela debe proveer un conjunto de herramientas y servicios de desarrollo compartidos que soporten transversalmente todos los proyectos, entre los que se destacan:

- Un repositorio centralizado de código fuente, que permita gestionar el versionamiento, la colaboración y el historial de cambios de cada proyecto.
- Un sistema centralizado de gestión de usuarios, que elimine la necesidad de que cada aplicación administre su propia base de datos de usuarios, evitando inconsistencias, duplicidad de datos y dificultades en la actualización de la información.

- Estrategias de integración y despliegue continuo, que automaticen los procesos de construcción, validación y puesta en producción del software.

La adopción de estas herramientas permitirá que los equipos conformados por estudiantes comprendan e implementen patrones y prácticas reales del sector tecnológico, garantizando la entrega de software de valor, con criterios de seguridad, mantenibilidad y escalabilidad. Adicionalmente, facilitará la recolección de métricas y estadísticas por proyecto, lo que brindará a la Escuela visibilidad sobre el desempeño y la evolución de cada solución desarrollada.

5.2.2.6 Descripción De La Solución. La solución propuesta consiste en el diseño e implementación de un Sistema Proveedor de Identidad (IdP) unificado para los servicios digitales de la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander, acompañado de la infraestructura de laboratorio de desarrollo necesaria para construirlo, validarlo e integrarlo de forma segura y colaborativa. Estos dos componentes forman una solución cohesionada: el laboratorio no es un entregable accesorio, sino la condición técnica que hace posible que el IdP sea desarrollado con calidad y sostenido en el tiempo.

El IdP permitirá centralizar la gestión y autenticación de todos los usuarios que interactúan con los sistemas digitales de la Escuela, eliminando la necesidad de que cada aplicación administre su propia base de datos de identidades. Mediante protocolos estándar de la industria, cualquier aplicación desarrollada dentro del ecosistema de la Escuela podrá delegar al IdP la responsabilidad de verificar quién es el usuario, qué permisos tiene y desde qué contexto está accediendo, garantizando una experiencia de autenticación consistente, segura y centralizada en todos los servicios.

Sin embargo, un Sistema Proveedor de Identidad no puede desarrollarse ni validarse de forma aislada. Por su naturaleza transversal, el IdP debe ser consumido e integrado simultáneamente por múltiples aplicaciones, lo que hace imprescindible contar con ambientes diferenciados de desarrollo, pruebas y producción. Realizar integraciones o correcciones directamente sobre un ambiente productivo representaría un riesgo crítico: un IdP mal configurado o en proceso de ajuste puede comprometer la autenticación de todos los servicios que dependen de él, con consecuencias irreparables. Los ambientes bajos o de pruebas permiten reproducir escenarios de fallo, validar flujos de autenticación y corregir bugs de integración de manera segura y controlada, antes de cualquier paso a producción.

Para soportar este ciclo de desarrollo, la solución incluye la implementación de un laboratorio de desarrollo compuesto por un repositorio centralizado de código fuente, pipelines de integración y despliegue continuo, y herramientas de aprovisionamiento de infraestructura. Este laboratorio garantiza que los distintos equipos de estudiantes puedan trabajar de forma coordinada sobre el mismo ecosistema, integrarse con el IdP desde sus propios proyectos y detectar incompatibilidades o errores en etapas tempranas del desarrollo, evitando conflictos que serían de difícil trazabilidad si cada equipo operará de manera independiente.

En conjunto, esta solución establece las bases técnicas y operativas que la Escuela necesita para desarrollar software de calidad de forma sostenida: un ecosistema donde los estudiantes apliquen prácticas reales de la industria, donde las identidades de los usuarios se gestionan de forma unificada y segura, y donde cada proyecto pueda integrarse, desplegarse y medirse dentro de un entorno estructurado y confiable.

5.2.2.7 Situación actual (AS-IS). Antes de proponer cualquier cambio arquitectónico, es fundamental comprender en detalle cómo opera hoy la Escuela de Ingeniería de Sistemas e Informática en lo que respecta al desarrollo y despliegue de sus proyectos de software. Este análisis permite documentar el estado actual de los procesos, la infraestructura y las prácticas de desarrollo vigentes, estableciendo el punto de partida sobre el cual se construirá la nueva arquitectura.

La Escuela asume el rol funcional y de negocio en cada proyecto que impulsa: define los requisitos del sistema, establece cómo debe comportarse, revisa entregas intermedias, evalúa prototipos y diseños de interfaz, y decide si aprobar o solicitar ajustes funcionales, visuales o de experiencia de usuario. En este esquema, la Escuela actúa como cliente, y los equipos de desarrollo actúan como proveedores del servicio de construcción del software.

Estos equipos están conformados por estudiantes próximos a graduarse, generalmente entre dos y cinco personas, a quienes se les asigna un proyecto con nombre y requerimientos definidos. A partir de ese momento, cada equipo trabaja de forma completamente autónoma: toma sus propias decisiones técnicas, define sus estándares de código, elige sus tecnologías, establece sus convenciones de diseño visual y gestiona su propio entorno de trabajo sin coordinación técnica con los demás equipos.

Para el manejo del código fuente, los equipos utilizan repositorios gratuitos en la nube a través de GitHub, una plataforma de uso público que, si bien es funcional para proyectos individuales, no está bajo el control ni la supervisión de la Escuela. La compilación del código y la ejecución de pruebas se realizan localmente en las máquinas de los estudiantes, lo que significa que no existe un proceso estandarizado ni automatizado de validación: cada

desarrollador prueba en su propio computador, con su propia configuración, y los resultados pueden variar de un equipo a otro.

Uno de los aspectos más relevantes del modelo actual es la forma en que cada aplicación gestiona la identidad de sus usuarios. En ausencia de un componente compartido entre proyectos, cada equipo de desarrollo implementa de forma independiente su propio sistema de registro, autenticación y autorización de usuarios. Cada aplicación mantiene su propia base de datos de usuarios, con su propio esquema de datos, sus propias reglas de validación de contraseñas, su propio mecanismo de inicio de sesión y su propia lógica de control de acceso.

En la práctica, esto significa que un usuario administrativo que interactúe con varios sistemas de la Escuela gestiona credenciales independientes en cada uno de ellos. Cualquier cambio en su información, como su correo institucional o su rol, debe registrarse por separado en cada aplicación. Del mismo modo, al no existir lineamientos técnicos unificados sobre autenticación y seguridad, cada equipo aplica los estándares que considera apropiados según su criterio y experiencia. Actualmente no existe una capa de identidad compartida entre los sistemas desarrollados en la Escuela.

5.2.2.8 Decisiones arquitectónicas (TO-BE). Con base en el análisis del estado actual, se definen las siguientes decisiones arquitectónicas que guiarán el diseño e implementación de la solución:

- Implementar una infraestructura de red segmentada y segura sobre los servidores existentes de la Escuela, estableciendo un perímetro de acceso controlado.

- Implementar un laboratorio de desarrollo centralizado con ambientes diferenciados de desarrollo, pruebas y producción, con herramientas y estándares compartidos para todos los equipos.
- Adoptar un repositorio centralizado de código fuente con capacidades de integración y despliegue continuo, automatizando los procesos de construcción, validación y entrega de software.
- Incorporar herramientas de calidad, observabilidad y gestión segura de credenciales transversales a todos los proyectos desarrollados en la Escuela.
- Implementar un Sistema Proveedor de Identidad unificado que centralice la gestión y autenticación de usuarios, eliminando la fragmentación actual y sirviendo como base de identidad para todos los servicios digitales de la Escuela.

5.2.2.9 Infraestructura del Laboratorio de Desarrollo. Un laboratorio de desarrollo es un entorno tecnológico centralizado que provee a los equipos de desarrollo las herramientas, los ambientes y los procesos necesarios para construir, validar y entregar software de forma organizada, segura y estandarizada. En lugar de que cada equipo configure y administre su propio entorno de forma independiente, el laboratorio ofrece una infraestructura compartida sobre la cual todos los proyectos operan bajo los mismos lineamientos y con acceso a los mismos recursos.

En el contexto de la Escuela de Ingeniería de Sistemas e Informática, el laboratorio cumple un rol fundamental: es el entorno desde el cual los equipos de estudiantes gestionan su código fuente, automatizan la construcción y despliegue de sus aplicaciones, monitorean su comportamiento y acceden de forma segura a los recursos compartidos. Sin este entorno, cada equipo opera de forma aislada, sin coordinación técnica, sin estándares comunes y sin visibilidad

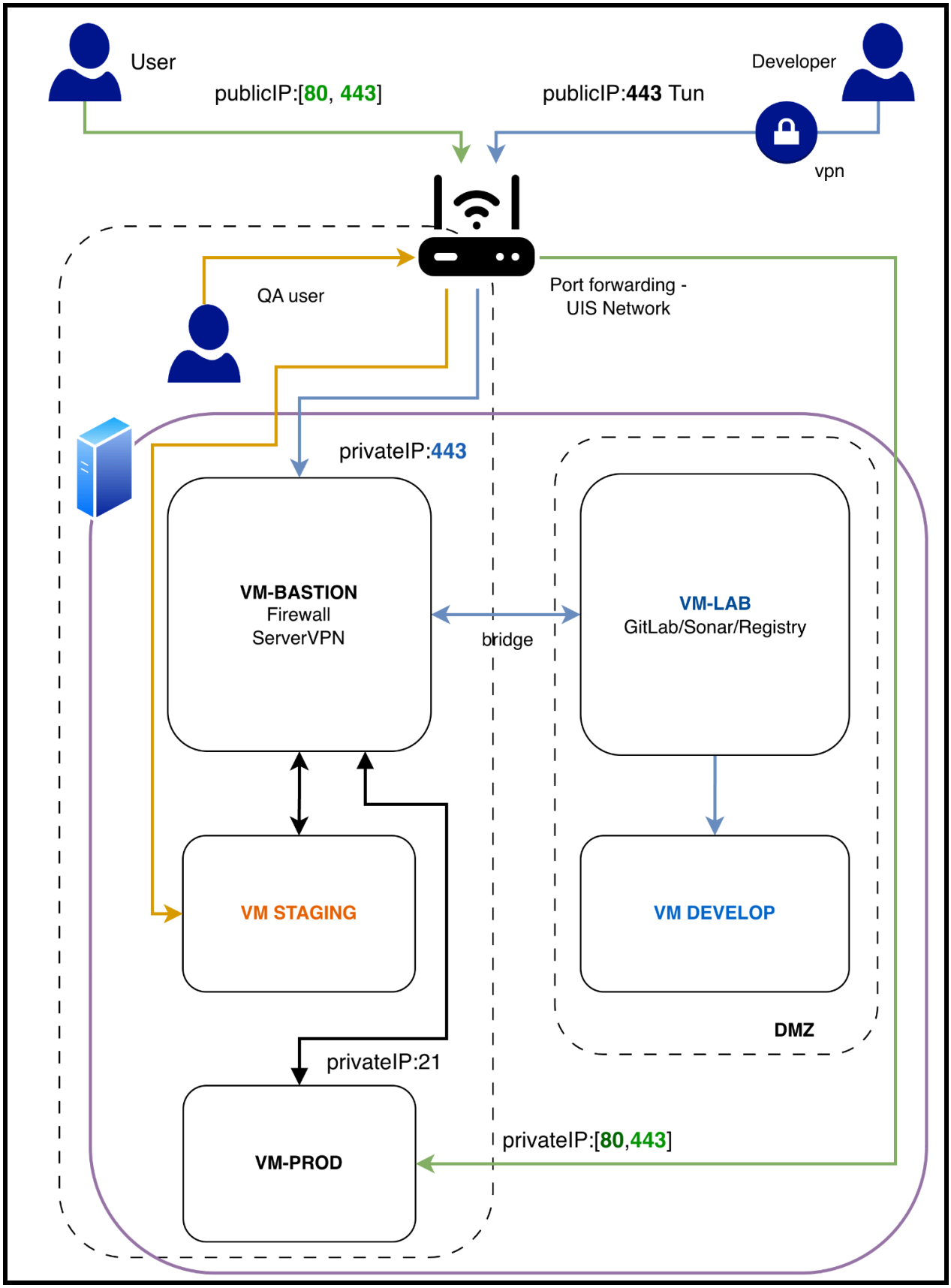
sobre el estado de los proyectos, tal como se describió en el análisis del estado actual.

Adicionalmente, el laboratorio es el prerrequisito técnico indispensable para el desarrollo del Sistema Proveedor de Identidad. Al ser el IdP un servicio transversal que debe integrarse con múltiples aplicaciones simultáneamente, su construcción y validación requieren ambientes diferenciados y controlados donde las integraciones puedan probarse de forma segura antes de llegar a producción.

5.2.2.10 Implementación. A continuación la arquitectura de implementación:

Figura 1

Arquitectura, flujo y máquinas virtuales


User
publicIP:[80, 443]
publicIP:443 Tun
Developer
vpn
QA user
Port forwarding -
UIS Network
privateIP:443
VM-BASTION
Firewall
ServerVPN
bridge
VM-LAB
GitLab/Sonar/Registry
VM STAGING
VM DEVELOP
DMZ
privateIP:21
VM-PROD
privateIP:[80,443]

Para el diseño del laboratorio de desarrollo se aprovechará la infraestructura física con la que ya cuenta la Escuela de Ingeniería de Sistemas e Informática, combinando el uso de máquinas virtuales, software especializado y una topología de redes definida, con el objetivo de construir un entorno seguro, organizado y funcional para todos los equipos de desarrollo.

El primer elemento a definir es la segmentación de redes sobre la cual operará todo el laboratorio. Esta separación es la base de la seguridad del entorno, ya que determina con precisión quién puede acceder a qué recursos y bajo qué condiciones.

5.2.2.10.1 ***LAN.*** Es una red privada virtual creada y administrada directamente dentro de Proxmox. Esta red existe únicamente en el contexto de los servidores de la Escuela y no tiene ninguna conectividad directa con la red de la universidad ni con internet. En ella residirán los recursos más críticos y sensibles del laboratorio, inaccesibles para cualquier usuario externo que no cuente con los mecanismos de acceso que se describen más adelante.

5.2.2.10.2 ***WAN.*** Corresponde a la red interna de la universidad, accesible para estudiantes y administrativos desde cualquier punto de las instalaciones universitarias. Aunque técnicamente es una red privada institucional, se denomina WAN dentro de la arquitectura del laboratorio porque representa el entorno externo a él: es la red desde la cual operan los usuarios y la puerta de comunicación entre el laboratorio y el resto de la universidad. Esta red no se crea, sino que se redefine su nombre para claridad en el contexto de este diseño.

5.2.2.10.3 ***VM-Bastion.*** Bastion es la máquina virtual que hace posible la separación entre ambas redes y actúa como el único punto de entrada al laboratorio. Es el único componente de toda la arquitectura que tiene presencia simultánea en las dos redes: cuenta con una interfaz conectada a la WAN y otra conectada a la LAN, lo que la convierte en el guardián y regulador de

todo el tráfico que intenta entrar o salir del laboratorio.

Para cumplir este rol, Bastion ejecuta OPNsense, un sistema operativo de código abierto especializado en funciones de firewall y enrutamiento de red. Un firewall es un sistema que inspecciona todo el tráfico de red entrante y saliente, y decide si permitirlo o bloquearlo según reglas definidas por el administrador. OPNsense permite configurar estas reglas con gran nivel de detalle, especificando qué tipo de tráfico, desde qué origen y hacia qué destino está permitido, bloqueando por defecto todo lo que no esté explícitamente autorizado.

Sobre esta base, Bastion implementa una VPN mediante OpenVPN. Una VPN, o red privada virtual, es una tecnología que establece un canal de comunicación cifrado entre el dispositivo del usuario y el servidor, de manera que todo el tráfico que viaja por ese canal es ilegible para cualquier tercero que intente interceptar. En términos prácticos, cuando un desarrollador activa su cliente VPN e ingresa sus credenciales, su dispositivo se comporta como si estuviera físicamente dentro de la LAN del laboratorio, pudiendo acceder a todos sus recursos de forma segura desde cualquier lugar. Sin este túnel activo, cualquier intento de alcanzar la LAN es bloqueado automáticamente por el firewall, sin excepción.

Este diseño centraliza el control de acceso en un único punto. Si un integrante abandona un equipo de desarrollo o si sus credenciales se ven comprometidas, el acceso puede revocarse de forma inmediata desde Bastion sin necesidad de modificar ningún otro componente del sistema.

5.2.2.10.4 ***VM-Lab.*** Lab es el núcleo operativo del laboratorio y el corazón de todas las herramientas de desarrollo. Al estar conectada exclusivamente a la LAN, sólo es accesible a través de la VPN gestionada por Bastion, garantizando que únicamente los miembros autorizados

pueden interactuar con ella. En esta máquina se alojan todos los servicios centrales que soportan el ciclo de vida del software en la Escuela.

El servicio principal GitLab actúa como repositorio centralizado de código fuente bajo el control directo de la Escuela, reemplazando el uso de repositorios públicos externos. Además de gestionar el versionamiento y la colaboración sobre el código, GitLab es el motor de los pipelines de integración y despliegue continuo, que automatizan los procesos de construcción, prueba y despliegue de las aplicaciones cada vez que un desarrollador sube cambios al repositorio.

Se integra a estos pipelines para realizar análisis estáticos de calidad del código de forma automática en cada entrega, identificando errores, vulnerabilidades de seguridad, código duplicado y malas prácticas antes de que lleguen a ambientes productivos haciendo uso del servicio SonarQube.

Se centraliza el almacenamiento seguro de información sensible como contraseñas, claves de acceso a bases de datos, tokens de autenticación y certificados. Todos los equipos consumen sus credenciales desde este repositorio seguro y auditado, en lugar de gestionarlas de forma individual. Este servicio es el Vault de HashiCorp.

El servicio de Grafana proporciona monitoreo y observabilidad en tiempo real sobre el estado de todas las aplicaciones y la infraestructura del laboratorio, permitiendo detectar caídas, degradaciones de rendimiento o comportamientos anómalos de forma oportuna. Complementan este entorno una wiki de documentación y tableros de gestión de tareas en formato Kanban, que permiten a cada equipo documentar sus decisiones técnicas y hacer seguimiento organizado a su trabajo, con visibilidad para la Escuela.

5.2.2.10.5 ***VM-Dev.*** Es el primer ambiente de despliegue real dentro del laboratorio. Conectada exclusivamente a la LAN y accesible únicamente mediante VPN, ofrece a los equipos de desarrollo un entorno controlado y privado donde desplegar sus aplicaciones en condiciones similares a las productivas, sin los riesgos de operar sobre un sistema en uso real.

En este ambiente los desarrolladores despliegan sus aplicaciones a través de los pipelines configurados en GitLab, realizan integraciones con el Sistema Proveedor de Identidad, validan el comportamiento de sus componentes y observan métricas de desempeño a través de Grafana. Cualquier error de integración o comportamiento inesperado puede identificarse y corregirse aquí antes de avanzar a las siguientes fases.

5.2.2.10.6 ***VM-Staging.*** Staging cumple un rol diferente al de Dev: está orientada a los usuarios administrativos de la Escuela que deben validar funcionalmente las aplicaciones antes de aprobar su paso a producción. A diferencia de los ambientes anteriores, esta máquina está conectada a la WAN, lo que permite que los administrativos accedan a ella directamente desde cualquier dispositivo en la red universitaria, sin necesidad de VPN ni configuración adicional.

Para que Staging pueda descargar desde el laboratorio las imágenes contenerizadas de las aplicaciones que necesita desplegar, se configura en Bastión una regla de excepción acotada: se permite el tráfico desde la dirección IP específica de la máquina Staging hacia la LAN, sin requerir VPN. Esta es la única excepción al principio de aislamiento total de la LAN, y está limitada de forma precisa a ese único origen.

5.2.2.10.7 ***VM-Producción.*** Producción es el destino final de las aplicaciones que han superado las fases de desarrollo y validación. Aquí se despliegan los sistemas que serán

utilizados por los usuarios reales de la Escuela y la universidad. Esta máquina está conectada a la WAN y puede exponerse opcionalmente a internet según lo requiera cada aplicación.

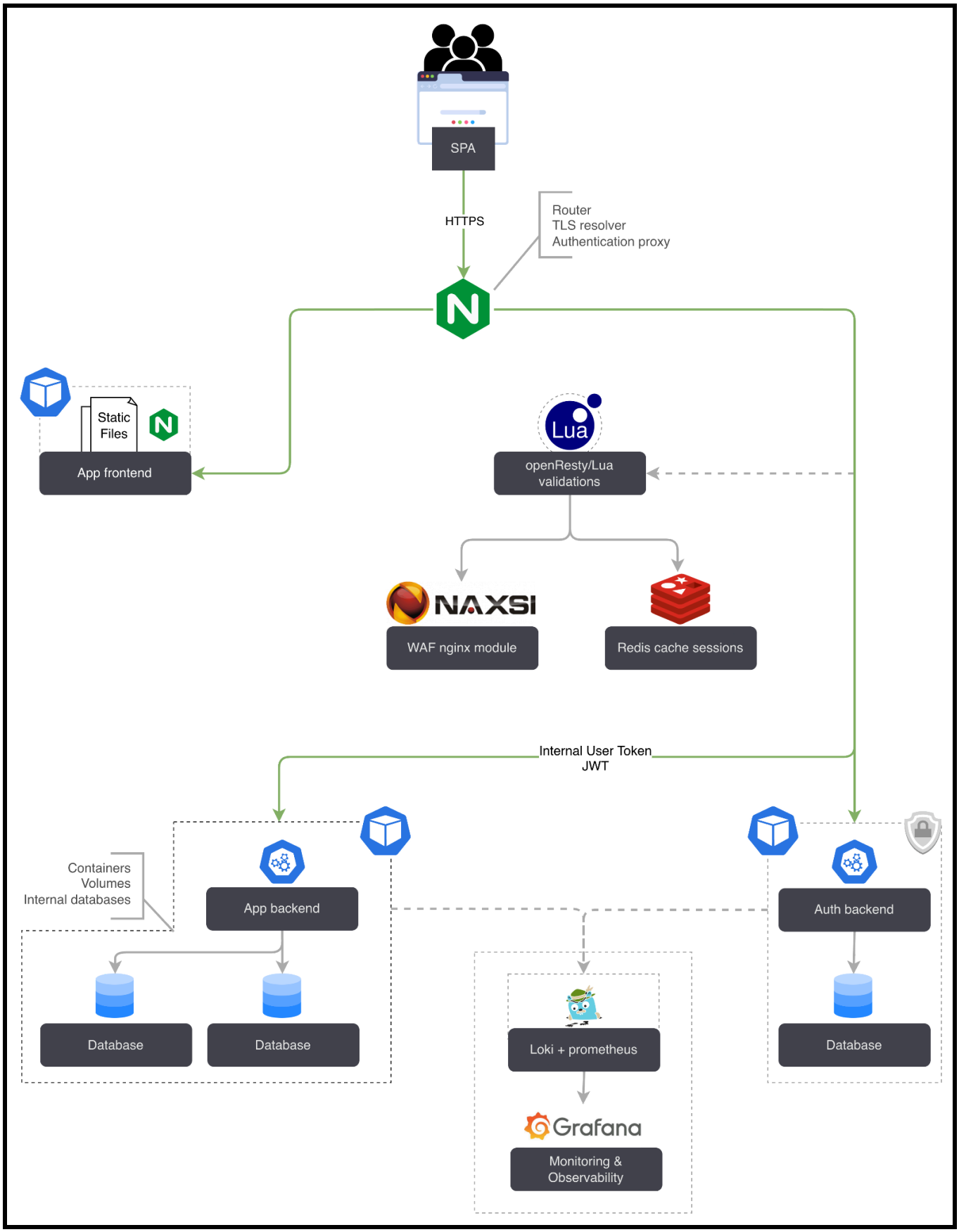
Para habilitar el acceso desde internet, se solicita al área de redes de la universidad la asignación de una dirección IP pública con los puertos 80 y 443 habilitados, correspondientes a los protocolos HTTP y HTTPS. Se configura entonces un port forwarding que redirige el tráfico entrante desde esa IP pública hacia la IP privada de la máquina de producción. Cada aplicación desplegada debe contar además con un dominio propio registrado en el DNS institucional, apuntando a esa IP pública, para que los usuarios puedan acceder a través de una dirección web reconocible.

Con este diseño, la Escuela dispone de un ecosistema de desarrollo estructurado en el que cada ambiente cumple un propósito específico, el acceso está controlado en cada nivel y los equipos pueden enfocarse en construir software de calidad sin asumir la carga de administrar infraestructura de forma improvisada.

Figura 2

Diagrama de aplicaciones y servicios.

SISTEMA PROVEEDOR DE IDENTIDAD



5.2.3 *Prototipo de interfaces de usuario*

A lo largo del desarrollo del proyecto se diseñaron y validaron diferentes interfaces de usuario orientadas a dos tipos de actores principales: usuarios administradores (o desarrolladores) y usuarios regulares que interactúan con los sistemas integrados al Proveedor de Identidad (IdP).

En el caso de los usuarios administradores, las interfaces presentadas en las Figuras 31 a 77 corresponden a funcionalidades avanzadas relacionadas con la gestión del sistema de identidad y la infraestructura de desarrollo. Estas interfaces incluyen paneles de administración del IdP, configuración de clientes, gestión de usuarios y roles, así como herramientas de integración continua y despliegue en plataformas como GitLab. El diseño de estas interfaces prioriza el acceso a configuraciones críticas del sistema, permitiendo a los administradores gestionar de forma centralizada los procesos de autenticación y autorización.

Adicionalmente, estas interfaces fueron diseñadas considerando principios de usabilidad y organización jerárquica de la información, facilitando la navegación entre módulos y reduciendo la complejidad operativa para los desarrolladores. Esto resulta clave en un entorno donde múltiples servicios y configuraciones deben ser gestionados de manera consistente.

Por otro lado, para los usuarios regulares, las interfaces mostradas en las Figuras 78 a 80 corresponden a los flujos esenciales de autenticación dentro del sistema. Estas incluyen las pantallas de inicio de sesión, autenticación centralizada y redirección hacia los servicios integrados. El diseño de estas interfaces se centró en la simplicidad y claridad, garantizando una

experiencia de usuario intuitiva y reduciendo la fricción en el proceso de autenticación.

Estas interfaces cumplen un rol fundamental en la validación del sistema, ya que permiten comprobar el correcto funcionamiento de los flujos de inicio de sesión, el uso de Single Sign-On (SSO) y la interacción entre el usuario y el proveedor de identidad. Asimismo, evidencian la separación clara entre las funcionalidades administrativas y las de usuario final, lo cual es un principio clave en el diseño del sistema propuesto.

5.3 Fase de Construcción

5.3.1 *Sistema Proveedor de Identidad*

Un Sistema Proveedor de Identidad, conocido por sus siglas en inglés como IdP (Identity Provider), es un servicio centralizado encargado de gestionar la identidad digital de los usuarios dentro de un ecosistema de aplicaciones. Su función principal es verificar quién es el usuario, qué permisos tiene y bajo qué condiciones puede acceder a cada recurso, delegando esta responsabilidad fuera de cada aplicación individual y concentrándose en un único componente especializado. Esto significa que las aplicaciones que se integran con el IdP no necesitan implementar su propia lógica de autenticación ni mantener su propia base de datos de usuarios: simplemente consultan al IdP y confían en su respuesta.

En la industria tecnológica, este modelo es ampliamente adoptado porque resuelve de forma estructural uno de los problemas más comunes en ecosistemas con múltiples aplicaciones: la fragmentación de identidades. Cuando cada aplicación gestiona sus propios usuarios de forma independiente, el resultado es un conjunto de islas de información desconectadas entre sí, con

todos los problemas de inconsistencia, duplicidad y seguridad que esto conlleva, tal como se describió en el análisis del estado actual de la Escuela.

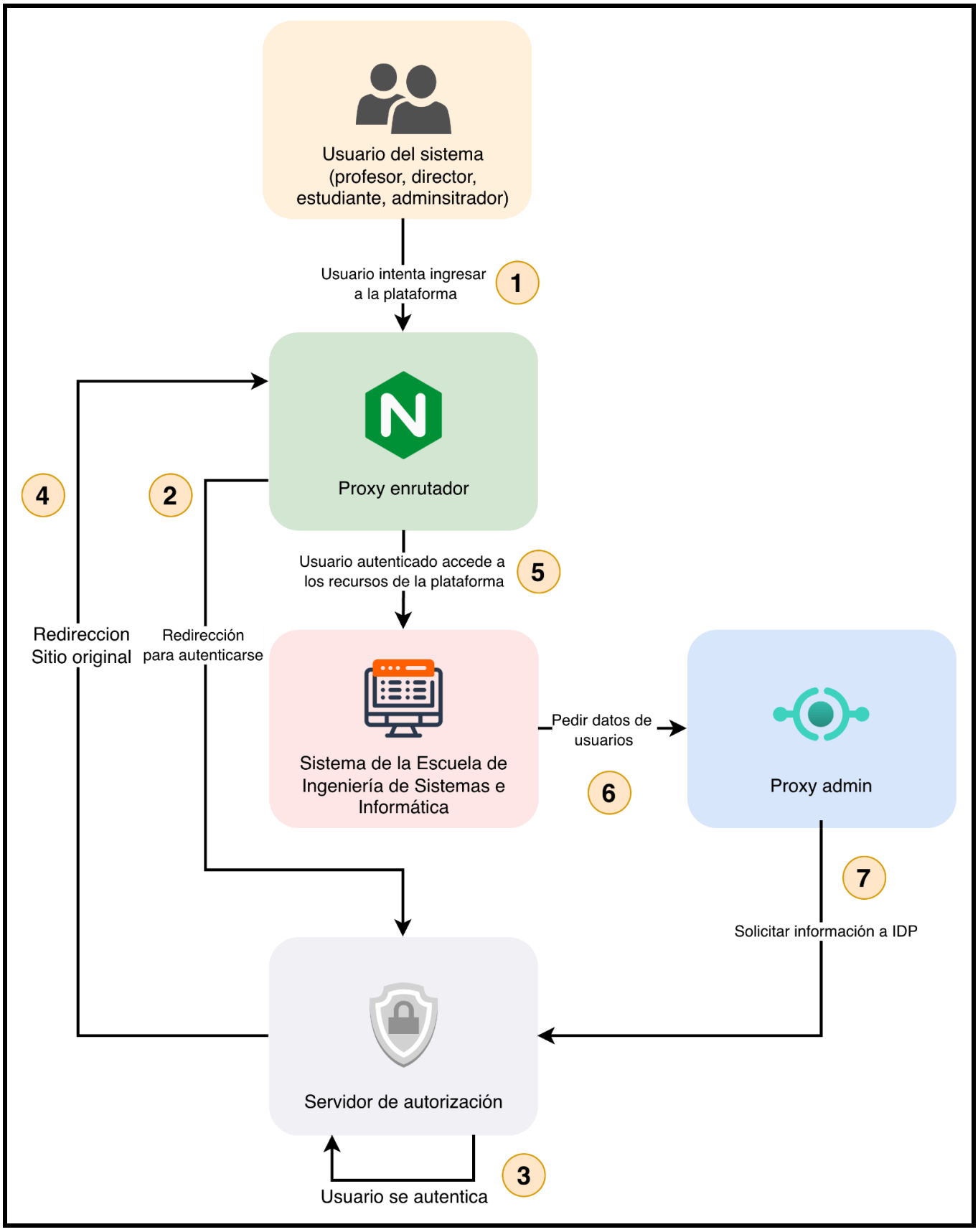
En el contexto de la Escuela de Ingeniería de Sistemas e Informática, la necesidad de un IdP es directa: actualmente cada sistema desarrollado por los equipos de estudiantes mantiene su propia base de datos de usuarios, su propia lógica de autenticación y sus propios criterios de seguridad. Con la implementación del IdP, todos los sistemas desarrollados en la Escuela contarán con una fuente única y centralizada de identidad, desde la cual se gestionan los usuarios, sus roles y sus permisos de forma consistente y segura para todas las aplicaciones.

El IdP se despliega sobre el laboratorio de desarrollo implementado en la fase anterior, aprovechando los ambientes diferenciados de desarrollo, pruebas y producción para construirlo, validarlo e integrarlo de forma controlada. Esta relación de dependencia entre ambos componentes es la que justifica que el laboratorio sea el primer entregable de la solución: sin los ambientes que este provee, no es posible desarrollar ni integrar el IdP de forma segura y coordinada entre los distintos equipos.

Figura 3

Flujo de acceso con el sistema proveedor de identidad

SISTEMA PROVEEDOR DE IDENTIDAD



El flujo de autenticación inicia cuando un usuario accede desde su navegador a cualquier plataforma web de la Escuela. El usuario puede llegar a través de un enlace directo o escribiendo manualmente la dirección en el navegador, por ejemplo saam.uis.edu.co. En ese momento, el navegador realiza una consulta al DNS (Domain Name Service), que es el servicio encargado de traducir un nombre de dominio legible como saam.uis.edu.co en la dirección IP del servidor donde está alojada la aplicación, de la misma forma en que una guía telefónica traduce un nombre en un número.

Sin embargo, antes de que el usuario pueda acceder a la aplicación solicitada, el sistema detecta que no existe una sesión activa y lo redirige automáticamente a auth.uis.edu.co, que es el dominio del Sistema Proveedor de Identidad. Aquí el usuario deberá ingresar sus credenciales institucionales: usuario y contraseña, y opcionalmente un código de verificación enviado a su correo, como segunda capa de seguridad. Este mecanismo se conoce como autenticación de múltiple factor o MFA (Multi-Factor Authentication).

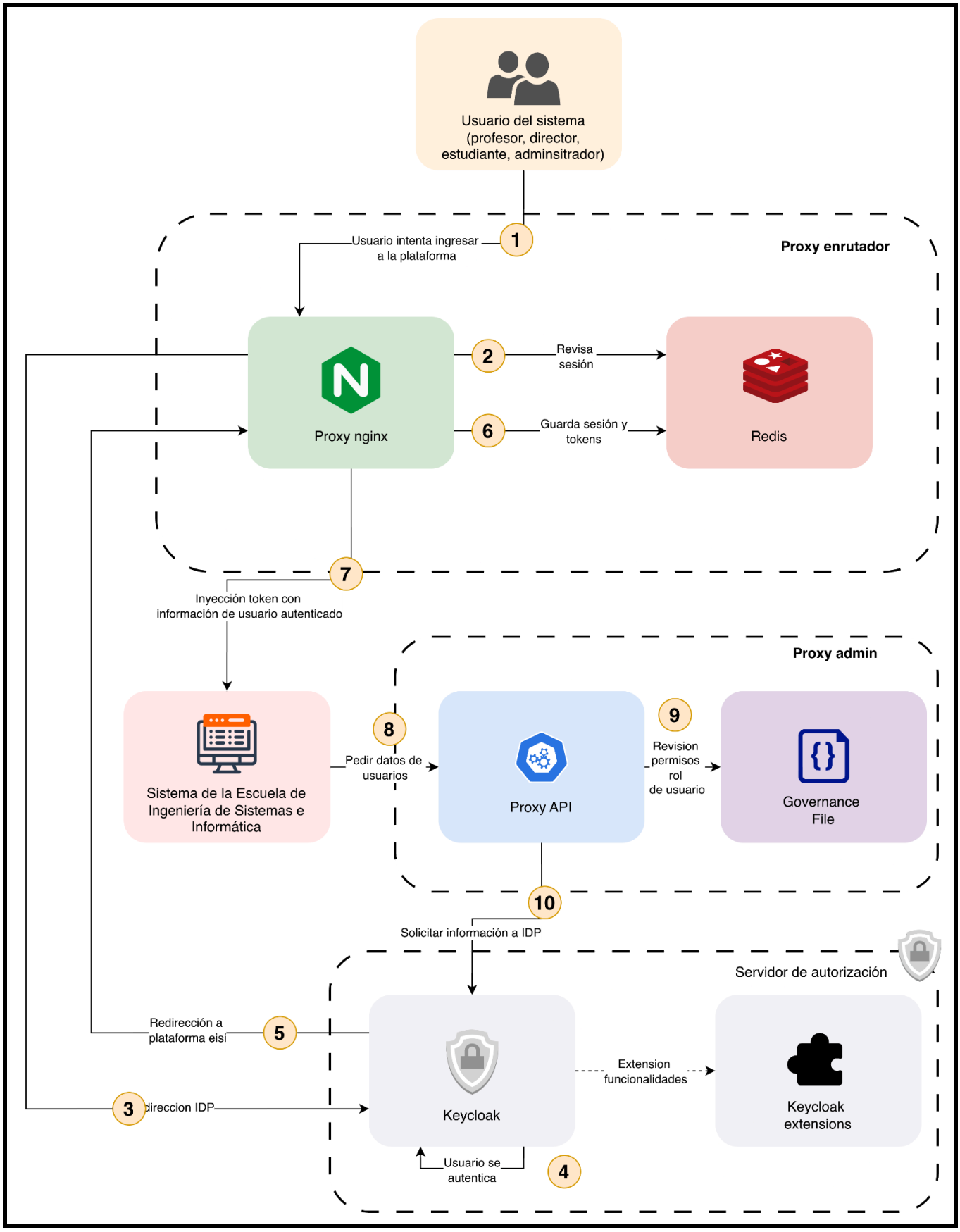
Una vez que el IdP verifica exitosamente la identidad del usuario, lo dirige de vuelta a la aplicación original saam.uis.edu.co, donde ya puede operar con normalidad. A partir de ese momento, la aplicación puede necesitar consultar información adicional sobre otros usuarios, como sus datos, roles o permisos, para ello ejecuta peticiones HTTPS REST hacia un proxy administrativo, que es un componente intermediario con permisos elevados de consulta sobre el IdP. Este proxy recibe la solicitud de la aplicación, la ejecuta de forma segura contra el IdP y devuelve los resultados, evitando que las aplicaciones tengan acceso directo y sin restricciones a la administración del sistema de identidad.

Todo este proceso, desde la autenticación inicial hasta las consultas administrativas, queda registrado y monitoreado centralmente por el IdP, proporcionando a la Escuela trazabilidad completa sobre quién accede a qué sistema y en qué momento.

Figura 4

Flujo de autenticación general

SISTEMA PROVEEDOR DE IDENTIDAD



Para que el flujo de autenticación funcione de forma centralizada y transparente para el usuario, todas las aplicaciones de la Escuela deben estar desplegadas detrás de un único proxy enrutador, implementado con Nginx. Este componente es el primero en recibir cualquier petición entrante, independientemente de si proviene del frontend o del backend de una aplicación, y es el responsable de decidir si el usuario puede continuar o debe autenticarse primero. Cuando una petición llega al proxy, Nginx consulta inmediatamente a Redis, un almacén de datos en memoria de alta velocidad, para verificar si existe una cookie de sesión válida asociada a esa petición. Si la cookie no existe o ha expirado, Nginx redirige automáticamente al usuario hacia Keycloak, el servidor de autorización del IdP, donde deberá ingresar sus credenciales. Una vez que Keycloak auténtica exitosamente al usuario, lo dirige de vuelta a la aplicación original, por ejemplo saam.uis.edu.co. Como esta URL está detrás del proxy, es Nginx quien intercepta esa redirección: recibe el código de autorización emitido por Keycloak, lo intercambia por los tokens de acceso correspondientes, genera una cookie de sesión y la almacena en Redis. A partir de ese momento, el usuario queda autenticado y Nginx le permite continuar hacia el frontend de la aplicación.

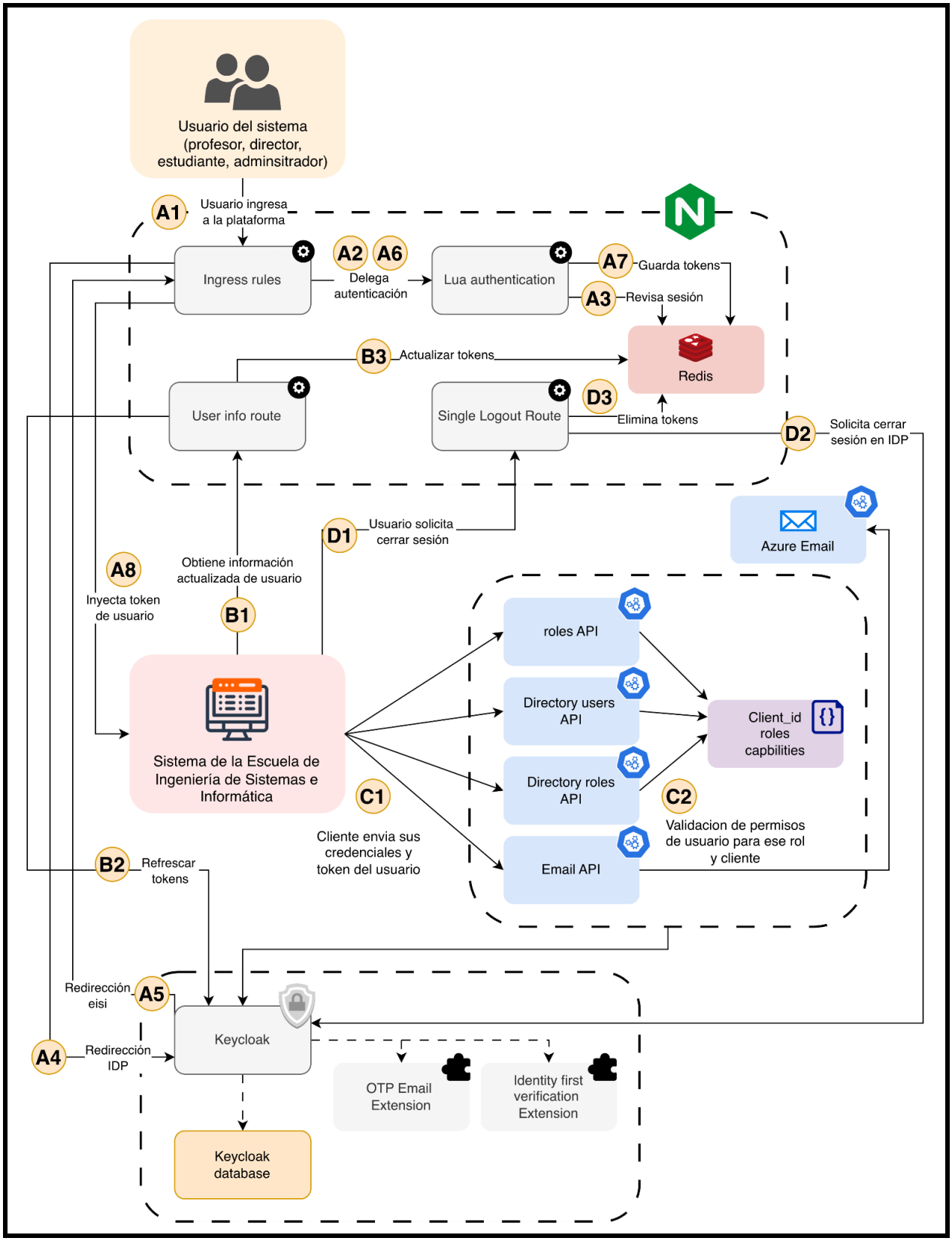
Cuando el frontend cargado en el navegador realiza peticiones hacia su backend, por ejemplo a saam.uis.edu.co/api, Nginx identifica que se trata de una petición a un servicio backend y, dado que comparte el mismo dominio, verifica que la cookie de sesión ya existe en Redis. En ese caso, en lugar de redirigir a Keycloak, Nginx inyecta directamente en la cabecera de la petición un token interno que contiene la información del usuario autenticado: su nombre, su identificador, sus roles y demás atributos relevantes. De esta forma, el backend recibe cada petición ya enriquecida con la identidad del usuario, sin necesidad de implementar ninguna lógica de autenticación propia, simplemente lee el token inyectado por el proxy y sabe con

certeza quién está haciendo la solicitud. Para las consultas administrativas sobre usuarios, como listar usuarios de un rol, obtener información de un perfil o consultar el directorio de la Escuela, las aplicaciones no se comunican directamente con Keycloak sino a través del proxy admin. Este componente expone un conjunto de endpoints REST definidos con capacidades específicas, como read:users o directory:read, que representan operaciones concretas sobre el directorio de usuarios. El control de acceso a estas capacidades se gestiona mediante un Governance File, un archivo YAML donde se define, por cada aplicación registrada mediante su client_id, qué roles de Keycloak tienen acceso a qué capacidades. Por ejemplo, se puede definir que dentro de la aplicación saam, únicamente el rol saam-admin tiene la capacidad directory:read, impidiendo que roles como el de estudiante puedan ejecutar consultas reservadas para administradores. Cuando una aplicación necesita consultar información de usuarios, su backend envía al proxy admin el token interno que recibió del proxy enrutador, junto con sus credenciales de cliente. El proxy admin valida esas credenciales, localiza el Governance File correspondiente al client_id de la aplicación, e inspecciona el token para determinar qué rol tiene el usuario autenticado. Con esa información, verifica si ese rol posee la capacidad requerida para ejecutar la operación solicitada. Si la verificación es exitosa, el proxy admin consulta a Keycloak y devuelve los resultados a la aplicación. Si no, la petición es rechazada. De esta manera, el control de acceso a la información del directorio de usuarios queda centralizado y gobernado por un único punto de control, sin depender del criterio de implementación de cada equipo de desarrollo.

Figura 5

Flujos de autenticación detallados

SISTEMA PROVEEDOR DE IDENTIDAD



El diagrama de componentes detalla el funcionamiento interno de cada contenedor del sistema, mostrando cómo se orquesta el flujo completo de autenticación, refresco de sesión, consultas administrativas y cierre de sesión. Para mayor claridad, cada flujo se identifica con una letra que agrupa los pasos relacionados.

Flujo A — Autenticación inicial: Cuando el usuario intenta ingresar a una plataforma (A1), la petición es recibida por las Ingress Rules de Nginx, que actúan como punto de entrada y determina hacia qué componente interno debe enrutar cada solicitud según su naturaleza. Las Ingress Rules delegan la decisión de autenticación al módulo de Lua Authentication (A2), un script embebido en Nginx que contiene toda la lógica de validación de sesión. Este módulo consulta inmediatamente a Redis (A3) para verificar si existe una cookie de sesión válida asociada a la petición entrante.

Si la cookie no existe o ha expirado, Lua Authentication redirige al usuario hacia Keycloak (A4), el servidor de autorización del IdP, donde deberá ingresar sus credenciales. Keycloak gestiona el proceso de autenticación, que puede incluir verificación de contraseña y un código OTP enviado al correo del usuario a través de Azure Email, gestionado por la extensión OTP Email Extension. Adicionalmente, la extensión Identity First Verification permite separar el paso de identificación del usuario del paso de verificación de contraseña, mejorando la experiencia de autenticación. Una vez autenticado, Keycloak registra la sesión en su propia base de datos y redirige al usuario de vuelta a la plataforma original (A5).

Como la plataforma está detrás del proxy, Nginx intercepta esa redirección: Lua Authentication recibe el código de autorización emitido por Keycloak, lo intercambia por los tokens de acceso (A6), y los almacena en Redis junto con la cookie de sesión generada (A7). A

partir de ese momento, en cada petición subsecuente, Nginx obtiene la información actualizada del usuario desde Redis (A8) e inyecta un token interno en la cabecera de la petición con los datos del usuario autenticado: nombre, identificador, roles y demás atributos relevantes. De esta forma, el backend de cada aplicación recibe la identidad del usuario de forma automática y sin necesidad de implementar ninguna lógica de autenticación propia.

Flujo B — Refresco de tokens: En situaciones donde una aplicación detecta que la información del usuario puede estar desactualizada, por ejemplo tras un cambio de rol o una actualización de perfil, puede invocar el endpoint especial de refresco (B1). Este endpoint hace que Nginx solicite a Keycloak tokens actualizados (B2), obtiene la información más reciente del usuario directamente desde el IdP, y actualiza los tokens almacenados en Redis (B3). A partir de ese momento, todas las peticiones subsecuentes recibirán el token inyectado con la información actualizada, garantizando que el backend siempre opere con datos vigentes sin requerir que el usuario cierre e inicie sesión nuevamente.

Flujo C — Consultas administrativas: Cuando una aplicación necesita consultar información del directorio de usuarios (C1), como listar usuarios de un rol, obtener perfiles o consultar roles disponibles, la petición se dirige al proxy admin, que expone cuatro APIs especializadas: Roles API, Directory Users API, Directory Roles API y Email API. Antes de ejecutar cualquier consulta, el proxy admin valida los permisos del solicitante consultando el archivo Client ID Roles Capabilities (C2), que es el Governance File en formato YAML donde se define qué roles de cada aplicación tienen acceso a qué capacidades. Solo si el rol del usuario autenticado posee la capacidad requerida para esa operación, el proxy administra y procede a consultar a Keycloak y devuelve los resultados. De lo contrario, la petición es rechazada,

garantizando que operaciones sensibles como la consulta del directorio completo de usuarios solo sean ejecutables por los roles autorizados explícitamente para ello.

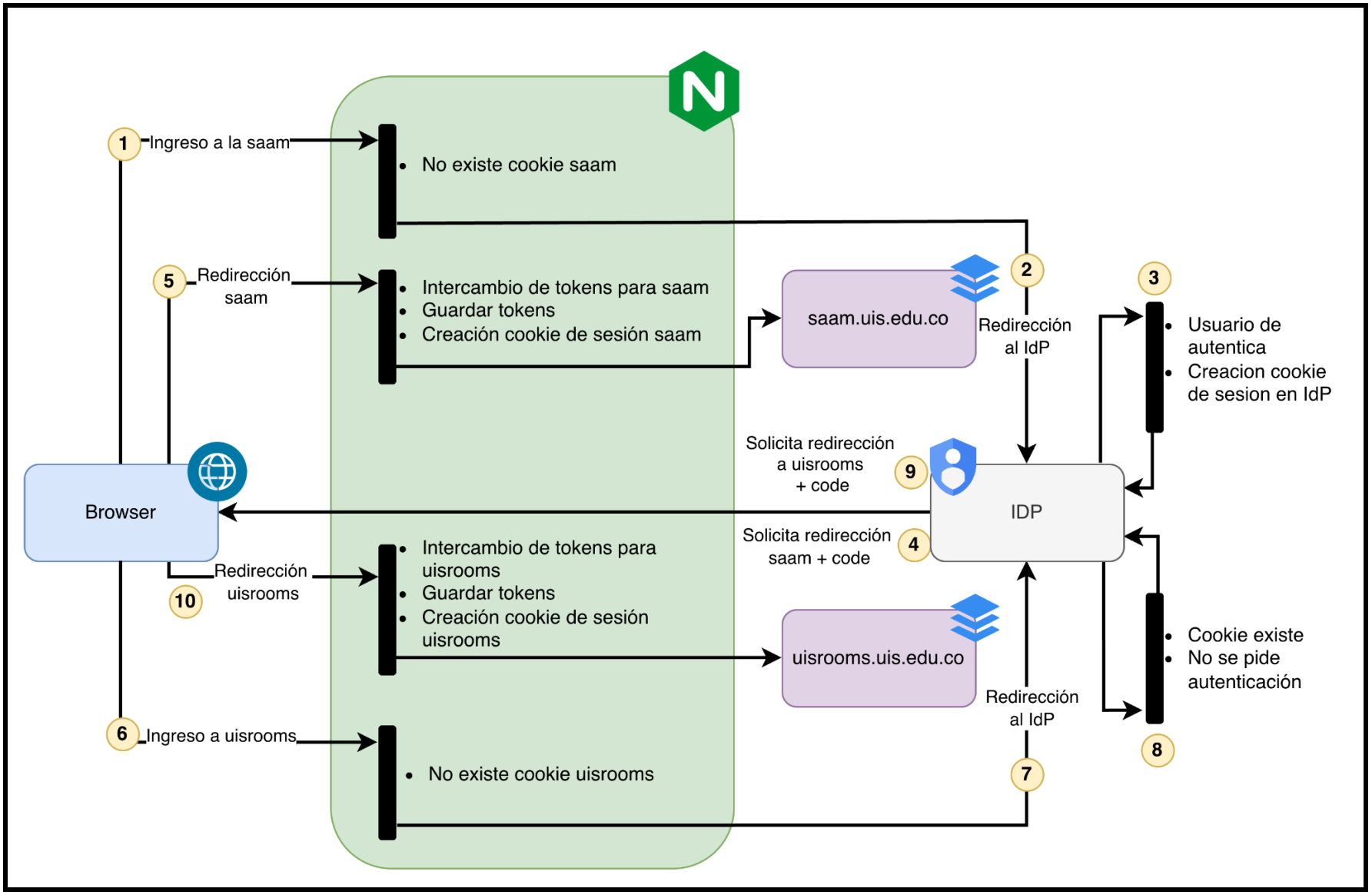
Flujo D — Cierre de sesión: Cuando el usuario solicita cerrar sesión (D1), la petición es recibida por la Single Logout Route de Nginx, un endpoint dedicado exclusivamente a gestionar este proceso. Este componente solicita a Keycloak que cierre la sesión activa del usuario en el IdP (D2) y, de forma simultánea, elimina de Redis todos los tokens y la cookie de sesión asociados a ese usuario (D3). Una vez completado este proceso, el usuario es redirigido automáticamente a la pantalla de autenticación de KeyCloak, ya que no cuenta con ninguna sesión activa que le permita acceder a las plataformas de la Escuela.

5.3.1.1 Single Sign-On (SSO). A continuación la implementación del SSO:

Figura 6

Flujo de autenticación en la implementación del SSO

SISTEMA PROVEEDOR DE IDENTIDAD



Una de las capacidades más importantes que habilita el Sistema Proveedor de Identidad centralizado es el Single Sign-On, conocido por sus siglas SSO. Este mecanismo permite que un usuario que ya se autenticó en una aplicación de la Escuela pueda acceder a cualquier otra sin necesidad de ingresar sus credenciales nuevamente, ofreciendo una experiencia fluida y transparente entre todos los sistemas del ecosistema.

Para entender por qué esto funciona, es importante comprender cómo están diseñadas las sesiones en esta arquitectura. Cada aplicación gestiona su propia cookie de sesión de forma independiente: la cookie de saam.uis.edu.co es exclusiva de ese dominio y la cookie de uisrooms.uis.edu.co es exclusiva del suyo. Sin embargo, Keycloak mantiene su propia sesión global en auth.uis.edu.co, completamente separada de las sesiones de cada aplicación. Es precisamente esta sesión global la que hace posible el SSO.

Primer acceso, autenticación en SAAM: Cuando el usuario intenta ingresar por primera vez a saam.uis.edu.co (paso 1), el proxy Nginx verifica en Redis si existe una cookie de sesión válida para ese dominio. Al no encontrarla, redirige al usuario hacia el IdP en auth.uis.edu.co (paso 2). Aquí el usuario ingresa sus credenciales y completa el proceso de autenticación, incluyendo el factor adicional de verificación si está configurado (paso 3). Keycloak valida las credenciales, crea una sesión global en su propio almacén y redirige al usuario de vuelta a saam.uis.edu.co acompañado de un código de autorización (paso 4). El proxy Nginx intercepta esta redirección, intercambia el código por los tokens de acceso, los almacena en Redis y genera la cookie de sesión específica para saam.uis.edu.co (paso 5). A partir de ese momento el usuario puede operar con normalidad en SAAM.

Segundo acceso, SSO en UisRooms: Cuando ese mismo usuario, sin haber cerrado sesión, intenta ingresar a uisrooms.uis.edu.co (paso 6), el proxy Nginx verifica en Redis si existe una cookie de sesión para ese dominio. Como el usuario nunca ha accedido a UisRooms antes, esta cookie no existe, y Nginx lo redirige nuevamente hacia el IdP (pasos 7 y 8). Aquí es donde ocurre la magia del SSO: Keycloak recibe la solicitud de autenticación para UisRooms, pero en lugar de presentarle al usuario el formulario de credenciales, detecta que ya existe una sesión global activa para ese usuario, creada cuando se autenticó en SAAM. Con esa sesión vigente, Keycloak emite directamente un nuevo código de autorización para UisRooms y redirige al usuario de vuelta sin haberle pedido ningún dato (paso 9). El proxy Nginx recibe esta redirección, repite el proceso de intercambio de tokens, crea la cookie de sesión específica para uisrooms.uis.edu.co y le permite el acceso (paso 10).

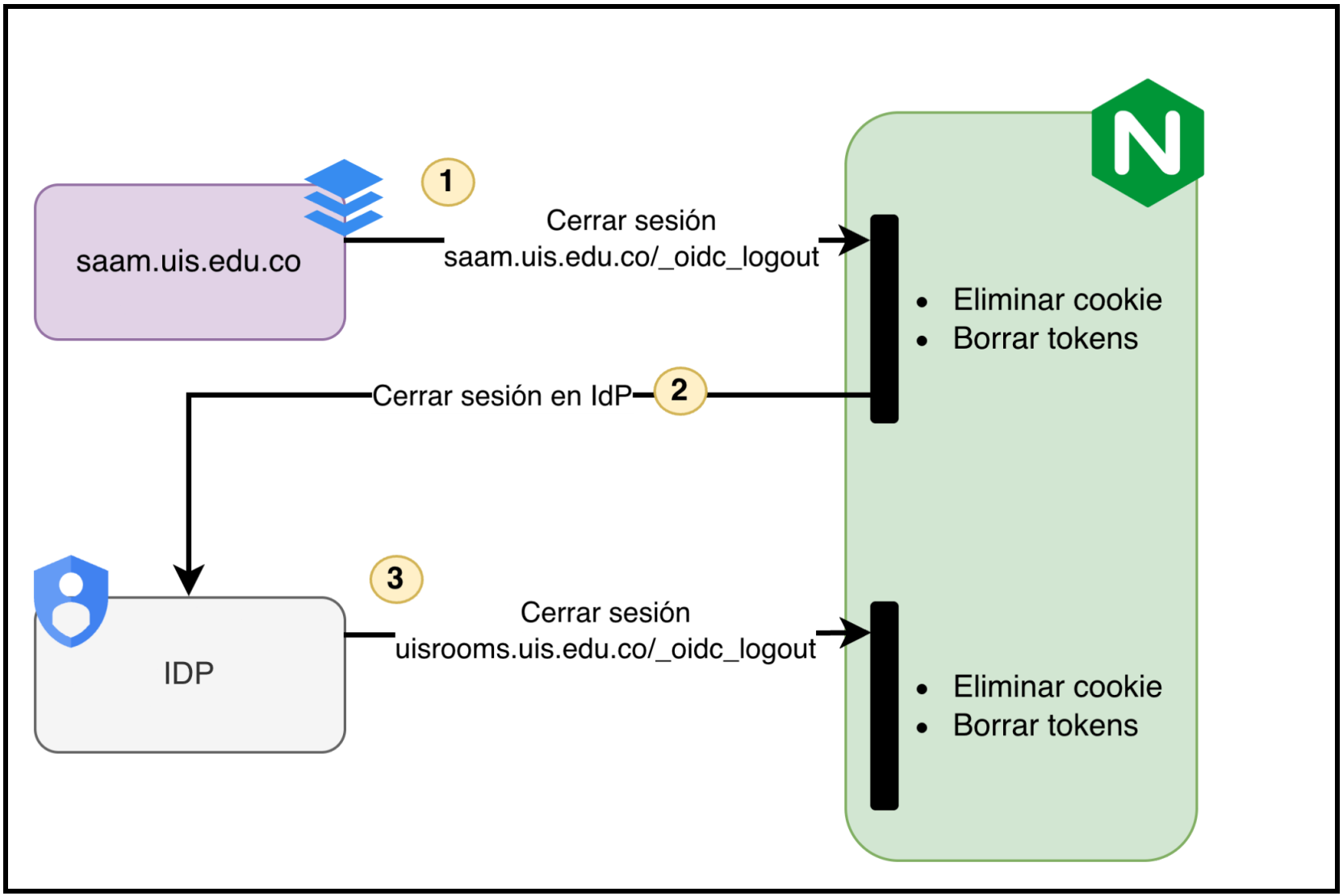
El resultado es que el usuario experimenta un acceso completamente transparente a UisRooms: fue redirigido brevemente al IdP y regresó de inmediato, sin ver ningún formulario de login. Cada aplicación tiene su propia cookie de sesión independiente gestionada por el proxy, pero todas comparten la misma sesión global de Keycloak como fuente de verdad. Esto significa también que cuando el usuario cierre sesión en cualquiera de las aplicaciones, el proceso de Single Logout destruirá tanto la cookie local de esa aplicación como la sesión global de Keycloak, invalidando el acceso a todas las demás aplicaciones del ecosistema de forma simultánea.

5.3.1.2 Single Logout (SLO). A continuación el flujo SLO en el IDP:

Figura 7

Flujo de SLO con el IdP

SISTEMA PROVEEDOR DE IDENTIDAD



El cierre de sesión, al igual que la autenticación, es un proceso centralizado y transparente que no requiere ninguna implementación por parte de los equipos de desarrollo. El endpoint /_oidc_logout está disponible automáticamente en todos los dominios gestionados por el proxy, ya que es una ruta transversal configurada directamente en Nginx a través de la librería resty.openidc. Cualquier aplicación que necesite cerrar la sesión del usuario simplemente redirige al usuario a ese endpoint bajo su propio dominio, y el proxy se encarga de todo el proceso.

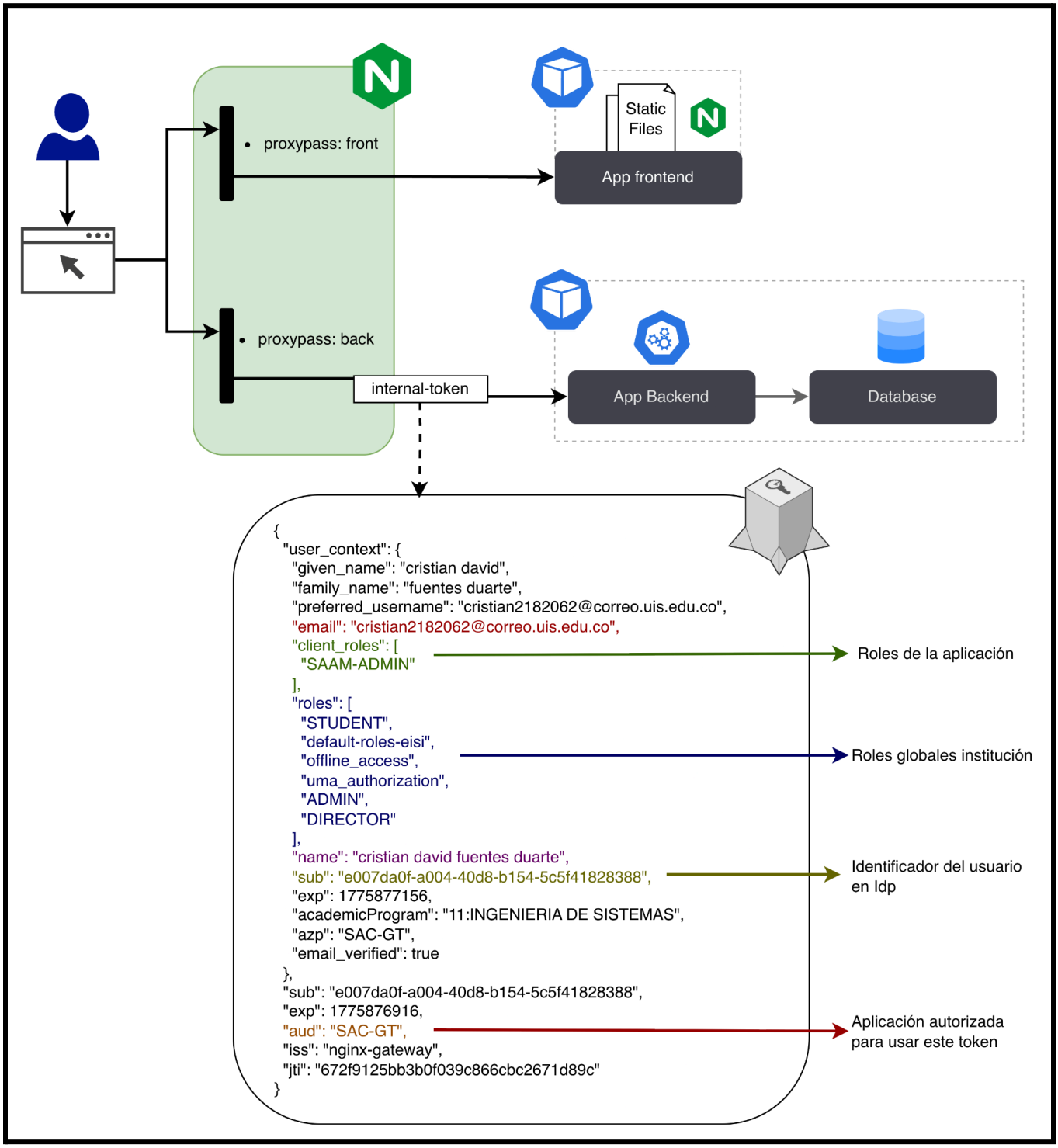
Cuando por ejemplo saam.uis.edu.co/_oidc_logout recibe la solicitud, el proxy elimina la cookie de sesión del dominio, borra los tokens almacenados en Redis y envía la notificación de cierre de sesión al IdP. Es en este punto donde Keycloak toma el control del proceso: al recibir la solicitud de logout, identifica todas las sesiones activas asociadas a ese usuario en los demás clientes registrados, en este caso uisrooms.uis.edu.co, y realiza una llamada directa al endpoint /_oidc_logout de cada uno de ellos. Cada instancia del proxy que recibe esta llamada de Keycloak ejecuta el mismo proceso de limpieza local: elimina la cookie de sesión del dominio correspondiente y borra sus tokens de Redis.

El resultado es que un cierre de sesión iniciado desde cualquier aplicación del ecosistema desencadena una cadena coordinada por Keycloak que invalida simultáneamente todas las sesiones activas del usuario en todas las plataformas, sin que ningún equipo de desarrollo haya tenido que implementar una sola línea de código para lograrlo.

Flujo de Autorización en las Aplicaciones:

Figura 8

Flujo e información redirigida por el proxy



Una vez que el proxy ha autenticado al usuario y verificado su sesión, cada petición que llega al backend viene acompañada de un Internal Token, un JWT firmado y generado por el

propio Nginx que contiene toda la información necesaria para que la aplicación tome decisiones de autorización sin necesidad de consultar al IdP en cada petición. El proxy diferencia automáticamente entre peticiones al frontend y peticiones al backend. Las peticiones al frontend son servidas directamente como recursos estáticos sin procesamiento adicional, ya que la autenticación ya fue validada a nivel del proxy. Las peticiones que el navegador realiza hacia el backend, identificadas por el prefijo /api, son interceptadas por Nginx, que inyecta el Internal Token en la cabecera X-Internal-Token antes de hacer el proxy_pass hacia el servicio correspondiente.

Estructura del Internal Token: El token que recibe el backend contiene dos niveles de información claramente diferenciados dentro del user_context. El primero es la identidad del usuario: su nombre, apellido, correo institucional, identificador único (sub) y programa académico. El sub es el identificador universal del usuario en todo el ecosistema, inmutable e irrepetible, que las aplicaciones deben usar como referencia principal para asociar registros internos a un usuario específico.

El segundo nivel son los roles, separados en dos campos con propósitos distintos. Los client_roles contienen únicamente los roles que el usuario tiene asignados dentro de esa aplicación específica, por ejemplo SAAM y “Administrador de SAAM” (rol específico). Estos roles son los que el equipo de desarrollo define en Keycloak para su propio cliente y los que deben usar para controlar el acceso a funcionalidades internas de su aplicación. Los roles en cambio son los roles globales o institucionales del usuario en todo el ecosistema, como Estudiante, Administrador o Director, que aplican transversalmente a todas las aplicaciones y representan la identidad institucional del usuario dentro de la Escuela. Adicionalmente, el token contiene campos de control como ‘aud’, que identifica a qué aplicación está destinado el token,

iss que indica que fue emitido por el gateway, exp que define su tiempo de expiración corto de 60 segundos, y jti que es un identificador único por petición que permite detectar intentos de reutilización del token.

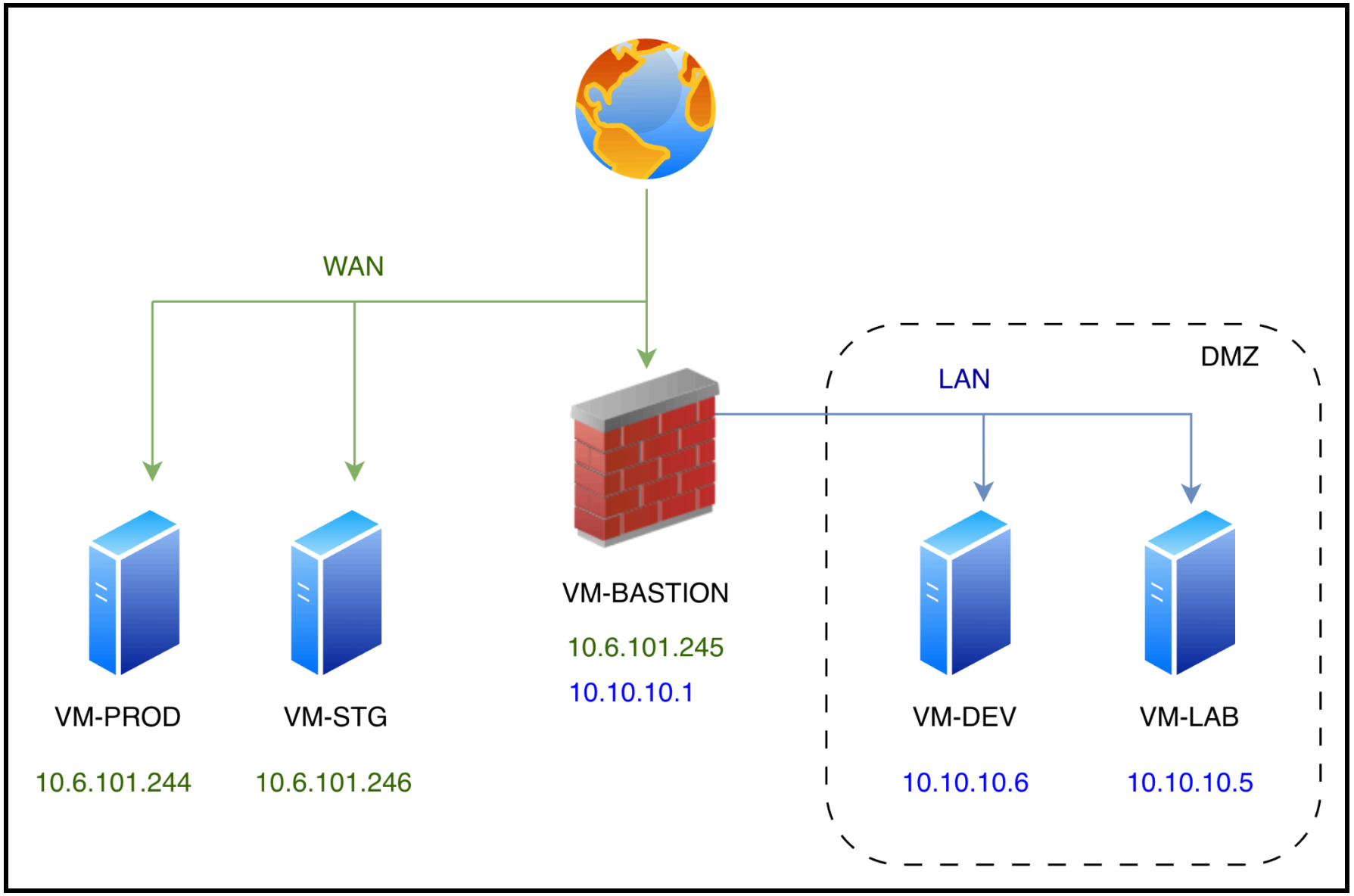
Validación en el backend: Cuando el backend recibe una petición, debe ejecutar tres validaciones sobre el Internal Token antes de procesar cualquier lógica de negocio. Primero verifica la firma del token usando el secreto hexadecimal (GW_INTERNAL_SECRET) que fue provisto por el laboratorio al equipo a través de Vault, garantizando que el token fue efectivamente generado por el proxy y no ha sido manipulado. Segundo verifica el audience (aud), confirmando que el token está destinado a su propia aplicación y no fue generado para otra. Tercero verifica la expiración (exp), rechazando tokens que hayan superado su ventana de validez de 60 segundos, lo que limita drásticamente la ventana de exposición en caso de que un token sea interceptado.

Una vez validado el token, la aplicación dispone de toda la información necesaria para tomar decisiones de autorización granulares: puede verificar si el usuario tiene un client_role específico para permitirle acceder a una funcionalidad administrativa, puede usar los roles globales para adaptar la experiencia según el tipo de usuario institucional, y puede usar el sub como identificador confiable para consultar o registrar información en su propia base de datos.

5.3.2 *Implementación de arquitectura*

Figura 9

Máquinas virtuales: IPs y redes


WAN
DMZ
LAN
VM-BASTION
10.6.101.245
10.10.10.1
VM-PROD
10.6.101.244
VM-STG
10.6.101.246
VM-DEV
10.10.10.6
VM-LAB
10.10.10.5

Una vez que la Escuela entrega las máquinas virtuales creadas en Proxmox y conectadas a sus respectivas redes, el primer paso es configurar VM-Bastion como el componente central de comunicación entre ambas redes. Como se describió en la arquitectura, Bastion es la única máquina con presencia en las dos redes simultáneamente, por lo que debe actuar como gateway predeterminado para todas las máquinas de la red LAN, es decir, como la puerta por la que todo el tráfico privado debe salir cuando necesita comunicarse con el exterior.

Las máquinas del laboratorio quedan distribuidas en dos redes con rangos de direcciones IP estáticas bien definidos. Las máquinas conectadas a la WAN (VM-Bastion, VM-Prod y VM-STG) tienen asignadas direcciones IP estáticas dentro del rango 10.66.101.0/24, que corresponde a la red interna de la universidad. Las máquinas conectadas a la LAN (VM-Dev y VM-Lab) tienen asignadas direcciones IP estáticas dentro del rango 10.10.10.0/24, que es la red privada del laboratorio administrada por Proxmox.

Un aspecto importante a entender es que el gateway predeterminado no es el mismo para todas las máquinas, y la razón es lógica: depende de por dónde necesita salir el tráfico de cada una.

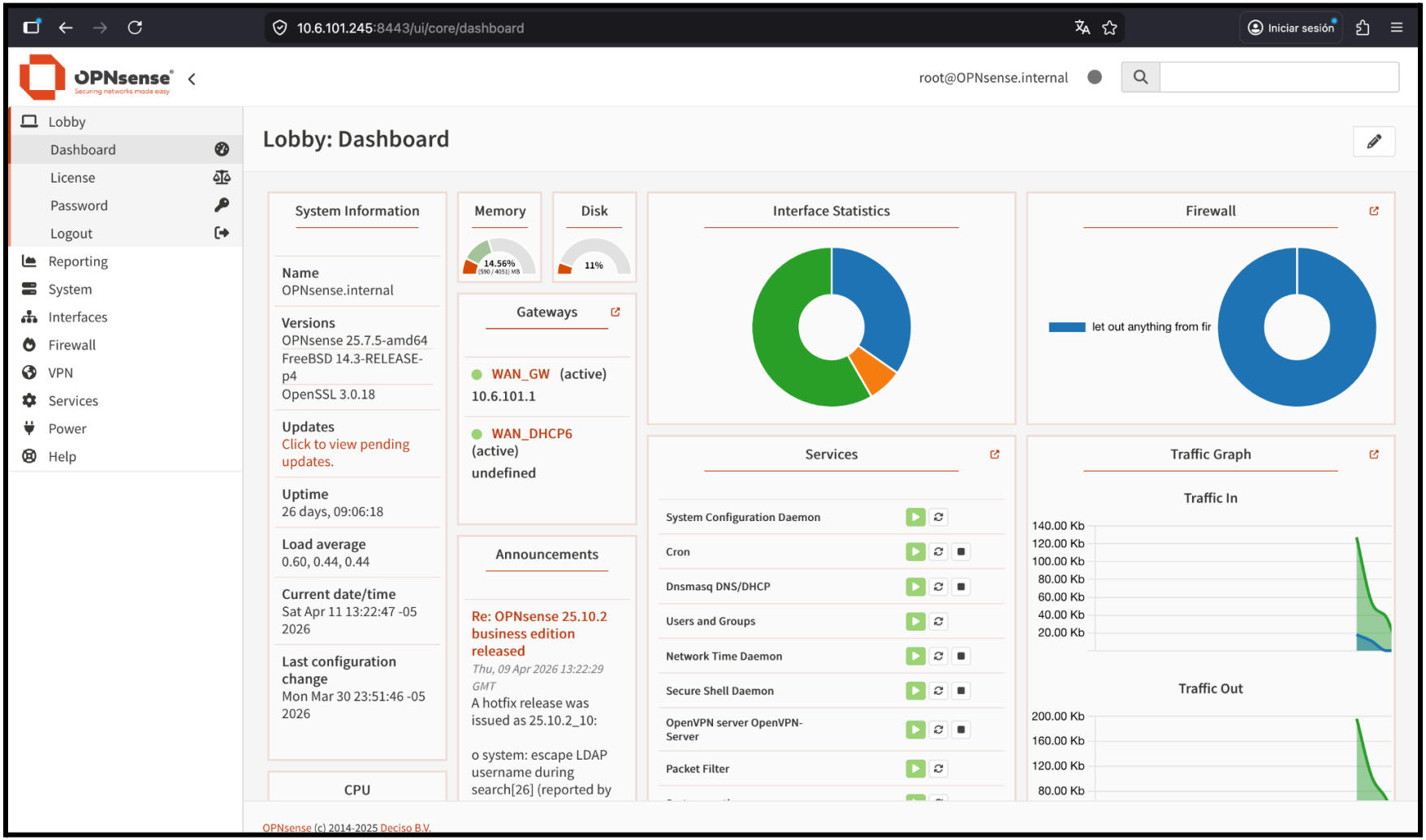
Las máquinas de la LAN, VM-Dev y VM-Lab, tienen una única puerta de enlace: 10.10.10.1, qué es la IP de VM-Bastion en esa red. Todo el tráfico que generen, ya sea hacia internet o hacia recursos de la WAN, debe pasar obligatoriamente por Bastion, que es el único punto de salida de la red privada.

Las máquinas de la WAN, VM-Prod y VM-STG, requieren dos puertas de enlace con propósitos distintos. Cuando necesitan salir a internet, utilizan el gateway 10.66.101.1, que es el

router de la universidad asignado a esa red. Cuando necesitan acceder a recursos internos del laboratorio ubicados en la LAN, como descargar imágenes contenerizadas desde VM-Lab, utilizan el gateway 10.66.101.245, que es la IP de VM-Bastion en la red WAN. Esta configuración de rutas múltiples es la que permite que VM-STG pueda acceder al laboratorio sin VPN, gracias a la regla de excepción configurada en Bastion que permite el tráfico desde su IP específica hacia la LAN.

Figura 10

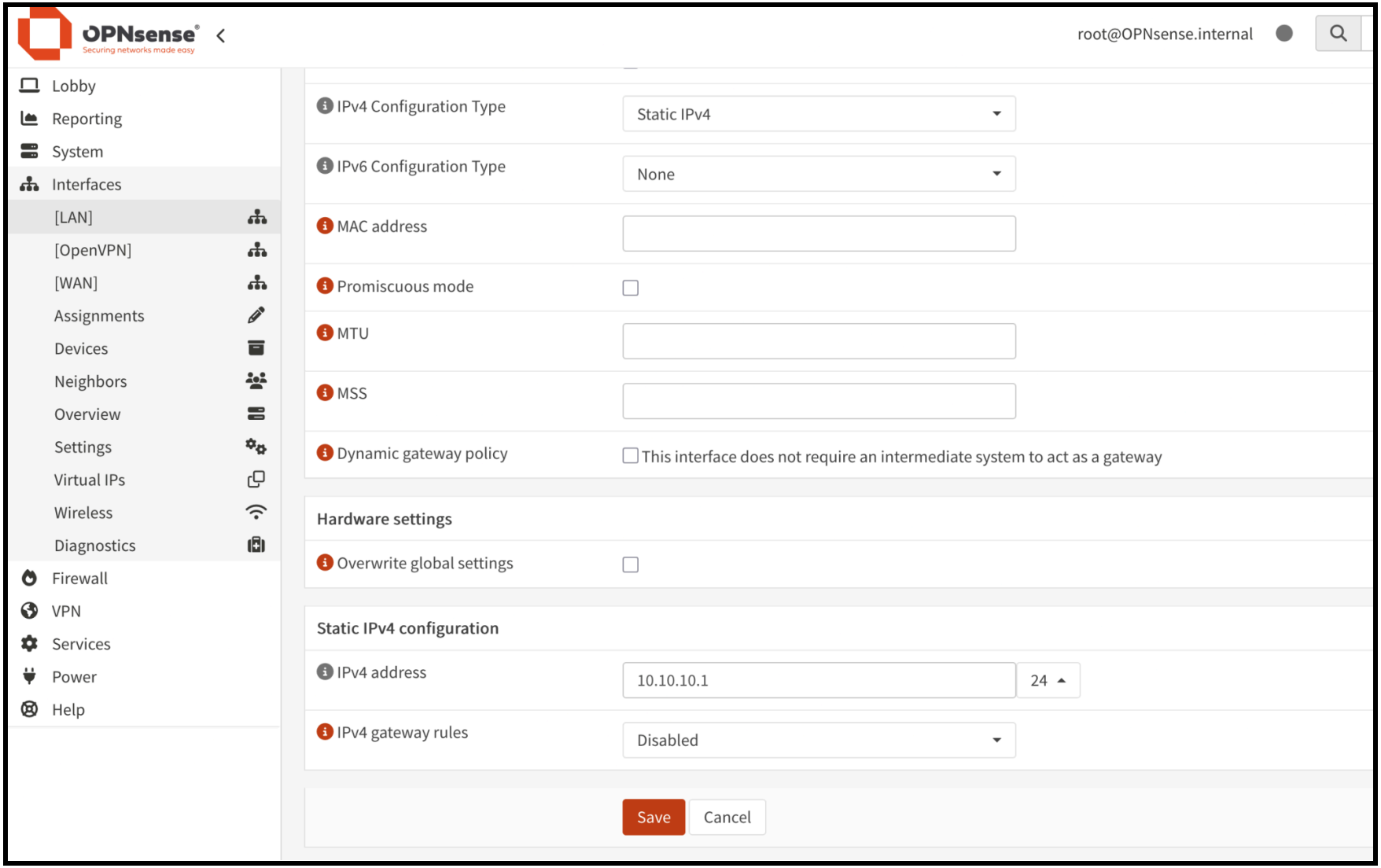
OPNsense Dashboard


10.6.101.245:8443/ui/core/dashboard
Iniciar sesión
OPNsense
Securing networks made easy
root@OPNsense.internal
Lobby
Dashboard
License
Password
Logout
Reporting
System
Interfaces
Firewall
VPN
Services
Power
Help
Lobby: Dashboard
System Information
Name
OPNsense.internal
Versions
OPNsense 25.7.5-amd64
FreeBSD 14.3-RELEASE-p4
OpenSSL 3.0.18
Updates
Click to view pending updates.
Uptime
26 days, 09:06:18
Load average
0.60, 0.44, 0.44
Current date/time
Sat Apr 11 13:22:47 -05 2026
Last configuration change
Mon Mar 30 23:51:46 -05 2026
CPU
Memory
14.56%
Disk
11%
Gateways
WAN_GW (active)
10.6.101.1
WAN_DHCP6 (active)
undefined
Announcements
Re: OPNsense 25.10.2 business edition released
Thu, 09 Apr 2026 13:22:29 GMT
A hotfix release was issued as 25.10.2_10:
o system: escape LDAP username during search[26] (reported by
Interface Statistics
Firewall
let out anything from fir
Services
System Configuration Daemon
Cron
Dnsmasq DNS/DHCP
Users and Groups
Network Time Daemon
Secure Shell Daemon
OpenVPN server OpenVPN-Server
Packet Filter
Traffic Graph
Traffic In
140.00 Kb
120.00 Kb
100.00 Kb
80.00 Kb
60.00 Kb
40.00 Kb
20.00 Kb
Traffic Out
200.00 Kb
160.00 Kb
120.00 Kb
80.00 Kb
OPNsense (c) 2014-2025 Deciso B.V.

OPNsense cuenta con una interfaz de administración web desde la cual se realizan todas las configuraciones del firewall. El primer paso es verificar y configurar las interfaces de red, que son los puntos de conexión entre VM-Bastion y cada una de las redes del laboratorio. Sin esta configuración correctamente establecida, ninguna de las reglas de comunicación entre redes podrá funcionar.

Figura 11

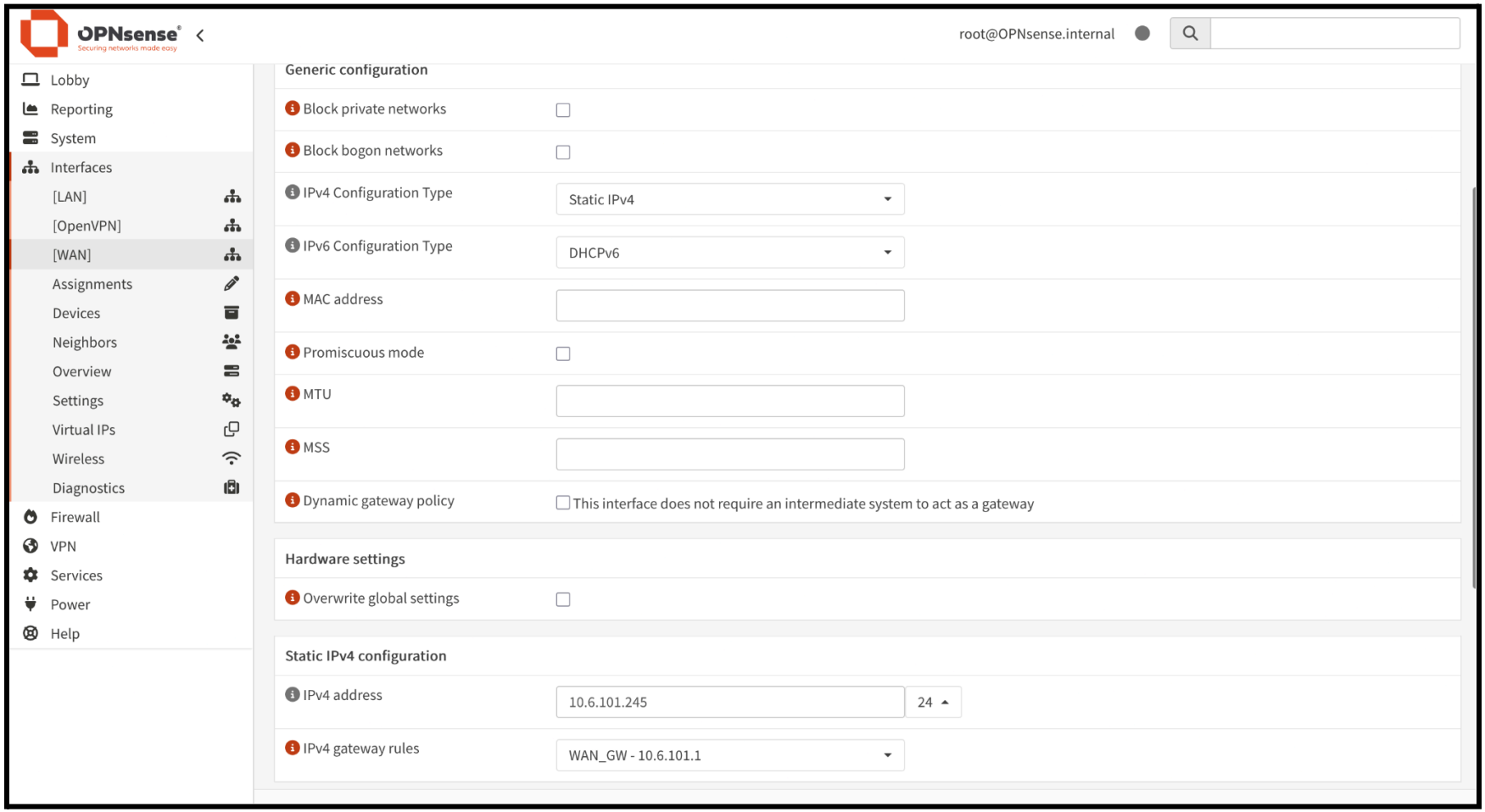
OPNSense Interfaces: LAN


OPNsense
Securing networks made easy
root@OPNsense.internal
Lobby
Reporting
System
Interfaces
[LAN]
[OpenVPN]
[WAN]
Assignments
Devices
Neighbors
Overview
Settings
Virtual IPs
Wireless
Diagnostics
Firewall
VPN
Services
Power
Help
IPv4 Configuration Type
Static IPv4
IPv6 Configuration Type
None
MAC address
Promiscuous mode
MTU
MSS
Dynamic gateway policy
This interface does not require an intermediate system to act as a gateway
Hardware settings
Overwrite global settings
Static IPv4 configuration
IPv4 address
10.10.10.1
24
IPv4 gateway rules
Disabled
Save
Cancel

La interfaz LAN aparece habilitada y configurada con la dirección IP asignada por la Escuela en Proxmox para esta red. Es importante notar que el gateway de esta interfaz está deshabilitado intencionadamente: la red LAN es una red privada completamente aislada que no tiene salida directa a internet. Cuando las máquinas de esta red necesiten comunicarse con el exterior, el tráfico deberá salir a través de la interfaz WAN de Bastion, no por la LAN.

Figura 12

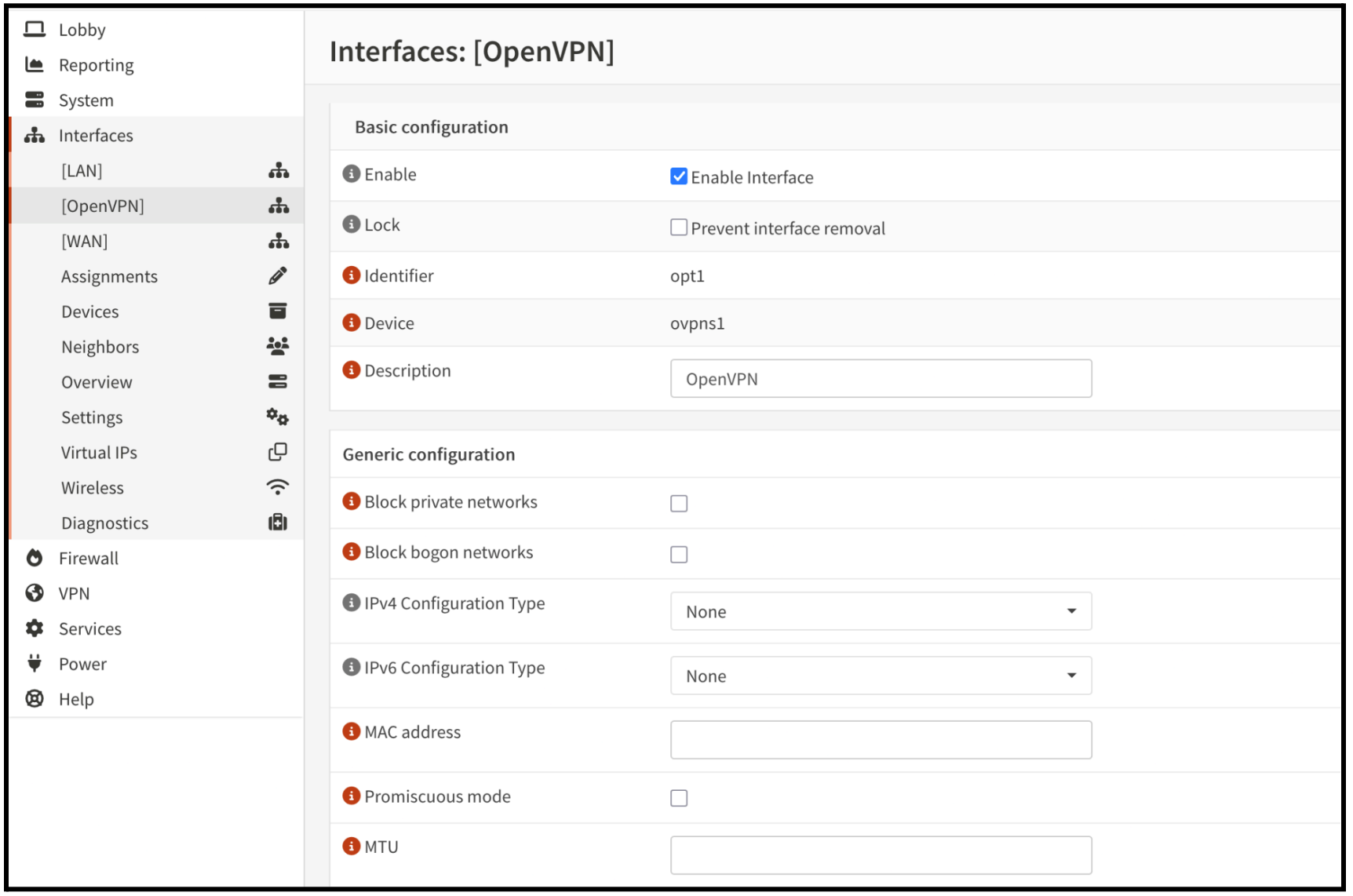
OPNSense Interfaces: WAN


OPNsense
Securing networks made easy
root@OPNsense.internal
Lobby
Reporting
System
Interfaces
[LAN]
[OpenVPN]
[WAN]
Assignments
Devices
Neighbors
Overview
Settings
Virtual IPs
Wireless
Diagnostics
Firewall
VPN
Services
Power
Help
Generic configuration
Block private networks
Block bogon networks
IPv4 Configuration Type
Static IPv4
IPv6 Configuration Type
DHCPv6
MAC address
Promiscuous mode
MTU
MSS
Dynamic gateway policy
This interface does not require an intermediate system to act as a gateway
Hardware settings
Overwrite global settings
Static IPv4 configuration
IPv4 address
10.6.101.245
24
IPv4 gateway rules
WAN_GW - 10.6.101.1

La interfaz WAN está configurada con la dirección IP privada asignada a VM-Bastion dentro de la red de la universidad, junto con el gateway de salida a internet correspondiente al router universitario. Esta es la única puerta de salida real hacia internet para todo el laboratorio.

Figura 13

OPNSense Interfaces: OpenVPN


Lobby
Reporting
System
Interfaces
[LAN]
[OpenVPN]
[WAN]
Assignments
Devices
Neighbors
Overview
Settings
Virtual IPs
Wireless
Diagnostics
Firewall
VPN
Services
Power
Help
Interfaces: [OpenVPN]
Basic configuration
Enable
Enable Interface
Lock
Prevent interface removal
Identifier
opt1
Device
ovpns1
Description
OpenVPN
Generic configuration
Block private networks
Block bogon networks
IPv4 Configuration Type
None
IPv6 Configuration Type
None
MAC address
Promiscuous mode
MTU

Adicionalmente se habilita la interfaz OpenVPN, que es una interfaz virtual interna gestionada por OPNsense. Esta interfaz no corresponde a una conexión física de red sino que es creada automáticamente por el servidor OpenVPN al activarse, y es a través de ella que se gestionará el tráfico cifrado de los clientes que se conecten al laboratorio mediante VPN.

Antes de continuar con la configuración del firewall, es necesario establecer las rutas de red en las máquinas virtuales VM-Dev y VM-Lab. Sin esta configuración, estas máquinas no sabrán por qué camino enviar sus respuestas cuando reciban tráfico, lo que haría imposible cualquier comunicación. La conexión a estas máquinas se realiza mediante SSH, usando el par de llaves previamente configurado por la Escuela. En Ubuntu, las rutas de red se definen editando el archivo /etc/netplan/50-cloud-init.yaml, donde se establece que el gateway predeterminado para cualquier tráfico saliente es 10.10.10.1, correspondiente a la IP de VM-Bastion en la red LAN.

Figura 14

Configuración VMs

```
root@VM-DEV:/home/EISI# cat /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    eth0:
      match:
        macaddress: "aa:95:a7:53:60:14"
      addresses:
      - "10.10.10.6/24"
      nameservers:
        addresses:
        - 192.168.19.110
        search:
        - uis.edu.co
      set-name: "eth0"
      routes:
      - to: "default"
        via: "10.10.10.1"
root@VM-DEV:/home/EISI#
```

Para VM-Prod y VM-STG la configuración es similar pero con una diferencia importante: estas máquinas necesitan dos rutas. La ruta predeterminada apunta al gateway de la universidad 10.66.101.1, que es por donde sale el tráfico hacia internet al estar en la WAN. Sin embargo, cuando el destino sea algún recurso dentro de la red privada del laboratorio, es decir cualquier dirección dentro del rango 10.10.10.0/24, las máquinas deben usar como puerta de enlace la IP 10.66.101.245, que es la dirección de VM-Bastion en la red WAN. Esta ruta específica es la que permite que VM-STG pueda descargar imágenes contenerizadas desde VM-Lab sin necesidad de VPN.

Figura 15

Configuración VMs: rutas

```
root@VM-PROD-244:/home/EISI# cat /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    eth0:
      match:
        macaddress: "7a:47:5c:32:0e:84"
      addresses:
      - "10.6.101.244/24"
      nameservers:
        addresses:
        - 1.1.1.1
        search:
        - uis.edu.co
      set-name: "eth0"
      routes:
      - to: "default"
        via: "10.6.101.1"
      - to: "10.10.10.0/24"
        via: "10.6.101.245"
root@VM-PROD-244:/home/EISI#
```

Con estas rutas configuradas en cada máquina, todo el ecosistema sabe exactamente qué camino debe seguir cada paquete: las máquinas de la LAN salen siempre por Bastion, y las máquinas de la WAN salen por el router universitario para internet y por Bastion para los recursos internos del laboratorio.

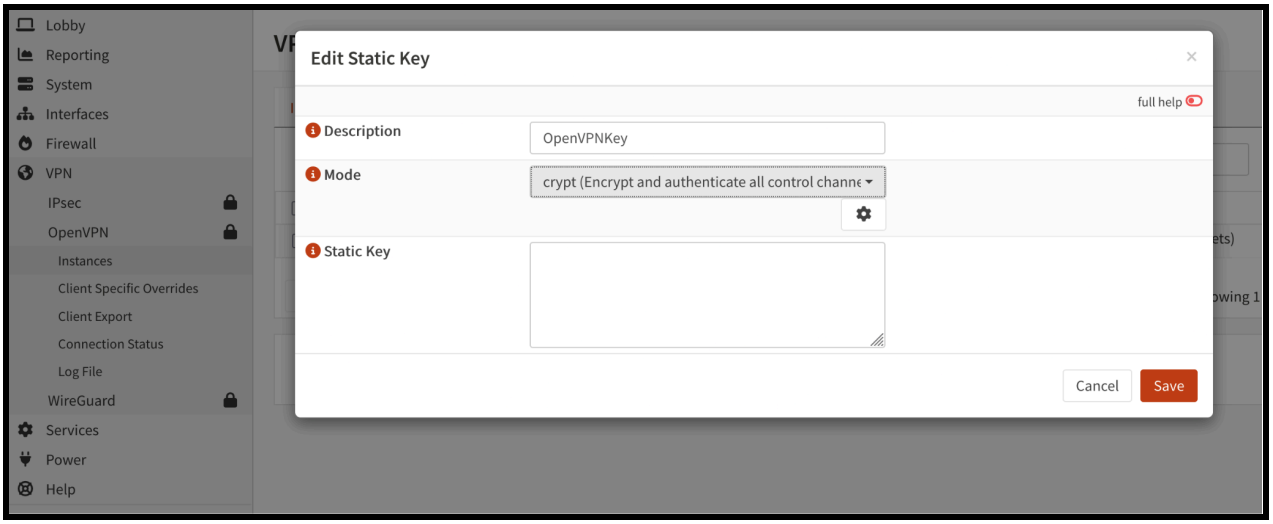
Para poder trabajar en el laboratorio, ya sea desde dentro o fuera de las instalaciones de la universidad, es necesario configurar el servidor VPN. Como se explicó anteriormente, todos los recursos críticos del laboratorio están en la red LAN, completamente aislada del exterior, por lo que la VPN es el único mecanismo de acceso autorizado independientemente de la ubicación física del desarrollador.

La configuración de OpenVPN en OPNsense se realiza en varias etapas: primero se generan las llaves y certificados necesarios para establecer comunicación segura, y luego se configura el servidor propiamente dicho.

Para la generación de la llave estática se dirige a VPN → OpenVPN → Instances → Static Keys para crear la llave criptográfica que cifrará el canal de comunicación del túnel VPN.

Figura 16

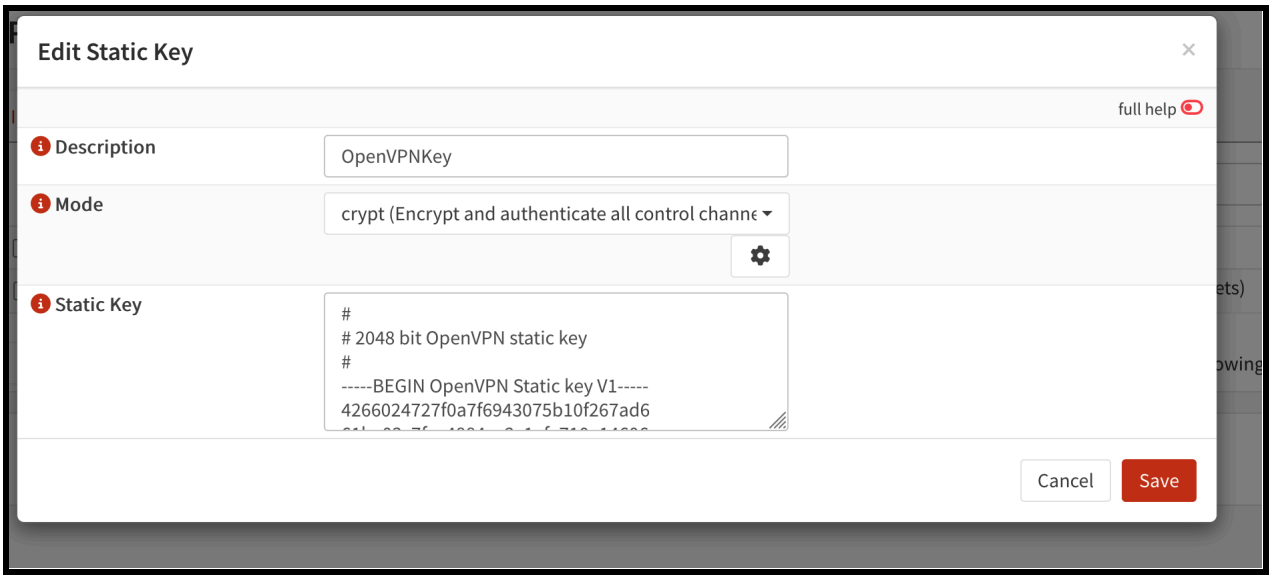
Static Keys



Asignamos un nombre descriptivo, en este caso OpenVpnKey, y seleccionamos el modo crypt, que indica que esta llave se usará específicamente para cifrar el canal de comunicación del túnel, garantizando que todo el tráfico que viaje a través de la VPN sea ilegible para cualquier tercero que intente interceptar. Al guardar y volver a abrir el registro, el campo Static Key habrá sido autogenerado por OPNsense con un valor criptográfico aleatorio.

Figura 17

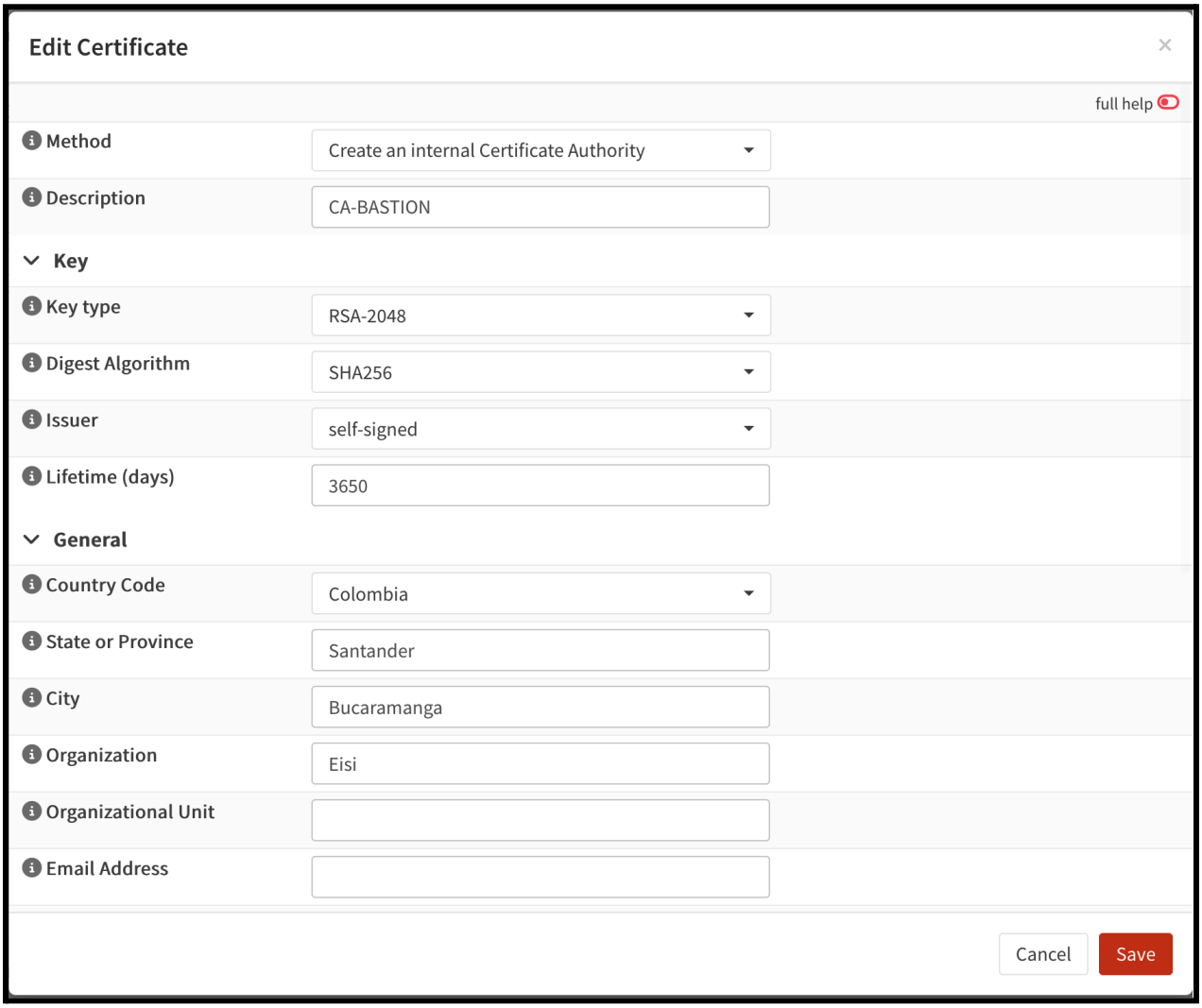
Static Keys: Creación



Antes de configurar el servidor OpenVPN necesitamos un Certificado de Autoridad, conocido como CA. Una CA es una entidad de confianza que firma y evalúa la validez de los certificados que se emitan bajo ella, de forma similar a como una notaría valida la autenticidad de un documento. OpenVPN usará esta CA para verificar que tanto el servidor como los clientes que intenten conectarse son legítimos. Nos dirigimos a System → Trust → Authorities.

Figura 18

Creación de certificados de autoridad

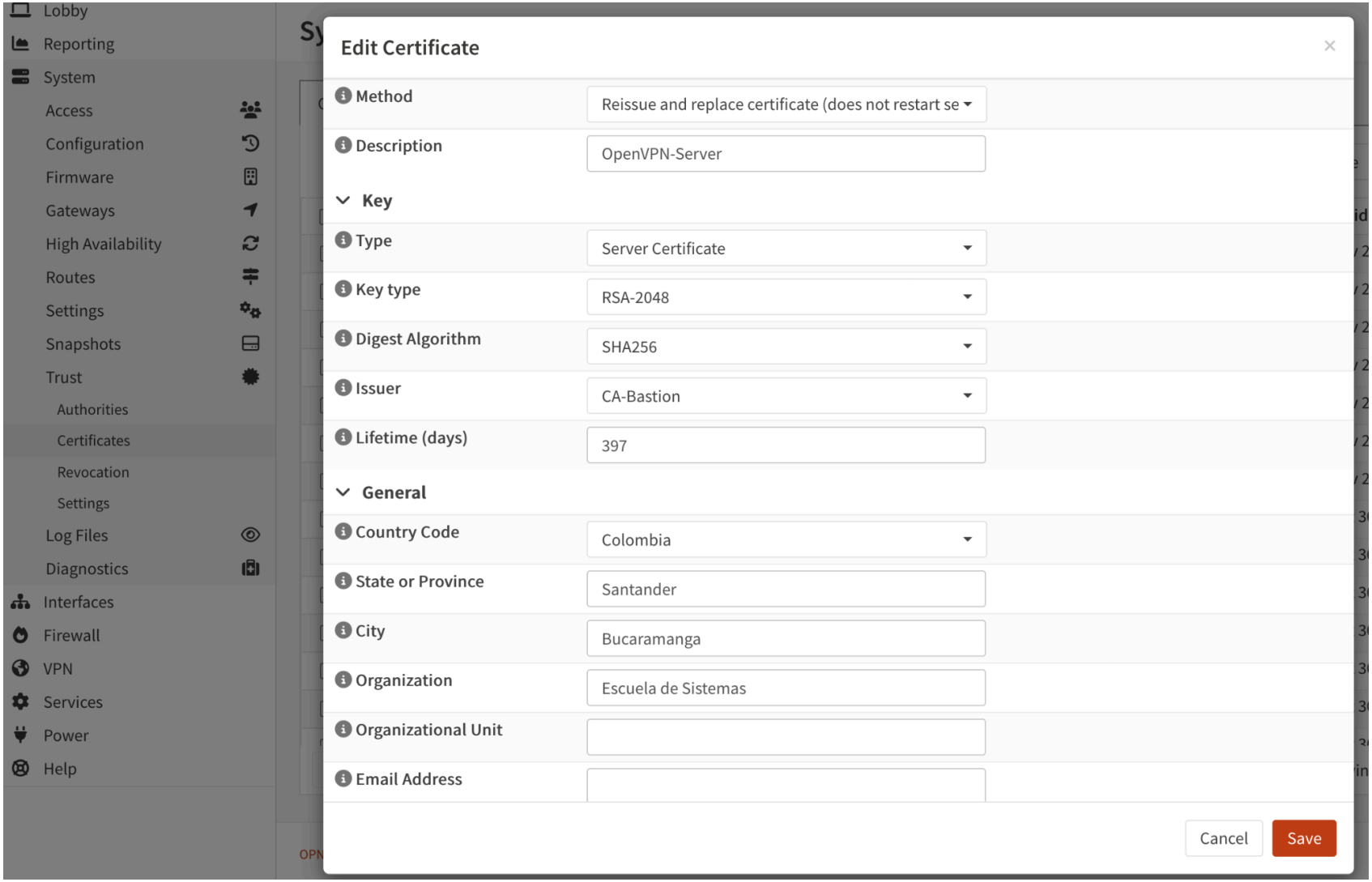


Creamos el certificado de autoridad con la descripción CA-BASTION y dejamos las configuraciones estándar: algoritmo RSA-2048, función de hash SHA256 y tipo self-signed, lo que significa que este certificado se firma a sí mismo sin depender de una CA externa. Configuramos una duración de 3650 días, equivalente a 10 años de validez. Es importante completar correctamente los campos de ubicación como país, provincia, ciudad y organización, y especialmente el campo commonName, que debe tener un valor como cabastion, ya que su ausencia puede causar errores en la generación del certificado. Al guardar y reabrir el registro, OPNsense habrá generado automáticamente el par de llaves pública y privada de la CA, que serán utilizadas en los siguientes pasos.

Con la CA creada, el siguiente paso es generar el certificado específico para el servidor OpenVPN. Este certificado es el que el servidor presenta a los clientes cuando intentan conectarse, para demostrar que están hablando con el servidor legítimo y no con un impostor. Nos dirigimos a System → Trust → Certificates y agregamos un nuevo certificado.

Figura 19

Creación de certificados de servidor

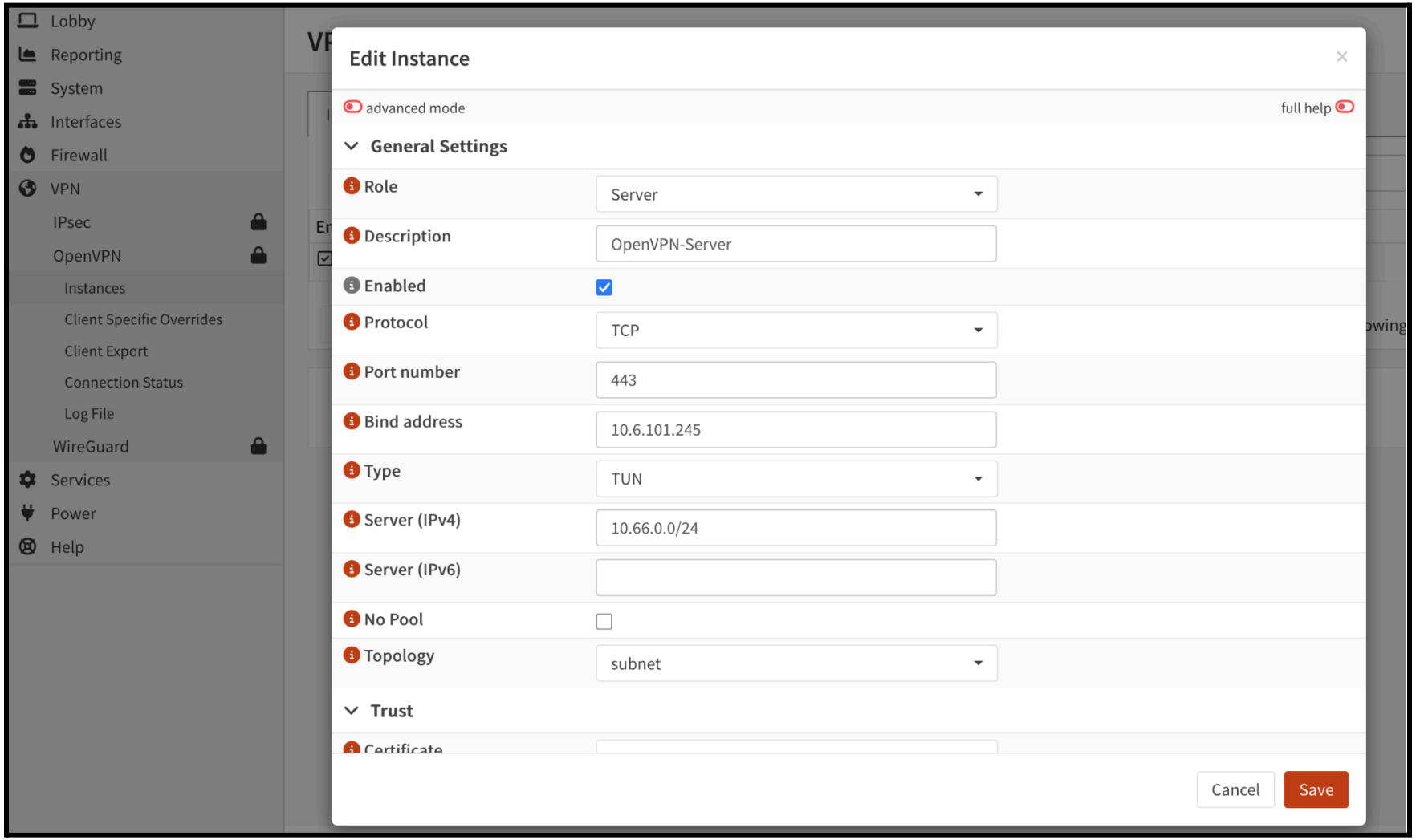

Lobby
Reporting
System
Access
Configuration
Firmware
Gateways
High Availability
Routes
Settings
Snapshots
Trust
Authorities
Certificates
Revocation
Settings
Log Files
Diagnostics
Interfaces
Firewall
VPN
Services
Power
Help
Edit Certificate
Method
Reissue and replace certificate (does not restart se
Description
OpenVPN-Server
Key
Type
Server Certificate
Key type
RSA-2048
Digest Algorithm
SHA256
Issuer
CA-Bastion
Lifetime (days)
397
General
Country Code
Colombia
State or Province
Santander
City
Bucaramanga
Organization
Escuela de Sistemas
Organizational Unit
Email Address
Cancel
Save

Asignamos la descripción OpenVPN-Server y seleccionamos el tipo Server Certificate, indicando que este certificado está destinado al uso exclusivo del servidor OpenVPN. En el campo Issuer seleccionamos CA-BASTION, el certificado de autoridad que creamos en el paso anterior. Al seleccionarlo, los campos de ubicación se completan automáticamente con los datos heredados de la CA. Guardamos y el certificado de servidor queda listo para ser usado en la configuración del servidor OpenVPN.

Con los certificados listos, procedemos a crear la instancia del servidor OpenVPN. Nos dirigimos a VPN → OpenVPN → Instances y agregamos una nueva instancia.

Figura 20

Instancia Post-Certificados


Lobby
Reporting
System
Interfaces
Firewall
VPN
IPsec
OpenVPN
Instances
Client Specific Overrides
Client Export
Connection Status
Log File
WireGuard
Services
Power
Help
Edit Instance
advanced mode
full help
General Settings
Role
Server
Description
OpenVPN-Server
Enabled
Protocol
TCP
Port number
443
Bind address
10.6.101.245
Type
TUN
Server (IPv4)
10.66.0.0/24
Server (IPv6)
No Pool
Topology
subnet
Trust
Certificate
Cancel
Save

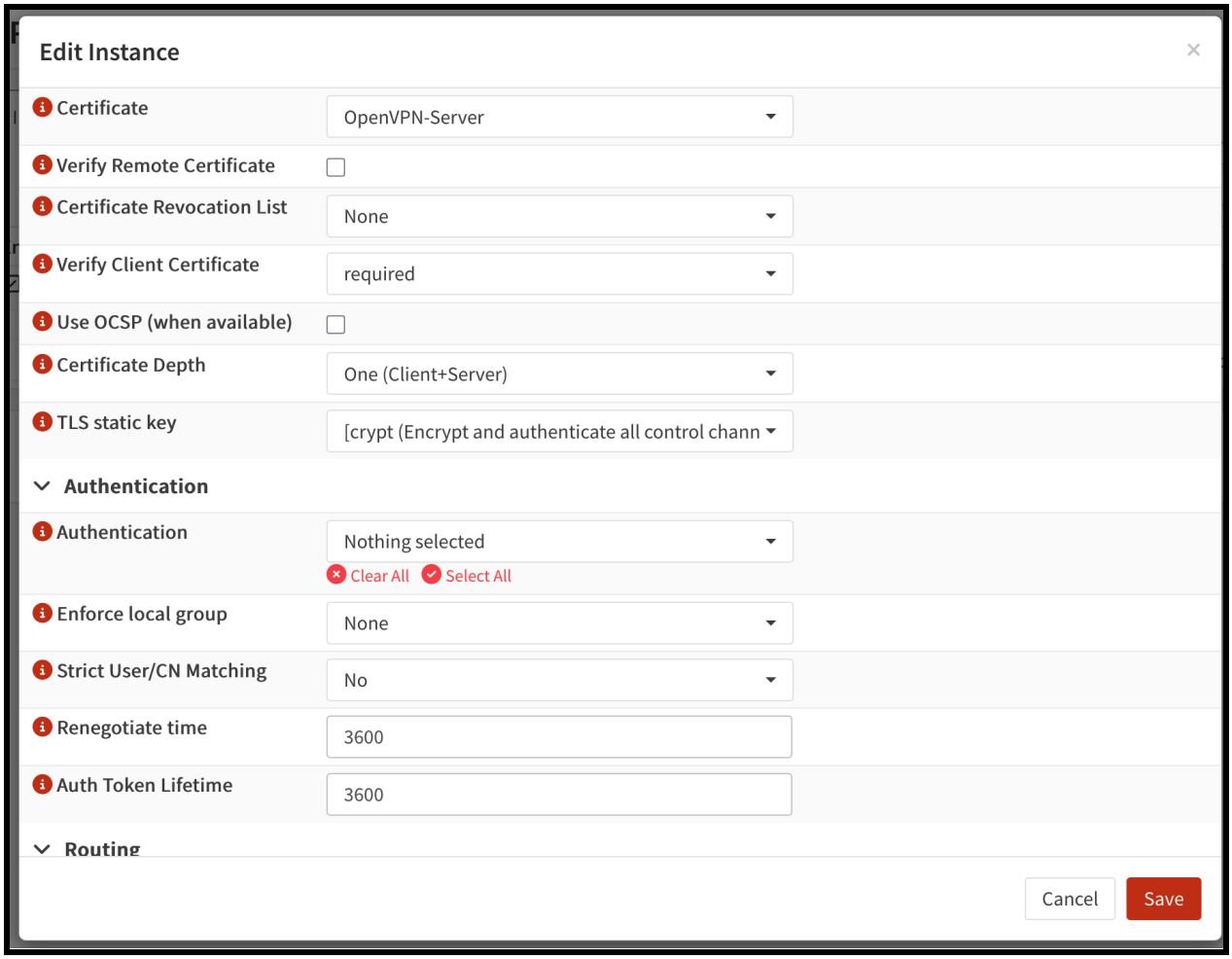
Configuramos el Role como Server, ya que VM-Bastion actuará como el servidor VPN al cual se conectarán todos los desarrolladores para acceder al laboratorio. Asignamos la descripción OpenVPN-Server y habilitamos la instancia.

Un aspecto importante de esta configuración es la elección del protocolo y puerto. El estándar de la industria para servidores VPN es usar el protocolo UDP en el puerto 1194, ya que ofrece mayor velocidad al no requerir confirmación de entrega de paquetes. Sin embargo, la red de la universidad restringe el tráfico saliente permitiendo únicamente el puerto 443, que normalmente se usa para HTTPS. Por esta razón configuramos el protocolo TCP y el puerto 443, lo que permite que el tráfico VPN pase desapercibido para el firewall de la universidad al utilizar el mismo puerto que el tráfico web seguro convencional.

En el campo Bind Address colocamos 10.66.101.245, que es la IP de VM-Bastion en la red WAN, indicando que el servidor OpenVPN escuchará conexiones entrantes en esa interfaz. Seleccionamos el tipo TUN, que crea un túnel de red virtual punto a punto entre el cliente y el servidor, a través del cual todo el tráfico viajará cifrado. En el campo Server (IPv4) definimos el rango 10.66.0.0/24, que es el pool de direcciones IP que se asignan dinámicamente a cada cliente que se conecte a la VPN. La topología la dejamos en subnet.

Figura 21

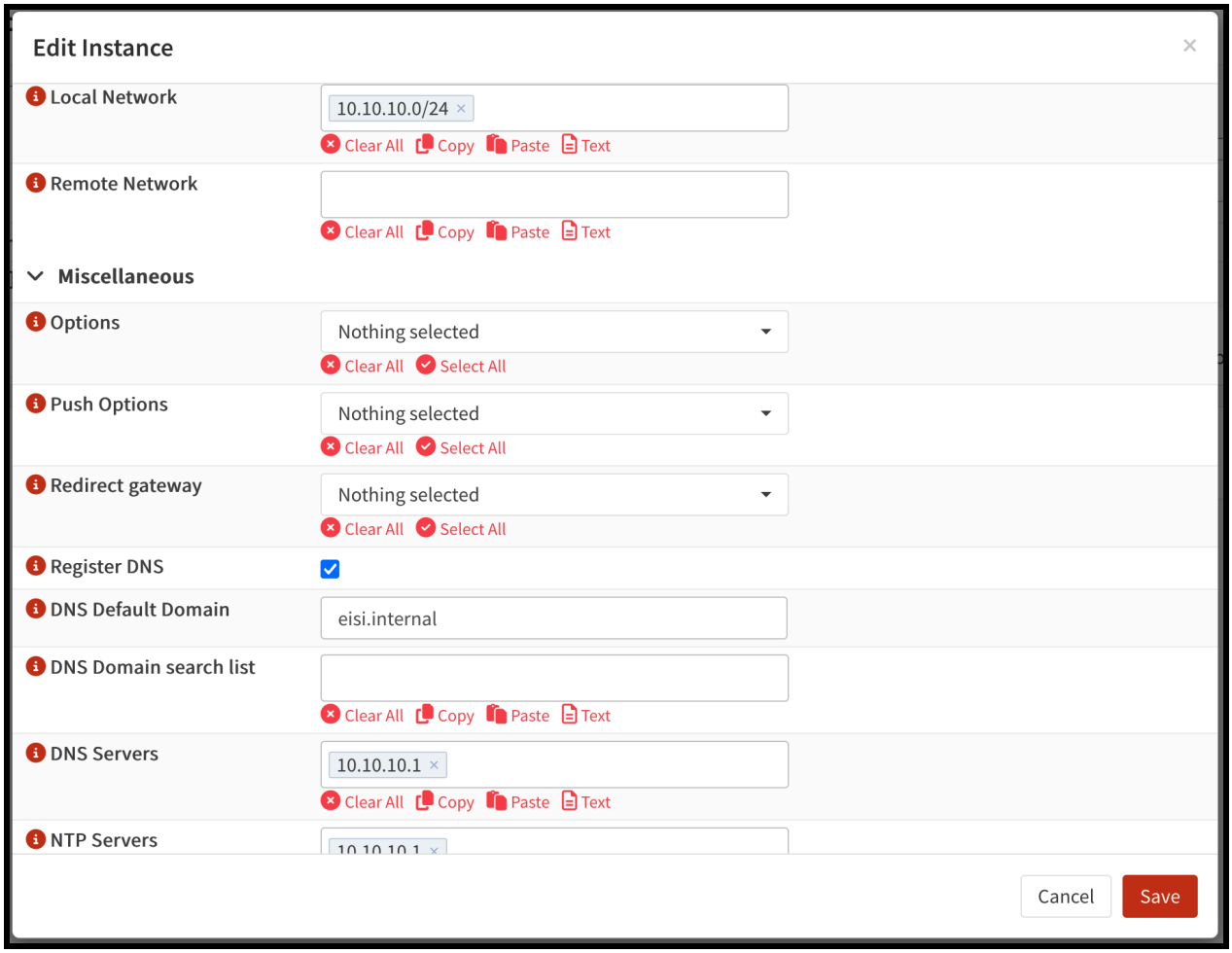
Configuración de instancia OpenVPN (1)



En el campo Certificate seleccionamos OpenVPN-Server, el certificado de servidor que creamos anteriormente, que será presentado a los clientes para validar la identidad del servidor. En el campo TLS Static Key seleccionamos la llave estática OpenVpnKey generada en el primer paso, que añade una capa adicional de cifrado al canal de comunicación antes incluso de que se complete el proceso de autenticación. Las demás opciones se dejan con sus valores predeterminados.

Figura 22

Configuración de instancia OpenVPN (2)



En el campo Local Network especificamos el rango 10.10.10.0/24, que corresponde a la red LAN del laboratorio. Esta configuración le indica al servidor OpenVPN que debe enrutar hacia esa red el tráfico de los clientes conectados, dándoles acceso efectivo a VM-Dev y VM-Lab como si estuvieran físicamente dentro de la red privada.

Habilitamos la opción Register DNS y en el campo DNS Default Domain colocamos eisi.internal. En DNS Servers colocamos 10.10.10.1, qué es la IP de VM-Bastion en la LAN. Esta configuración es la que hace posible el uso de dominios internos exclusivos del laboratorio: cuando un desarrollador se conecta por VPN, su dispositivo sabrá que cualquier dominio bajo .eisi.internal debe resolverse consultando al servidor DNS del firewall en 10.10.10.1. Esto

permite definir dominios como gitlab.eisi.internal o sonarqube.eisi.internal, que son completamente inaccesibles desde fuera de la VPN y que apuntan directamente a los servicios alojados en VM-Lab. Un usuario sin VPN activa que intente acceder a cualquiera de estos dominios no obtendrá ninguna respuesta, ya que esas direcciones no existen fuera del contexto de la red privada del laboratorio.

Con esta configuración guardada, el servidor OpenVPN está listo para aceptar conexiones.

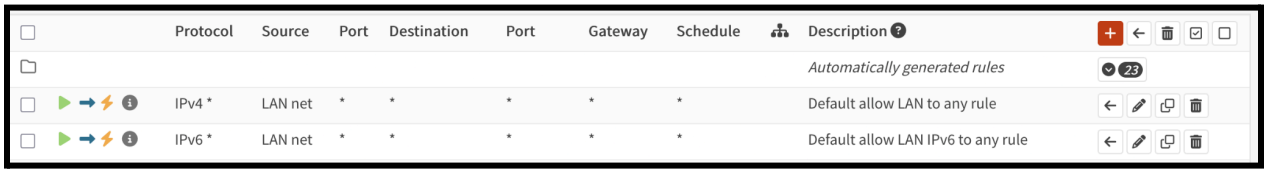
Con el servidor OpenVPN configurado, el siguiente paso es establecer las reglas del firewall que controlan qué tráfico está permitido entrar y salir de cada red. Estas reglas son el mecanismo central de seguridad del laboratorio y deben diseñarse con criterio: demasiado permisivas exponen recursos críticos, demasiado restrictivas impiden el trabajo de los equipos. Las reglas se definen considerando las siguientes restricciones y criterios de seguridad por cada interfaz de red:

Para la red LAN, todo el tráfico entrante está permitido por defecto, y se debe habilitar tráfico saliente SSH para permitir que las máquinas inicien conexiones hacia otras máquinas de forma autónoma. Para la red WAN, se permite tráfico entrante SSH para administración y mantenimiento, tráfico HTTPS entrante por el puerto 443, y todo el tráfico proveniente de la IP de VM-Prod. Para la interfaz OpenVPN, los clientes conectados tendrán acceso exclusivamente a la red LAN, sin posibilidad de saltar hacia recursos de la WAN, y solo podrán usar los protocolos HTTP y HTTPS en los puertos 80 y 443, más el puerto 2222 para clonar repositorios desde GitLab.

Entonces se sigue a Firewall → Rules → LAN.

Figura 23

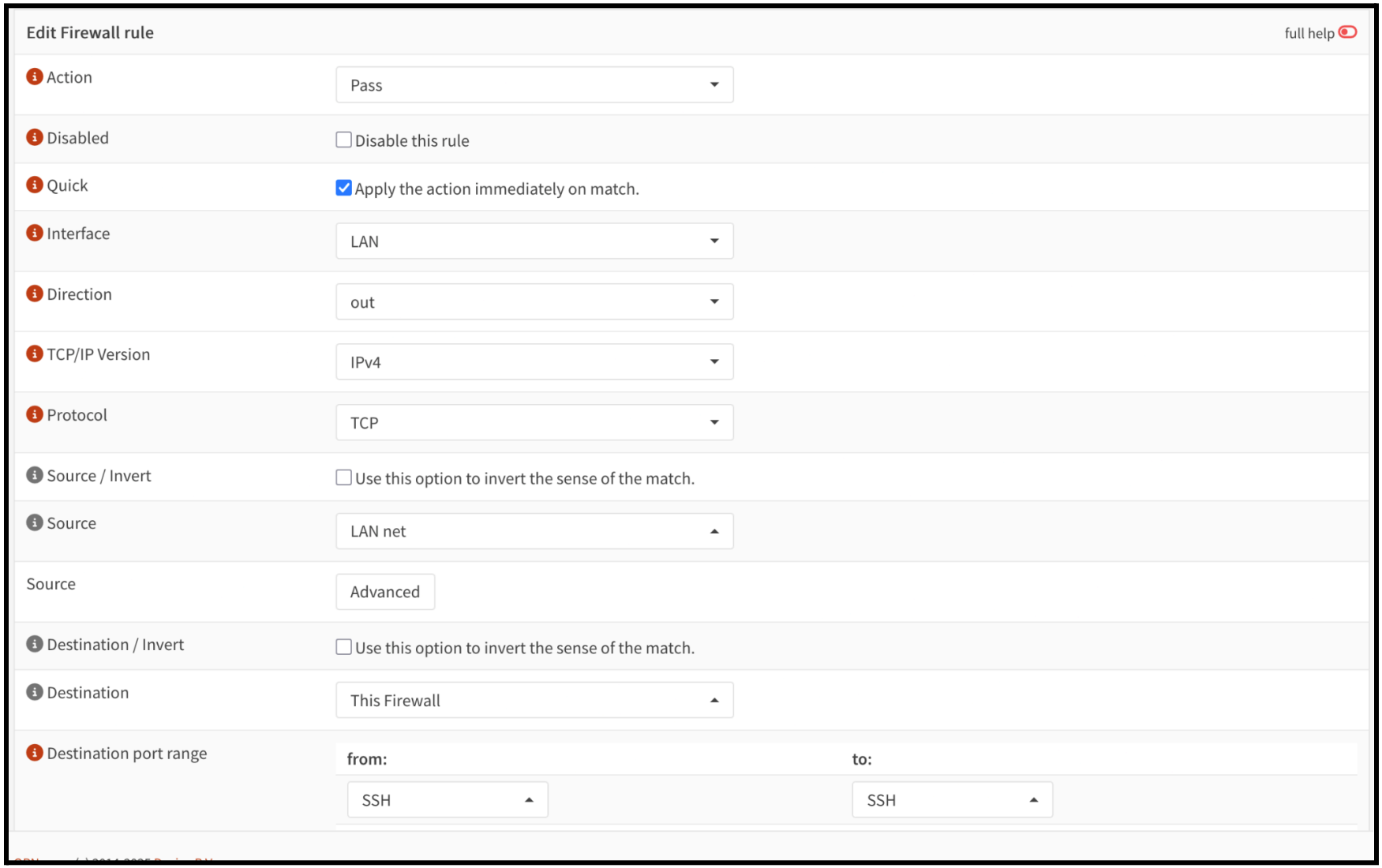
OPNSense: LAN rules & firewall



Por defecto OPNsense permite todo el tráfico entrante en la LAN. Un aspecto fundamental a entender aquí es que estas reglas son de tipo stateful, es decir, mantienen el estado de cada conexión. Esto significa que cuando se establece una conexión entrante, esa misma conexión bidireccional queda registrada y las respuestas pueden salir por ella sin necesidad de una regla de salida explícita. Por esta razón no es necesario crear reglas de salida para el tráfico de respuesta. Sin embargo, cuando una máquina de la LAN necesita iniciar una conexión de salida de forma autónoma, por ejemplo VM-Lab conectándose a VM-Prod, sí es necesario crear una regla explícita que lo permita. Agregamos una nueva regla para este propósito.

Figura 24

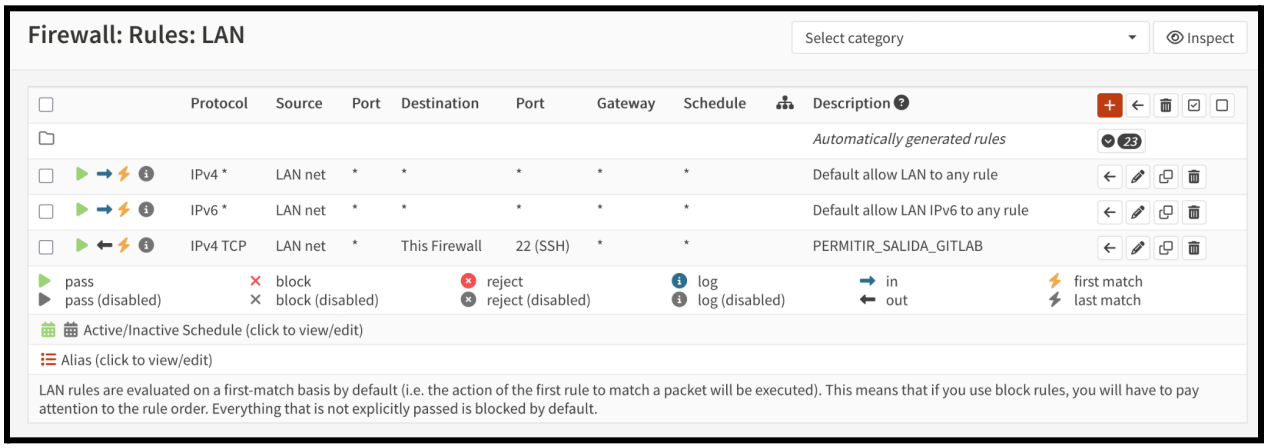
LAN: Creación de reglas


Edit Firewall rule
full help
Action
Pass
Disabled
Disable this rule
Quick
Apply the action immediately on match.
Interface
LAN
Direction
out
TCP/IP Version
IPv4
Protocol
TCP
Source / Invert
Use this option to invert the sense of the match.
Source
LAN net
Source
Advanced
Destination / Invert
Use this option to invert the sense of the match.
Destination
This Firewall
Destination port range
from:
to:
SSH
SSH

Configuramos la regla como tipo Pass para permitir el tráfico, habilitamos la opción Apply the action immediately on match para que la regla se evalúe y aplique en el momento exacto en que el tráfico coincida con sus condiciones, sin esperar a evaluar las demás reglas. La dirección la establecemos como Out, indicando que es una regla de salida iniciada desde la LAN. Seleccionamos protocolo TCP, versión IPv4, y en el rango de puertos de destino configuramos SSH to SSH, habilitando únicamente el puerto 22. Con esto, las máquinas de la LAN pueden iniciar conexiones SSH hacia otras máquinas, pero no pueden iniciar otro tipo de conexiones salientes de forma autónoma.

Figura 25

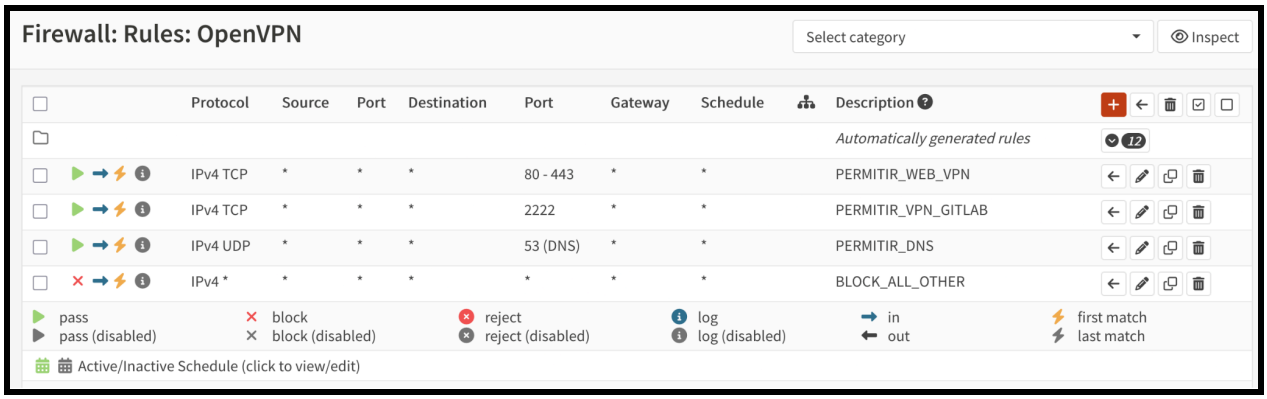
LAN: Visualización de la regla creada



Ahora configuramos las reglas para la interfaz OpenVPN.

Figura 26

OpenVPN: Creación de reglas



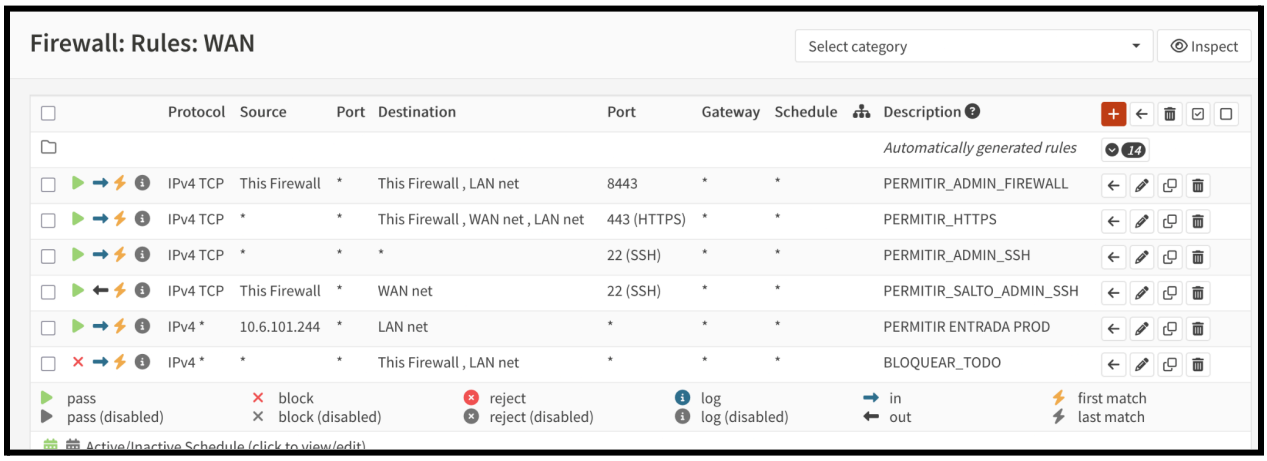
Para la interfaz OpenVPN definimos cuatro reglas evaluadas en orden de arriba hacia abajo. La primera permite tráfico en los puertos 80 y 443 para HTTP y HTTPS respectivamente, que son los únicos protocolos de aplicación que los clientes VPN necesitan para acceder a los servicios del laboratorio. La segunda permite tráfico en el puerto 2222, habilitado específicamente para las operaciones de clonado de repositorios desde GitLab, que por seguridad no usa el puerto SSH estándar. La tercera permite tráfico en el puerto 53, que es el puerto del protocolo DNS, necesario para que los clientes VPN puedan resolver los dominios internos del laboratorio como gitlab.eisi.internal consultando al servidor DNS del firewall en 10.10.10.1. La cuarta y última regla es de tipo Reject y bloquea todo el tráfico restante que no haya coincidido con ninguna de las reglas anteriores.

La lógica de evaluación es secuencial: cuando llega tráfico por la interfaz OpenVPN, OPNsense recorre las reglas de arriba hacia abajo y aplica la primera que coincida con las características del tráfico. Si un cliente VPN intenta iniciar una conexión SSH convencional por el puerto 22, ninguna de las tres primeras reglas coincidirá y llegará inevitablemente a la regla de bloqueo total, siendo rechazado. Esto garantiza que incluso usuarios con acceso VPN válido solo puedan usar los protocolos y puertos explícitamente autorizados.

Ahora configuramos las reglas de la red WAN.

Figura 27

WAN: Creación de reglas



Las reglas de la WAN controlan qué tráfico proveniente de la red universitaria puede entrar al laboratorio. La primera regla permite tráfico con origen y destino en el propio firewall en el puerto 8443, que es el puerto de la consola de administración web de OPNsense. Esta regla está diseñada intencionalmente para que la consola de administración no sea accesible directamente desde internet: para acceder a ella es necesario primero establecer una conexión SSH con reenvío de tipo Socks hacia el firewall, usando una llave privada, y luego acceder a través de esa conexión desde el navegador. Esto eleva considerablemente el nivel de seguridad de la administración del firewall, ya que requiere dos factores de acceso independientes.

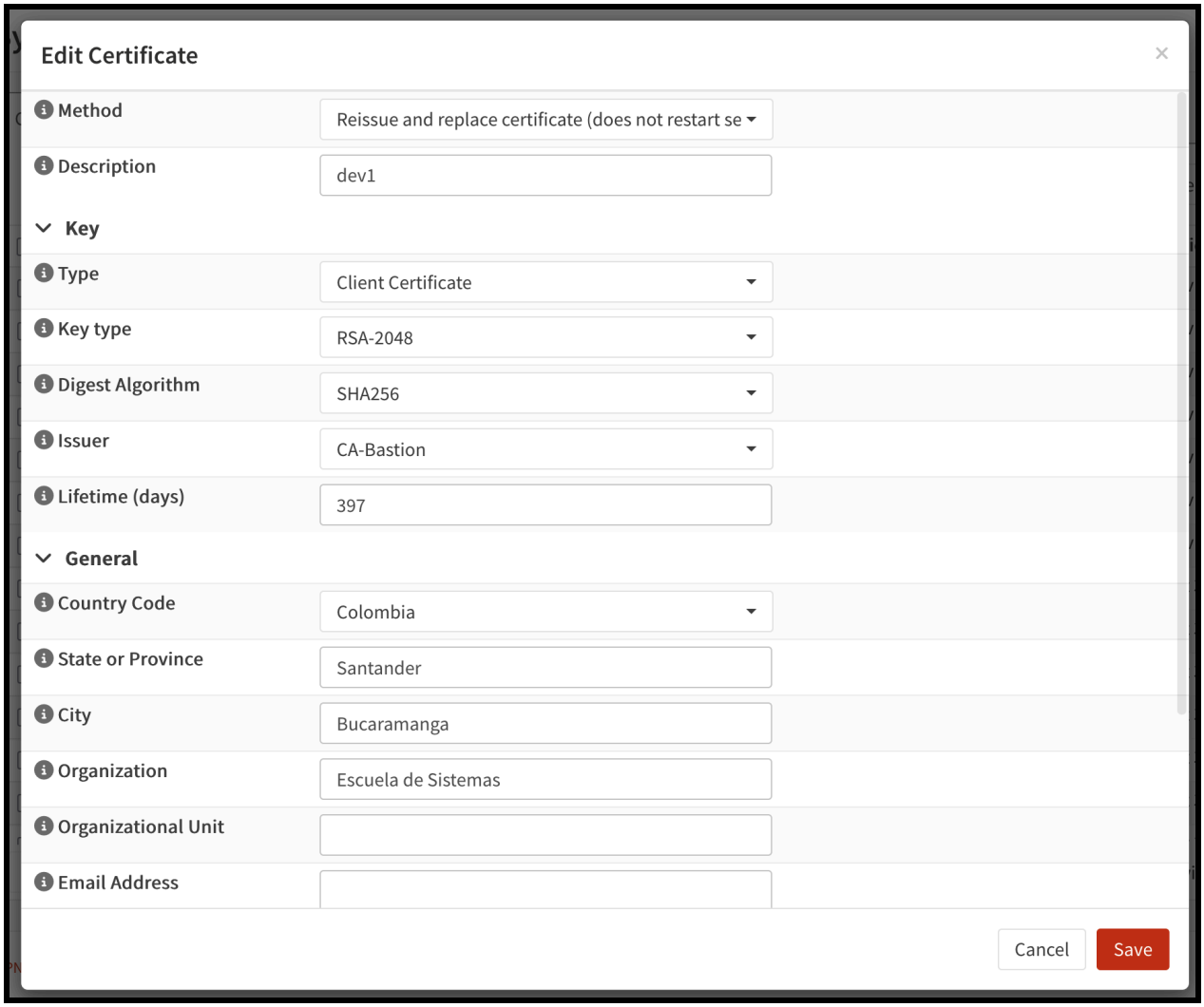
La segunda regla permite todo el tráfico HTTPS entrante por el puerto 443, que es el puerto por donde llegarán las peticiones de los usuarios hacia las aplicaciones desplegadas en el laboratorio. La tercera regla permite tráfico SSH entrante por el puerto 22, habilitando la administración remota de las máquinas del laboratorio desde la red universitaria. La cuarta regla establece una excepción especial para la IP de VM-Prod 10.66.101.244, permitiendo todo el tráfico originado desde esa dirección hacia la red LAN, que es la regla que habilita a VM-Prod

acceder a los recursos del laboratorio para descargar imágenes contenerizadas. Finalmente, una regla de bloqueo total al final de la lista rechaza cualquier tráfico que no haya coincidido con ninguna de las reglas anteriores, garantizando que ningún acceso no contemplado pueda atravesar el firewall.

Con el servidor OpenVPN configurado y las reglas del firewall establecidas, el último paso es generar las credenciales que permitirán a un usuario conectarse al laboratorio. Cada usuario que requiera acceso necesitará un certificado de cliente único y un archivo de configuración .ovpn que agrupa todas las credenciales necesarias para establecer la conexión. Entonces se sigue a System → Trust → Certificates.

Figura 28

Certificados de cliente



Repetimos el mismo proceso que seguimos para crear el certificado del servidor, con dos diferencias clave. El campo Type esta vez debe ser Client Certificate, indicando que este certificado está destinado a un cliente y no al servidor. El Issuer sigue siendo CA-BASTION, ya que todos los certificados del ecosistema deben estar firmados por la misma autoridad de confianza para que OpenVPN los reconozca como válidos. El campo Common Name es especialmente importante en los certificados de cliente porque actúa como identificador único del usuario dentro del sistema VPN, por lo que debe ser un valor descriptivo y único por persona, como el nombre de usuario o el correo institucional. Al guardar y reabrir el certificado, en la

sección Output encontraremos las llaves pública y privada del cliente, que necesitaremos en el siguiente paso.

Figura 29

Certificados de cliente: Outputs

El archivo .ovpn es el perfil de conexión que el cliente OpenVPN necesita para establecer el túnel con el servidor. Este archivo agrupa en un único lugar toda la información necesaria para la conexión: los parámetros del servidor como la dirección IP, el puerto y el protocolo, el certificado de la CA para verificar la identidad del servidor, el certificado del cliente y su llave privada para que el servidor verifique la identidad del cliente, y la llave estática que cifrará el canal de comunicación. La estructura del archivo sigue un formato estándar de OpenVPN como el siguiente:

Figura 30

Archivo estándar de OpenVPN

```
client
dev tun
proto tcp
remote eisiremoto.uis.edu.co 443
remote 10.6.101.245 443
resolv-retry infinite
nobind
persist-key
persist-tun
cipher AES-256-GCM
auth SHA256
remote-cert-tls server
verb 3


<ca>
 # Aquí pegamos el Certificate Data del CA-Bastion
</ca>

<cert>
 # Aquí pegamos el Certificate Data del certificado del cliente
</cert>

<key>
 # Aquí pegamos el Private Key Data del certificado del cliente
</key>

<tls-crypt>
 # Aquí pegamos la clave estática que generamos para OpenVpn
</tls-crypt>
```

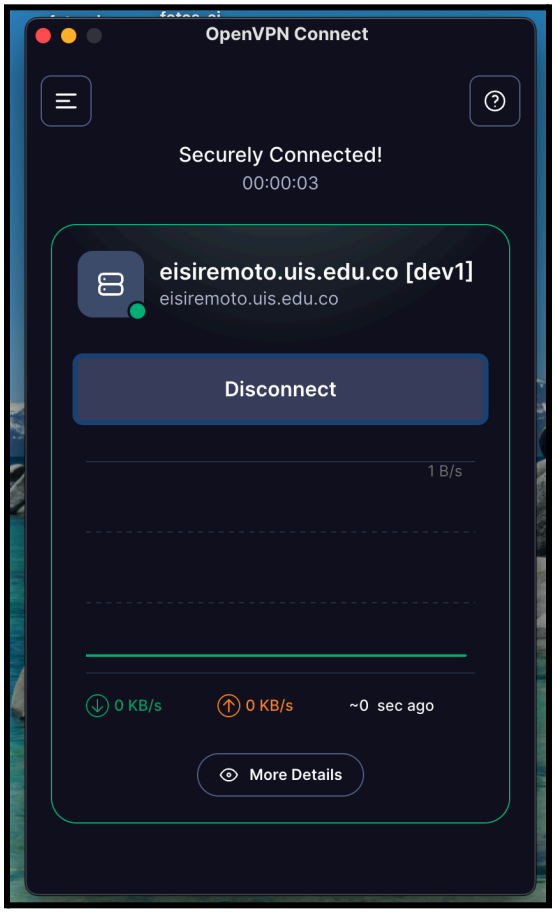
Con el archivo .ovpn armado y guardado con la extensión correcta, el proceso de conexión al laboratorio es sencillo: el usuario descarga el cliente OpenVPN en su dispositivo, agrega un nuevo perfil importando este archivo, e inicia la conexión. Si todos los certificados y parámetros están correctamente configurados, el cliente establecerá el túnel cifrado con

VM-Bastion y el usuario quedará conectado a la red LAN del laboratorio, con acceso a todos sus recursos internos y capacidad de resolver los dominios privados bajo .eisi.internal.

Es importante destacar que este archivo .ovpn es personal e intransferible: contiene la llave privada del certificado del cliente, por lo que compartirlo equivale a compartir el acceso completo al laboratorio. En caso de que un usuario abandone el equipo de desarrollo o sus credenciales se vean comprometidas, el certificado de cliente correspondiente debe ser revocado desde OPNsense, invalidando inmediatamente ese archivo .ovpn sin afectar el acceso de los demás usuarios.

Figura 31

Interfaz de OpenVPN



Con la conexión VPN funcionando correctamente y el acceso a la red LAN garantizado, el siguiente paso es desplegar todos los servicios que conforman el laboratorio de desarrollo. Para esto utilizamos Docker con Docker Compose, que nos permite definir, levantar y gestionar múltiples contenedores de forma declarativa y reproducible. Los servicios que conforman el laboratorio son: Keycloak, GitLab, SonarQube, GitLab Runner, BookStack, Grafana, Hashicorp Vault, Nginx y Planka.

Para que estos servicios puedan comunicarse entre sí de forma interna sin exponer puertos innecesarios al exterior, todos se conectan a una red Docker compartida denominada proxy-network. El acceso desde el exterior hacia cada servicio se gestiona exclusivamente a través de un proxy inverso centralizado, que es el único punto de entrada del laboratorio.

Figura 32

Keycloak y Docker

```yaml
version: '3.8'
services:
  keycloak-db:
    image: postgres:16-alpine
    container_name: keycloak-db
    restart: always
    env_file: .env
    environment:
      POSTGRES_DB: keycloak
      POSTGRES_USER: keycloak
      POSTGRES_PASSWORD: ${KC_DB_PASSWORD}
      TZ: America/Bogota
    volumes:
      - ./db_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U keycloak -d keycloak || exit 1"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - keycloak-network
  keycloak-app:
    image: quay.io/keycloak/keycloak:26.0.4
    container_name: keycloak-app
    restart: always
    env_file: .env
    environment:
      KC_BOOTSTRAP_ADMIN_USERNAME: ${KC_ADMIN_USER}
      KC_BOOTSTRAP_ADMIN_PASSWORD: ${KC_ADMIN_PASSWORD}
      KC_DB: postgres
      KC_DB_USERNAME: keycloak
      KC_DB_PASSWORD: ${KC_DB_PASSWORD}
      KC_DB_URL: jdbc:postgresql://keycloak-db/keycloak
      KC_HOSTNAME: auth.eisi.internal
      KC_HOSTNAME_STRICT: "true"
      KC_HOSTNAME_STRICT_HTTPS: "true"
      KC_HTTP_ENABLED: "true"
      KC_PROXY_HEADERS: xforwarded
      KC_METRICS_ENABLED: "true"
      KC_HEALTH_ENABLED: "true"
    command: ["start"]
    depends_on:
      keycloak-db:
        condition: service_healthy
    expose:
      - "8080"
      - "9000"
    networks:
      - keycloak-network
      - proxy-network
networks:
  keycloak-network:
    driver: bridge
  proxy-network:
    external: true
```

El primer servicio que se levanta es el proxy inverso Nginx, ya que todos los demás servicios dependen de él para ser accesibles. Se despliega mediante Docker Compose y se conecta a la red proxy-network que creamos previamente.

Figura 33

Proxy y Docker

```
services:
  nginx-proxy:
    image: nginx:1.28.0-alpine
    container_name: nginx-proxy
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./ssl:/etc/nginx/ssl:ro
      - ./logs/nginx:/var/log/nginx
    networks:
      - proxy-network
networks:
  proxy-network:
    external: true
volumes:
  nginx_logs:
```

Este es el único contenedor del laboratorio que expone puertos al exterior: el 80 para HTTP y el 443 para HTTPS. Todos los demás servicios permanecen en la red interna de Docker sin puertos expuestos directamente, y son accesibles únicamente a través del proxy. Dado que el laboratorio es inaccesible desde internet público sin VPN, los certificados SSL se generan como autofirmados, lo cual es suficiente para garantizar comunicación cifrada dentro de este entorno privado. La configuración de Nginx para cada servicio sigue siempre el mismo patrón de tres bloques. El bloque upstream define el nombre del contenedor de destino y su puerto interno, creando una referencia nombrada que el proxy puede usar sin necesidad de conocer detalles de red. El bloque de redirección HTTP implementa HSTS estricto: cualquier petición que llegue por

el puerto 80 es redirigida automáticamente al puerto 443, forzando siempre comunicación cifrada por TLS. El bloque de proxy pass es el núcleo de la configuración: escucha en el puerto 443, resuelve TLS usando los certificados autofirmados, define los protocolos y el tamaño máximo del cuerpo de las peticiones, y enruta cualquier ruta entrante hacia el upstream correspondiente, enriqueciendo cada petición con cabeceras que informan al servicio destino sobre el protocolo original, el host, la IP del cliente y la versión HTTP utilizada.

Figura 34

Upstream, HTTP y TLS

```
# Keycloak upstream
upstream keycloak {
    server keycloak-app:8080;
}
# Redirigir HTTP a HTTPS para Keycloak
server {
    listen 80;
    server_name auth.eisi.internal;
    return 301 https://$server_name$request_uri;
}
# Keycloak proxy pass
server {
    listen 443 ssl;
    server_name auth.eisi.internal;
    ssl_certificate /etc/nginx/ssl/nginx-keycloak.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx-keycloak.key;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    client_max_body_size 20M;
    location / {
        proxy_pass http://keycloak;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port 443;
        proxy_set_header Upgrade $http_upgrade;
        proxy_read_timeout 90s;
    }
}
```

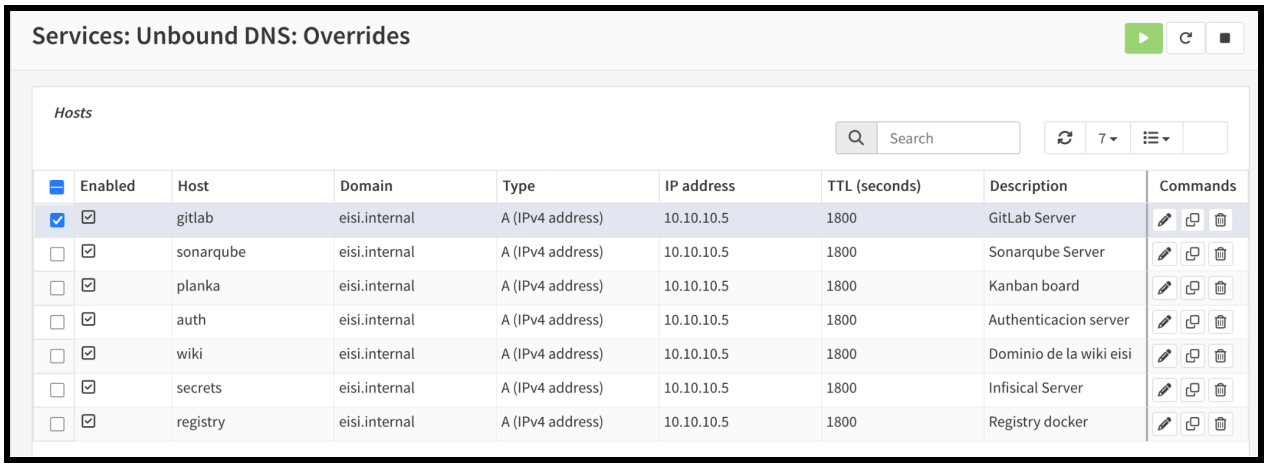
Para que todos estos servicios sean accesibles a través de sus dominios internos cuando un usuario está conectado por VPN, es necesario registrar cada dominio en el servidor DNS de OPNsense. Como se configuró anteriormente, cuando un cliente VPN activa su conexión, su dispositivo queda instruido para resolver cualquier dominio bajo .eisi.internal consultando al firewall en 10.10.10.1. Sin embargo, para que el firewall pueda responder a esas consultas, cada dominio debe estar registrado manualmente apuntando a la IP de VM-Lab donde residen los servicios.

Esto significa que dominios como gitlab.eisi.internal, sonarqube.eisi.internal, secrets.eisi.internal, auth.eisi.internal o registry.eisi.internal deben tener un registro DNS en OPNsense apuntando a 10.10.10.5, que es la IP de VM-Lab. Cuando un desarrollador conectado por VPN escribe cualquiera de estos dominios en su navegador, su dispositivo consulta al firewall, este responde con la IP correspondiente y la petición llega al proxy inverso de Nginx en VM-Lab, que la enruta al contenedor correcto según el dominio solicitado.

Sin este registro en el DNS del firewall, los dominios internos serían irresolubles incluso estando conectado por VPN, ya que no existen en ningún servidor DNS público y sólo tienen sentido dentro del contexto privado del laboratorio.

Figura 35

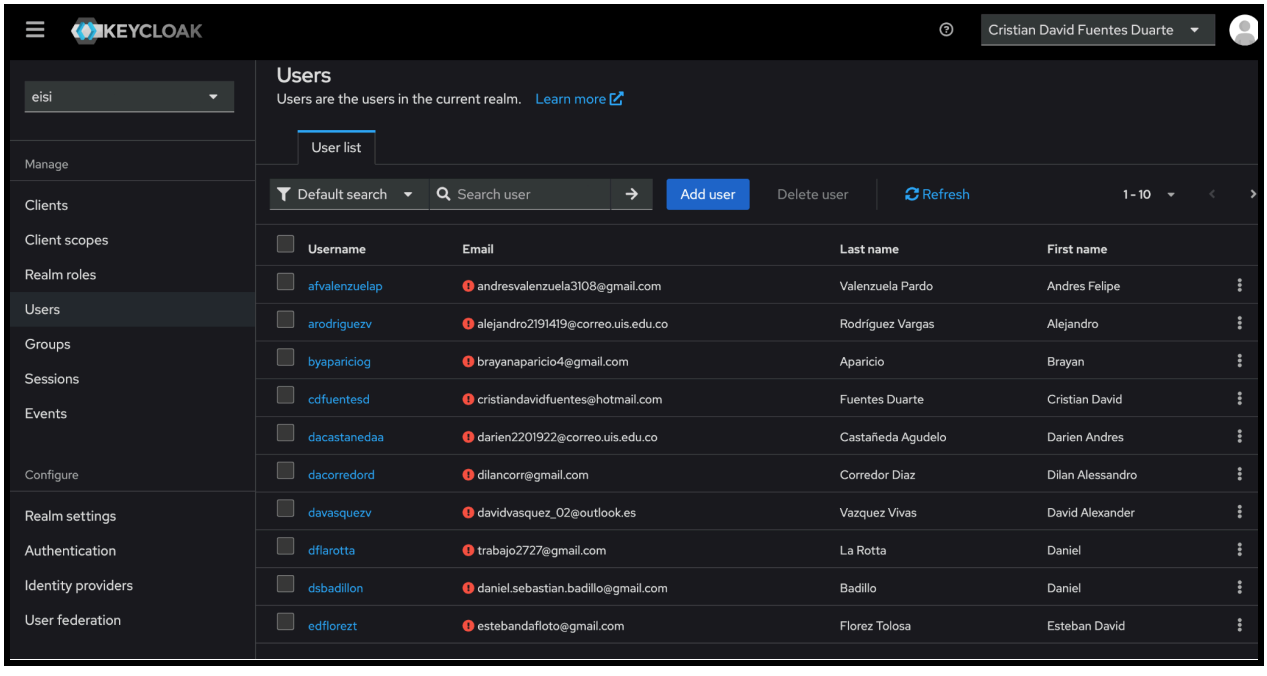
Visualización del DNS



Con este patrón establecido se puede verificar que Keycloak está funcionando correctamente accediendo a su dominio interno desde un navegador conectado por VPN. El navegador mostrará una advertencia de seguridad por el certificado autofirmado, que debe aceptarse para continuar, comportamiento esperado y normal en entornos privados que no usan certificados emitidos por una CA pública.

Figura 36

Servicio de Auth (Keycloak)

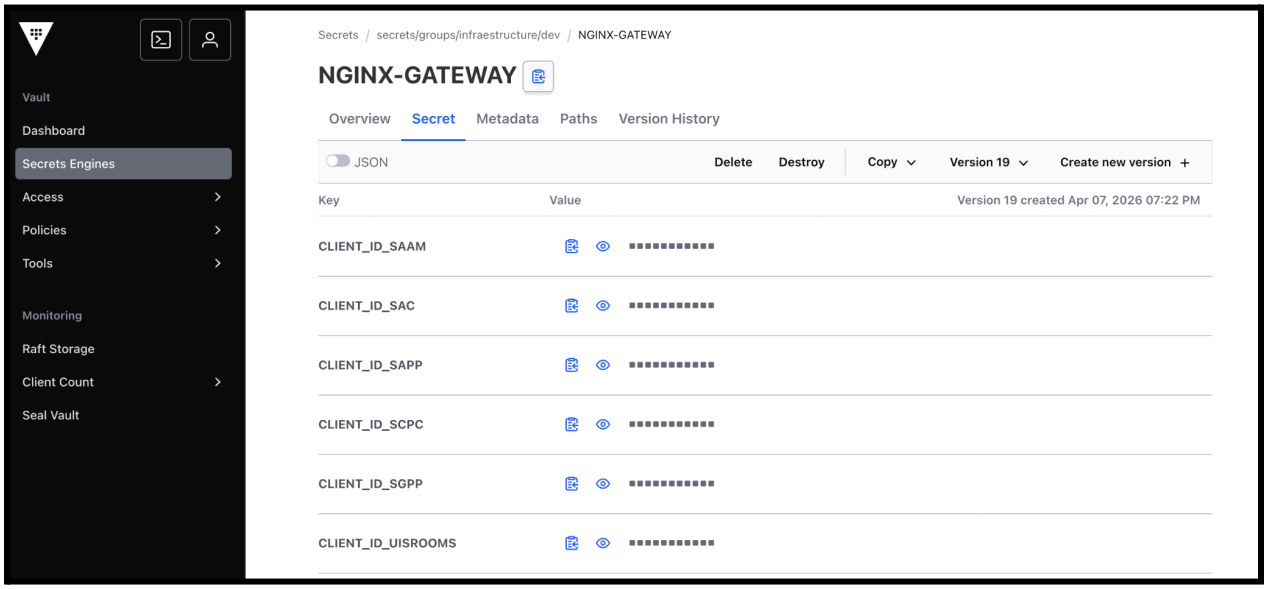


Keycloak cumple un rol central en el laboratorio más allá de ser el IdP de las aplicaciones en desarrollo: también es el proveedor de identidad de todas las herramientas internas del laboratorio, como GitLab, SonarQube, Grafana y Planka. Esto significa que los equipos de desarrollo solo necesitan un único usuario y contraseña para acceder a todas estas plataformas. La Escuela crea los usuarios en Keycloak con contraseñas temporales y los entrega a cada equipo, quienes deberán cambiarlas en su primer inicio de sesión.

El resto de los servicios se levanta siguiendo el mismo patrón descrito: Docker Compose, conexión a proxy-network, configuración de Nginx con sus tres bloques, e integración con Keycloak como proveedor de identidad. Con todos los servicios operativos, el laboratorio está listo para soportar el ciclo completo de desarrollo.

Figura 37

Servicio de Vault (Secrets)



Uno de los problemas más comunes en equipos de desarrollo es el manejo inadecuado de información sensible como contraseñas, claves de API, cadenas de conexión a bases de datos y

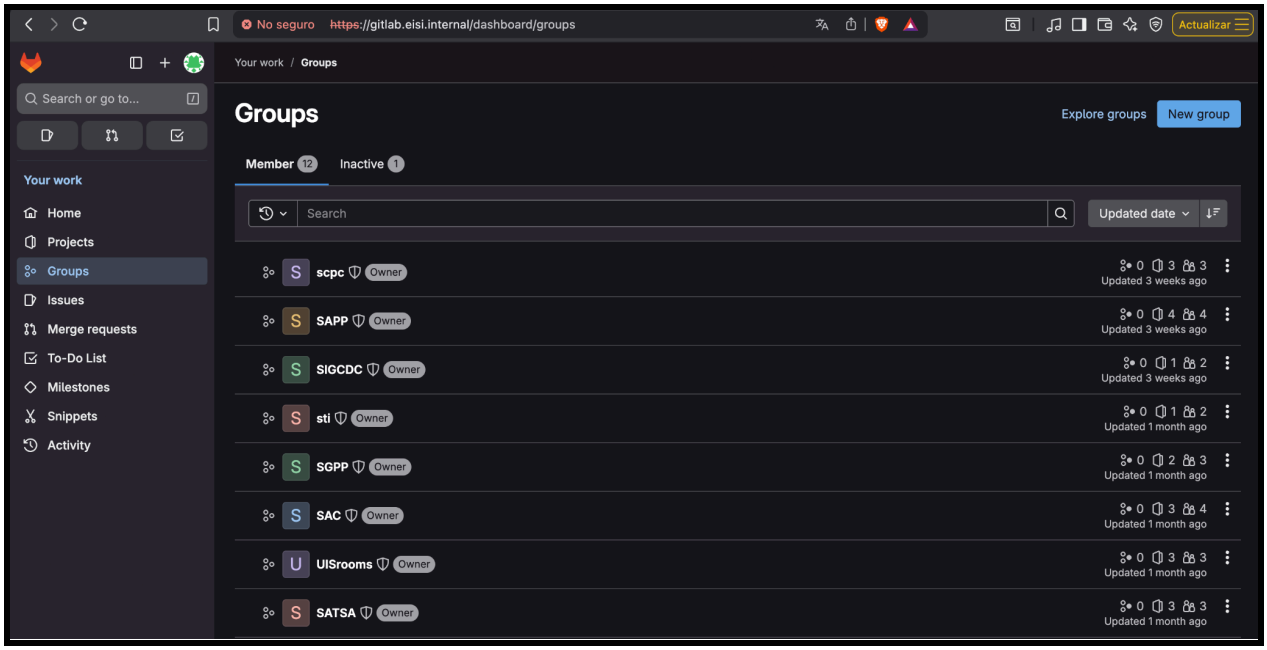
tokens de autenticación. La práctica de incluir estos valores directamente en el código fuente o en archivos de configuración versionados representa un riesgo de seguridad significativo, ya que cualquier persona con acceso al repositorio puede ver esa información.

Para resolver esto, el laboratorio cuenta con Hashicorp Vault, un gestor centralizado de secretos que actúa como bóveda segura para toda la información sensible de los proyectos. El principio de funcionamiento es simple: en el código y en los pipelines de despliegue se hace referencia al nombre del secreto, nunca a su valor. En el momento del despliegue, el sistema se conecta a Vault, se autentica con las credenciales del equipo y descarga el valor real del secreto en tiempo de ejecución. De esta forma, ningún valor sensible queda expuesto en el repositorio ni en los archivos de configuración versionados.

A cada equipo de desarrollo se le crean credenciales de acceso propias a Vault, garantizando que cada equipo solo pueda ver y gestionar sus propios secretos. Adicionalmente, se configuran tres motores de secretos independientes por equipo, uno por cada ambiente: dev, stg y prod. Esto permite que cada equipo defina de forma autónoma los valores que necesita para cada ambiente, por ejemplo una cadena de conexión a una base de datos de desarrollo diferente a la de producción, sin que esos valores sean visibles para otros equipos ni queden registrados en ningún repositorio de código.

Figura 38

Servicio de GitLab



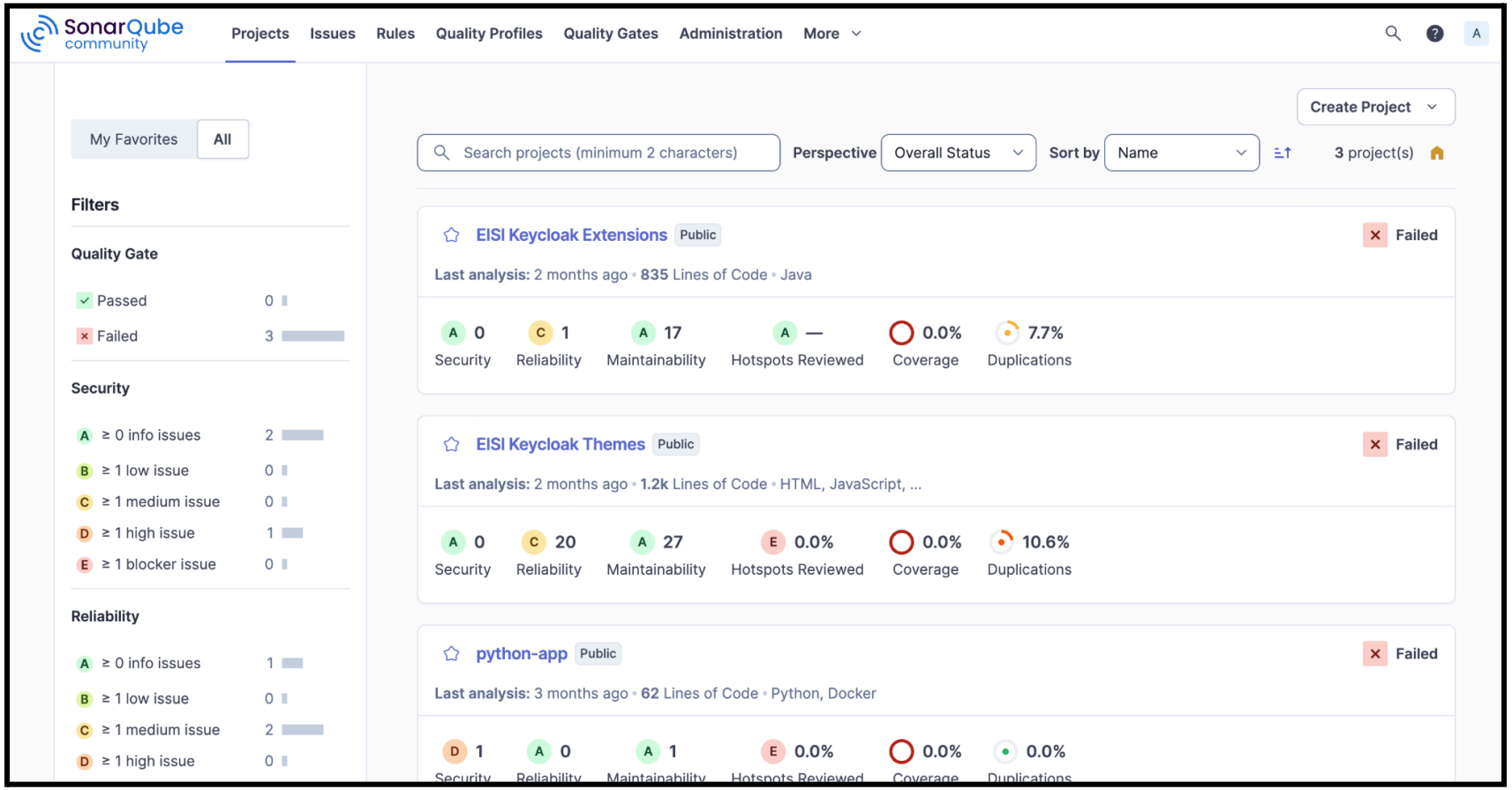
GitLab es el núcleo del laboratorio de desarrollo. Es la plataforma donde reside y se versiona todo el código fuente de los proyectos de la Escuela, y desde donde se orquestan los procesos automáticos de integración y despliegue continuo. A diferencia del uso de repositorios públicos en la nube como GitHub, GitLab se despliega de forma privada dentro del laboratorio, lo que significa que el código de todos los proyectos está bajo el control directo de la Escuela, es accesible únicamente por VPN y nunca sale de la infraestructura institucional.

GitLab organiza el trabajo en grupos y proyectos. A cada equipo de desarrollo se le asigna su propio grupo, dentro del cual pueden crear los proyectos que necesiten, gestionar el acceso de sus integrantes, organizar su código y explorar las funcionalidades de la plataforma con total autonomía. Este aislamiento por grupo garantiza que cada equipo trabaje en su propio espacio sin interferir con los demás, pero bajo una plataforma común que la Escuela puede supervisar y administrar de forma centralizada.

Más allá del versionamiento de código, GitLab actúa como el motor de integración y despliegue continuo del laboratorio. Cada vez que un equipo sube cambios a su repositorio, GitLab puede ejecutar automáticamente una secuencia de pasos definidos en un pipeline: compilar el código, ejecutar pruebas, analizar la calidad con SonarQube, construir la imagen contenerizada y desplegarla en el ambiente correspondiente. Esto elimina los procesos manuales de despliegue, reduce el margen de error humano y garantiza que cada entrega pase por los mismos controles de calidad sin excepción. Al igual que todos los servicios del laboratorio, GitLab se autentica a través de Keycloak, por lo que los equipos acceden con el mismo usuario institucional que usan para las demás herramientas, sin necesidad de crear credenciales adicionales.

Figura 39

Servicio de Sonar


SonarQube community
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
A
Create Project
My Favorites
All
Search projects (minimum 2 characters)
Perspective
Overall Status
Sort by
Name
3 project(s)
Filters
Quality Gate
Passed 0
Failed 3
Security
A ≥ 0 info issues 2
B ≥ 1 low issue 0
C ≥ 1 medium issue 0
D ≥ 1 high issue 1
E ≥ 1 blocker issue 0
Reliability
A ≥ 0 info issues 1
B ≥ 1 low issue 0
C ≥ 1 medium issue 2
D ≥ 1 high issue 0
EISI Keycloak Extensions
Public
Failed
Last analysis: 2 months ago • 835 Lines of Code • Java
A 0 Security
C 1 Reliability
A 17 Maintainability
A — Hotspots Reviewed
0.0% Coverage
7.7% Duplications
EISI Keycloak Themes
Public
Failed
Last analysis: 2 months ago • 1.2k Lines of Code • HTML, JavaScript, ...
A 0 Security
C 20 Reliability
A 27 Maintainability
E 0.0% Hotspots Reviewed
0.0% Coverage
10.6% Duplications
python-app
Public
Failed
Last analysis: 3 months ago • 62 Lines of Code • Python, Docker
D 1 Security
A 0 Reliability
A 1 Maintainability
E 0.0% Hotspots Reviewed
0.0% Coverage
0.0% Duplications

SonarQube es la plataforma de análisis estático de código del laboratorio. Su propósito es inspeccionar automáticamente el código fuente de cada proyecto en busca de problemas que podrían comprometer la calidad, la seguridad o la mantenibilidad del software antes de que lleguen a un ambiente productivo.

Cada vez que un equipo sube cambios a su repositorio en GitLab, el pipeline de integración continua envía el código a SonarQube para su análisis. La plataforma examina el código y genera un reporte detallado que puede incluir bugs, vulnerabilidades de seguridad, código duplicado, malas prácticas de programación y deuda técnica acumulada. Con base en estos resultados, el pipeline puede configurarse para bloquear un despliegue si el código no supera los umbrales mínimos de calidad definidos, garantizando que ninguna entrega con problemas críticos avance al siguiente ambiente.

Para los estudiantes, SonarQube representa una herramienta de aprendizaje continuo: los reportes no solo señalan qué está mal sino que explican por qué es un problema y cómo corregirlo, lo que contribuye directamente a la formación en buenas prácticas de programación. Cada equipo tiene acceso a su propio proyecto en SonarQube, donde puede hacer seguimiento a la evolución de la calidad de su código a lo largo del tiempo. Al igual que los demás servicios del laboratorio, SonarQube se integra con Keycloak como proveedor de identidad, permitiendo que los equipos accedan con sus credenciales institucionales sin necesidad de gestionar usuarios adicionales.

Figura 40

Configuración Gitlab Runner

```
services:
  gitlab-runner:
    image: gitlab/gitlab-runner:alpine
    container_name: gitlab-runner
    restart: always
    extra_hosts:
      - "gitlab.eisi.internal:10.10.10.5"
      - "secrets.eisi.internal:10.10.10.5"
      - "registry.eisi.internal:10.10.10.5"
      - "sonarqube.eisi.internal:10.10.10.5"
    volumes:
      - ./config:/etc/gitlab-runner
      - /var/run/docker.sock:/var/run/docker.sock
      - ./config/certs:/usr/local/share/ca-certificates:ro
```

```
EISI@VM-LAB:~$ sudo cat gitlab-runner/config/config.toml
concurrent = 1
check_interval = 0
shutdown_timeout = 0
[session_server]
  session_timeout = 1800
[[runners]]
  name = "shared-runner"
  url = "https://gitlab.eisi.internal"
  id = 1
  token = "glrt-7UtTfk0k9QBmyIyKAZ7Pj286MQpwOjgKdDozCnU6Yg8"
  token_obtained_at = 2025-11-14T06:51:21Z
  token_expires_at = 0001-01-01T00:00:00Z
  executor = "docker"
  [runners.cache]
    MaxUploadedArchiveSize = 0
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
  [runners.docker]
    tls_verify = false
    image = "alpine:3.20"
    privileged = false
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/cache","/var/run/docker.sock:/var/run/docker.sock"]
    shm_size = 0
    network_mtu = 0
    extra_hosts = ["gitlab.eisi.internal:10.10.10.5",
"sonarqube.eisi.internal:10.10.10.5","secrets.eisi.internal:10.10.10.5","registry.eisi.internal:1
0.10.10.5"]
```

GitLab Runner es el agente encargado de ejecutar físicamente los pipelines de integración y despliegue continuo definidos en cada proyecto de GitLab. Mientras GitLab orquesta y define

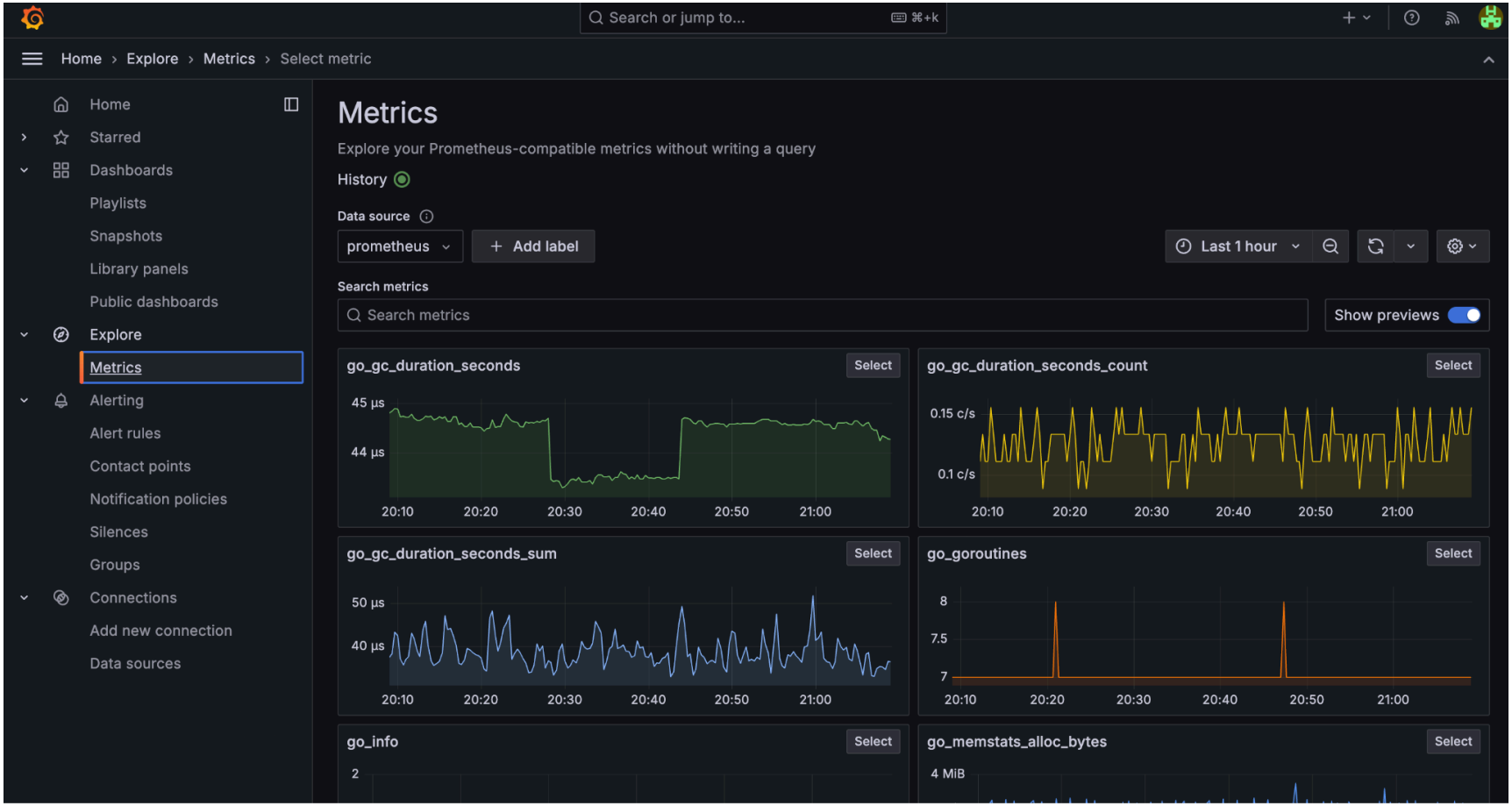
qué debe ejecutarse, el Runner es quien realmente corre los trabajos: compila el código, ejecuta las pruebas, construye las imágenes Docker y despliega los servicios en los ambientes correspondientes.

Se despliega una instancia de GitLab Runner configurada como shared runner, lo que significa que está disponible para todos los proyectos y grupos del laboratorio sin necesidad de que cada equipo configure su propio runner. El executor configurado es de tipo Docker, lo que significa que cada job del pipeline se ejecuta dentro de un contenedor aislado y efímero que se crea al inicio del job y se destruye al finalizar, garantizando que cada ejecución parte de un ambiente limpio y reproducible sin interferencias entre proyectos.

Un aspecto importante de esta configuración es la resolución de dominios internos. Dado que el Runner opera dentro de un contenedor Docker y necesita comunicarse con otros servicios del laboratorio como GitLab, SonarQube, Vault y el registro de imágenes, se configuran entradas de extra_hosts tanto en el contenedor del Runner como en los contenedores que este levanta durante los jobs. Estas entradas mapean los dominios internos del laboratorio como gitlab.eisi.internal o sonarqube.eisi.internal a la IP de VM-Lab 10.10.10.5, permitiendo que el Runner y sus jobs puedan resolver estos dominios privados sin depender del DNS externo. Adicionalmente, el Runner tiene acceso al socket de Docker del host mediante el volumen /var/run/docker.sock, lo que le permite construir y publicar imágenes contenerizadas directamente desde los pipelines hacia el registro interno del laboratorio.

Figura 41

Servicio de Grafana


Search or jump to... ⌘+k
Home › Explore › Metrics › Select metric
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Metrics
Alerting
Alert rules
Contact points
Notification policies
Silences
Groups
Connections
Add new connection
Data sources
Metrics
Explore your Prometheus-compatible metrics without writing a query
History
Data source
prometheus
+ Add label
Last 1 hour
Search metrics
Search metrics
Show previews
go_gc_duration_seconds
Select
45 µs
44 µs
20:10
20:20
20:30
20:40
20:50
21:00
go_gc_duration_seconds_count
Select
0.15 c/s
0.1 c/s
20:10
20:20
20:30
20:40
20:50
21:00
go_gc_duration_seconds_sum
Select
50 µs
40 µs
20:10
20:20
20:30
20:40
20:50
21:00
go_goroutines
Select
8
7.5
7
20:10
20:20
20:30
20:40
20:50
21:00
go_info
Select
2
go_memstats_alloc_bytes
Select
4 MiB

En un ecosistema de producción, el acceso directo a los servidores está restringido por razones de seguridad: los equipos de desarrollo no tienen acceso SSH ni consola sobre las máquinas productivas, lo que significa que no pueden inspeccionar logs, revisar el estado de los procesos ni diagnosticar problemas directamente sobre el servidor. Grafana resuelve esta limitación siendo los ojos del laboratorio y de los equipos de desarrollo sobre todos los ambientes.

Grafana es la plataforma centralizada de monitoreo y observabilidad del laboratorio. Desde su interfaz los equipos pueden visualizar en tiempo real el rendimiento de las máquinas virtuales, el consumo de recursos como CPU, memoria y almacenamiento, el estado de los contenedores y servicios desplegados, y los logs generados por cada aplicación. Toda esta información se presenta en tableros de control configurables que permiten tener una visión clara del estado del ecosistema sin necesidad de acceder directamente a ningún servidor.

Más allá del monitoreo reactivo, Grafana permite establecer reglas de detección de comportamientos anómalos y configurar alertas automáticas que notifican al equipo cuando algún indicador supera umbrales definidos, como un pico inusual en el consumo de memoria, una tasa de errores elevada en algún servicio o una caída en el tiempo de respuesta de una aplicación. Esto transforma el monitoreo de una actividad pasiva a una capacidad proactiva de detección temprana de problemas.

En el contexto de seguridad, Grafana es un componente clave: la centralización de logs permite correlacionar eventos entre diferentes servicios, identificar patrones de acceso

sospechosos y mantener un registro auditable de lo que ocurre en la infraestructura, aspectos fundamentales para cualquier ecosistema que maneje información institucional sensible.

Resolución de Dominios Internos en VM-Lab: Dado que los servicios del laboratorio se comunican entre sí a través del proxy inverso usando sus dominios internos, VM-Lab necesita ser capaz de resolver sus propios dominios para que esas peticiones lleguen correctamente al proxy y este las enrute al contenedor correspondiente. Para lograr esto se configuran los registros de hosts en la plantilla /etc/cloud/templates/hosts.debian.tmpl, apuntando cada dominio interno a 10.10.10.5, que es la propia IP de VM-Lab. De esta forma, cuando cualquier servicio o proceso dentro de la máquina haga una solicitud a gitlab.eisi.internal o sonarqube.eisi.internal, la máquina sabrá que debe dirigir esa petición hacia sí misma, donde el proxy inverso Nginx la recibirá y enruta el contenedor correcto. Al igual que en VM-Dev, esta configuración se hace sobre la plantilla de cloud-init para garantizar que los registros persistan tras cada reinicio del sistema.

Figura 42

VM-Lab: Dominios internos

```
EISI@VM-LAB:~$ cat /etc/cloud/templates/hosts.debian.tmpl
## template:jinja
{#
This file (/etc/cloud/templates/hosts.debian.tmpl) is only utilized
if enabled in cloud-config.  Specifically, in order to enable it
you need to add the following to config:
   manage_etc_hosts: True
-#}
# Your system has configured 'manage_etc_hosts' as True.
# As a result, if you wish for changes to this file to persist
# then you will need to either
# a.) make changes to the master file in /etc/cloud/templates/hosts.debian.tmpl
# b.) change or remove the value of 'manage_etc_hosts' in
#     /etc/cloud/cloud.cfg or cloud-config from user-data
#
{# The value '{{hostname}}' will be replaced with the local-hostname -#}
127.0.1.1 {{fqdn}} {{hostname}}
127.0.0.1 localhost
# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.10.10.5 registry.eisi.internal
10.10.10.5 gitlab.eisi.internal
10.10.10.5 sonarqube.eisi.internal
10.10.10.5 auth.eisi.internal
10.10.10.5 prometheus-server.eisi.internal
```

Figura 43

Estructura de stack-provisioner

```
stack-provisioner/
├── .gitlab-ci.yml
├── README.md
├── ci/
│   └── provision-common.yml
├── infra-apps/
│   ├── SAAM/
│   │   └── team.yaml
│   ├── SAC/
│   ├── SAPP/
│   └── ...
└── scripts/
    └── provision_teams.sh
```

Una de las metas fundamentales del laboratorio es que sea autogestionable y no dependa de administradores para incorporar nuevos equipos de desarrollo. Sin esta capacidad, cada vez que un nuevo equipo comenzará un proyecto, alguien del equipo de infraestructura tendría que crear manualmente el grupo en GitLab, configurar los permisos, generar las credenciales en Vault, crear los motores de secretos por ambiente y registrar el proyecto en SonarQube, un proceso propenso a errores, inconsistente entre equipos y que no escala.

Para resolver esto se creó Stack Provisioner, una herramienta de automatización basada en el modelo GitOps, donde la infraestructura se define y gestiona a través de archivos declarativos versionados en un repositorio, de la misma forma en que se versiona el código. El repositorio es administrado por el equipo de infraestructura de la Escuela y es el primer proyecto del grupo de infraestructura del laboratorio.

El proceso de incorporación de un nuevo equipo se divide en dos fases. La primera fase es el registro. Cuando un nuevo equipo necesita comenzar a trabajar, un desarrollador crea una carpeta con el nombre del equipo dentro del directorio infra-apps/ del repositorio y coloca dentro un archivo team.yaml con una estructura mínima: el nombre del equipo, su descripción y el usuario de GitLab del responsable. Con ese archivo listo, el desarrollador abre un Merge Request hacia la rama principal del repositorio. Este Merge Request queda sujeto a revisión y aprobación por parte del equipo de infraestructura, garantizando que el proceso tenga un punto de control humano antes de ejecutarse.

Figura 44

Archivo de aprovisionamiento team.yaml

```
team: SAAM
description: sistema de informacion de apoyo y ajuste de matricula
owner: edflorezt
```

La segunda fase se activa automáticamente en el momento en que el Merge Request es aprobado y fusionado a la rama principal. GitLab CI detecta el cambio y ejecuta el script central provision_teams.sh, que itera sobre todas las carpetas de infra-apps/ y por cada equipo encontrado ejecuta cuatro módulos de integración en secuencia.

El primer módulo gestiona la identidad en GitLab: realiza peticiones a la API de GitLab para crear el grupo del equipo si no existe, y asigna automáticamente al usuario declarado en el team.yaml como administrador del grupo, dándole autonomía total para crear proyectos, gestionar integrantes y organizar su trabajo sin depender de ningún administrador.

El segundo módulo configura los permisos de despliegue: mediante una mutación GraphQL, incluye al grupo recién creado en la lista de permitidos del proyecto de despliegue general del laboratorio, habilitando que los pipelines del equipo puedan interactuar con la infraestructura de forma segura y controlada.

El tercer módulo gestiona los secretos en Vault: el script se conecta a Vault y genera de forma automática una contraseña aleatoria y segura de 15 caracteres en hexadecimal para el equipo. Esta contraseña nunca es vista por ningún desarrollador: el script la inyecta directamente como variable enmascarada en la configuración CI/CD del grupo en GitLab bajo el nombre VAULT_PASSWORD, y la usa para crear tres motores de secretos independientes en Vault, uno para cada ambiente: dev, stg y prod. Además configura una política estricta que garantiza que el equipo solo puede leer y escribir dentro de sus propios motores, sin posibilidad de acceder a los secretos de otros equipos.

El cuarto módulo configura la calidad de código en SonarQube: el script se comunica con la API de SonarQube, genera un token de análisis para el equipo y lo almacena como variable enmascarada ‘SONAR_TOKEN’ en el grupo de GitLab. Desde ese momento, cualquier proyecto que el equipo cree tendrá análisis automático de calidad de código disponible sin ninguna configuración adicional.

Todo este proceso, que manualmente podría tomar varias horas distribuidas entre distintos perfiles técnicos, se ejecuta en cuestión de segundos de forma completamente automática. Cada equipo que se incorpora al laboratorio comienza con el mismo nivel de herramientas, permisos y estándares de seguridad, garantizando consistencia en todos los proyectos desde el primer día.

Figura 45

Estructura de stack-infrastructure

```
stack-infrastructure/
├── .gitlab-ci.yml
├── infrastructure/
│   ├── core/
│   └── teams/
│       └── SAAM/
│           └── infra.yaml
└── scripts/
    ├── core.sh
    ├── teams.sh
    ├── catalog/
    │   ├── mongo.sh
    │   ├── minio.sh
    │   └── generic.sh
    └── lib/
```

A medida que el número de equipos de desarrollo crece dentro del laboratorio, la gestión de la infraestructura que cada equipo necesita, bases de datos, volúmenes de almacenamiento, redes internas y servicios genéricos, se convierte rápidamente en un cuello de botella. Sin una solución estructurada, cada equipo terminaría solicitando recursos de forma manual, dependiendo de que alguien del equipo de infraestructura los aprovisiona por SSH, con el riesgo de colisiones de puertos y nombres entre proyectos, credenciales manejadas de forma insegura y configuraciones poco consistentes entre equipos.

Para resolver esto se diseñó Stack Infrastructure, un motor de infraestructura declarativo basado en el modelo GitOps, que permite a los equipos solicitar y aprovisionar sus propios recursos de infraestructura sin intervención manual de ningún administrador. El principio es el mismo que en Stack Provisioner: los equipos no interactúan directamente con los servidores, sino que declaran lo que necesitan en un archivo YAML, y el sistema se encarga de materializar de forma segura, consistente y auditable. Esto lleva a un modelo declarativo: Cada equipo define sus requerimientos de infraestructura en un archivo infra.yaml ubicado dentro de su carpeta en el directorio infrastructure/teams/. En este archivo el equipo especifica los recursos que necesita: redes internas, bases de datos y volúmenes de almacenamiento, referenciando módulos del catálogo de infraestructura sin necesidad de conocer los detalles operativos de Docker ni de los servidores subyacentes.

Figura 46

Archivo infra.yaml

```
group: SAAM
networks:
  - SAAM-internal-net
databases:
  - image: mongo:4.4.29
    service_name: SAAM-mongo-db
    volume: SAAM-mongo-volume
```

El concepto de catálogo es fundamental en este diseño. En lugar de permitir que los equipos especifique parámetros Docker arbitrarios, lo que podría derivar en el uso de imágenes de orígenes no confiables o configuraciones inseguras, el catálogo expone un conjunto de módulos estandarizados y auditados como mongo.sh, minio.sh o postgres.sh. El equipo simplemente indica qué tipo de recurso necesita y el catálogo se encarga de levantarlo con los parámetros correctos, ocultando la complejidad operativa y garantizando que todos los recursos del laboratorio se crean bajo los mismos estándares.

Flujo de ejecución: Cuando un equipo modifica su infra.yaml y fusiona los cambios a la rama principal mediante un Merge Request, GitLab CI activa el pipeline de infraestructura. El motor ejecuta el proceso en varias etapas secuenciales.

Primero realiza una validación sintáctica del archivo YAML para detectar errores de formato antes de intentar cualquier operación. Luego ejecuta un control anti-colisión que verifica que el equipo no esté solicitando un nombre de contenedor o un puerto que ya esté siendo usado por otro equipo en el laboratorio, previniendo conflictos en el servidor compartido.

El paso más importante es el cálculo de estado mediante hashing: el motor toma todas las variables de configuración del equipo, las agrupa y genera un hash SHA-256 que representa el estado deseado de la infraestructura. Luego consulta el servidor remoto para obtener el hash del

estado actual desplegado. Si ambos hashes son idénticos, el motor emite un resultado “No Change” y termina sin ejecutar ninguna operación, evitando despliegues innecesarios que podrían generar interrupciones. Si los hashes difieren, significa que la configuración cambió y el motor procede a aplicar los cambios.

La aplicación remota se realiza de forma eficiente agrupando todas las operaciones necesarias en un único bloque de comandos que se envía al servidor vía SSH en una sola conexión, en lugar de ejecutar múltiples conexiones SSH individuales. Antes de levantar cualquier contenedor, el motor se conecta a Vault para inyectar las credenciales y secretos necesarios directamente en tiempo de ejecución, garantizando que ningún valor sensible quede expuesto en los archivos de configuración ni en los logs del pipeline.

El resultado es que cualquier equipo del laboratorio puede solicitar y actualizar su infraestructura de forma autónoma, con la certeza de que sus recursos están aislados de los demás equipos, sus credenciales son gestionadas de forma segura y cualquier cambio queda registrado y auditado en el historial de GitLab.

Con la infraestructura aprovisionada y los equipos registrados en el laboratorio, el siguiente reto es garantizar que el proceso de llevar código desde el repositorio hasta un ambiente de ejecución sea seguro, automatizado y consistente para todos los proyectos. Este es el propósito de Stack Promote, el motor centralizado de despliegue continuo del laboratorio.

A diferencia de los proyectos anteriores, Stack Promote no almacena código de aplicaciones ni definiciones de infraestructura de equipos específicos. Es un repositorio de lógica de despliegue que otros proyectos consumen desde sus propios pipelines en GitLab CI. Esta centralización tiene una ventaja fundamental: cualquier mejora en la estrategia de despliegue,

como agregar una nueva política de seguridad o mejorar el manejo de secretos, se aplica automáticamente a todos los proyectos del laboratorio sin que cada equipo tenga que actualizar nada en su propio repositorio.

Flujo de despliegue: Cada despliegue se ejecuta en dos etapas secuenciales. La primera es la etapa de Plan, que actúa como una simulación en seco: antes de tocar cualquier contenedor en producción, el sistema valida que los permisos de puertos estén autorizados, que la comunicación con Vault funcione correctamente y que todos los secretos necesarios estén disponibles. Esta etapa permite detectar problemas de configuración sin interrumpir ningún servicio activo.

La segunda etapa es el Deploy, que ejecuta el despliegue real sobre el servidor de destino mediante una conexión SSH segura. Los ambientes de desarrollo se despliegan de forma automática cada vez que hay cambios en el repositorio, implementando Continuous Deployment real. Los ambientes de staging y producción requieren aprobación manual explícita antes de ejecutarse, estableciendo un punto de control deliberado para las entregas a ambientes críticos.

Integración segura de secretos: Antes de levantar cualquier contenedor, el motor se conecta a Vault y descarga dinámicamente los secretos correspondientes al ambiente de destino. Estos valores se inyectan como variables de entorno del contenedor exclusivamente en tiempo de ejecución, nunca quedan expuestos en los logs del pipeline ni en los archivos de configuración. Durante la etapa de Plan, los valores de los secretos son enmascarados mostrándose únicamente como VAULT_VERIFIED_HIDDEN, confirmando que la conexión con Vault funciona sin revelar el contenido.

Una de las características más importantes del motor es su capacidad de recuperación ante fallos, implementada mediante un mecanismo de Blue-Green local. Antes de iniciar

cualquier despliegue, el contenedor activo no es eliminado sino renombrado con el sufijo ‘_rollback’, manteniéndolo disponible como respaldo. El nuevo contenedor se levanta y el motor lo monitorea durante los primeros segundos consultando la API de Docker cada 5 segundos para verificar su estado de salud, conteo de reinicios y posibles fallos.

Si el nuevo contenedor falla al iniciar o muestra inestabilidad, el motor ejecuta automáticamente el rollback: destruye la versión defectuosa, reactiva inmediatamente el contenedor de respaldo y marca el pipeline como fallido para notificar al equipo. El resultado es que el servicio nunca queda inactivo durante un despliegue fallido, los usuarios finales no perciben ninguna interrupción y el equipo recibe la notificación del problema para corregirlo antes del siguiente intento.

Adicionalmente, el motor detecta si el contenedor a desplegar corresponde a una migración de base de datos mediante la bandera ”Is Liquibase”. En ese caso omite completamente el ciclo Blue-Green y ejecuta el contenedor de migración en primer plano de forma efímera, esperando a que termine antes de continuar. Si la migración falla, el flujo se detiene completamente, protegiendo la integridad de los datos.

Para evitar que los equipos expongan puertos de forma arbitraria en los servidores compartidos, el motor implementa una lista blanca de puertos autorizados definida en el archivo ‘authorized_repos_ports.txt’. Si un equipo intenta abrir un puerto de host que no está en esa lista, el despliegue es rechazado antes de ejecutarse. Esto garantiza que los servicios internos permanezcan inaccesibles desde el exterior a menos que estén explícitamente autorizados, reduciendo la superficie de ataque del ecosistema.

Para los ambientes más restringidos como producción, donde el servidor no es directamente accesible desde el runner de GitLab, el motor configura automáticamente el salto a través de VM-Bastion mediante ProxyJump SSH, estableciendo la conexión en dos saltos sin necesidad de que el equipo configure nada adicional.

El conjunto de estos tres proyectos, Stack Provisioner, Stack Infrastructure y Stack Promote, forma el ecosistema de automatización del laboratorio: el primero incorpora equipos, el segundo aprovisiona sus recursos, y el tercero lleva su código hasta producción de forma segura y repetible.

Uno de los retos más comunes en entornos con múltiples equipos de desarrollo es la fragmentación: cada proyecto termina definiendo su propio proceso de construcción, sus propias reglas de calidad y su propia lógica de despliegue, lo que genera inconsistencias, duplicación de esfuerzo y dificultad para mantener estándares comunes. El repositorio gitlab-ci-templates resuelve este problema actuando como el núcleo centralizado de estandarización para todos los flujos de Integración y Despliegue Continuo del laboratorio.

En lugar de que cada equipo construya su pipeline desde cero, este repositorio provee plantillas modulares y reutilizables que los proyectos simplemente incluyen en su configuración de GitLab CI. La ventaja fundamental de esta centralización es que cualquier mejora en el proceso, como actualizar la versión del analizador de código, refinar la lógica de versionado de imágenes o agregar una nueva política de seguridad, se realiza una sola vez en este repositorio y se propaga automáticamente a todos los proyectos que lo consumen. Los desarrolladores de aplicaciones no necesitan ser expertos en infraestructura ni en configuraciones avanzadas de

CI/CD; su responsabilidad se reduce a incluir las plantillas correspondientes y declarar los parámetros propios de su servicio.

Estrategias de flujo de trabajo: Los pipelines están diseñados para adaptarse al modelo de control de versiones que cada equipo elija, controlado mediante la variable “WorkFlow Strategy”. La primera estrategia soportada es Trunk-Based Development, pensada para equipos que integran código frecuentemente sobre una rama principal. En este modelo, cada push a main activa automáticamente el ambiente de desarrollo, un trigger manual desde la interfaz web activa el ambiente de staging, y la creación de un nuevo tag semántico como v1.0.0 desencadena el despliegue a producción. La segunda estrategia es GitFlow, orientada a proyectos con ciclos de liberación más estructurados, donde las ramas de feature apuntan a desarrollo, las ramas de staging o release apuntan al ambiente de pruebas, y al igual que en la estrategia anterior, solo un tag explícito habilita el despliegue productivo.

Módulos del repositorio: El repositorio organiza su lógica en cuatro directorios que corresponden a las dimensiones del ciclo de vida del software. El directorio build/ contiene plantillas ajustadas por ecosistema tecnológico, como NestJS, Maven o React, cada una encargada de ejecutar las herramientas de compilación propias del lenguaje y de configurar variables críticas según el ambiente destino, como establecer “Node Env=production” para builds de staging y producción. El resultado de esta etapa son artefactos compilados que quedan disponibles para las etapas siguientes del pipeline.

El directorio security/ alberga la integración con SonarQube para el análisis estático de código, garantizando estándares de calidad técnica, cobertura de pruebas y detección temprana de vulnerabilidades en cada ciclo de integración. Esta plantilla también se encarga de instalar

automáticamente los certificados TLS necesarios para comunicarse con el servidor interno del laboratorio, ocultando completamente esa complejidad al desarrollador.

El directorio docker/ contiene la plantilla responsable de la contenerización del artefacto construido. Utilizando Docker-in-Docker, genera la imagen del servicio y le asigna un tag apropiado según el contexto: si el pipeline fue disparado por un tag de versión, la imagen recibe ese tag explícito; si corresponde a desarrollo o staging, se etiqueta con el hash corto del commit, lo que permite trazabilidad total entre una imagen desplegada y el código exacto que la originó. La imagen resultante es publicada en el registro privado de contenedores del laboratorio.

Finalmente, el directorio deploy/ contiene la plantilla de orquestación, que es el corazón del flujo. En lugar de ejecutar comandos de despliegue directamente desde el repositorio de la aplicación, esta plantilla actúa como un disparador remoto: realiza una llamada HTTP a la API de GitLab para activar el pipeline del repositorio central de infraestructura, enviando como payload los metadatos necesarios para el despliegue, como el ambiente destino, el nombre de la aplicación, la imagen generada, los puertos, volúmenes y variables adicionales. Cuando el servicio requiere secretos, esta etapa formula la ruta dinámica correspondiente en el clúster de HashiCorp Vault y la delega al orquestador de infraestructura, garantizando que los valores sensibles nunca transiten por los pipelines intermedios.

Separación de responsabilidades: Este modelo establece una separación clara de responsabilidades entre tres capas: el repositorio de la aplicación define qué se construye, gitlab-ci-templates define el estándar de cómo construirlo y validarlo, y el repositorio de infraestructura se encarga de desplegarlo. Esta arquitectura también garantiza escalabilidad tecnológica: incorporar soporte para un nuevo lenguaje como Golang implica únicamente

agregar un archivo en el directorio build/, sin afectar en nada la lógica de contenedorización ni de despliegue que es compartida por todos los proyectos.

Desde la perspectiva del desarrollador, toda esta maquinaria queda abstraída en pocas líneas de configuración. Un proyecto que requiera el ciclo completo de análisis, construcción y despliegue solo necesita declarar qué plantillas incluir y los parámetros propios de su servicio, como su nombre, red y puertos, delegando el resto al estándar del laboratorio.

La gestión de identidades y autenticación es uno de los componentes más críticos en cualquier sistema de software. Ante la posibilidad de desarrollar un Identity Provider completamente desde cero, la decisión técnica fue clara: hacerlo representaría una irresponsabilidad de ingeniería. Construir un sistema de autenticación propio implica asumir el riesgo de introducir vulnerabilidades desconocidas, carecer de los controles de seguridad que solo se adquieren con años de auditorías y pruebas en producción, y prescindir de la comunidad activa que alerta y responde ante nuevas amenazas. En seguridad, la trayectoria y el historial de parches de un software no son opcionales, son garantías fundamentales.

Por estas razones, el laboratorio adopta Keycloak como base del sistema de autenticación. Keycloak es un software open source con amplia adopción empresarial, sometido continuamente a controles de seguridad, actualizaciones y correcciones de vulnerabilidades. La estrategia no es reemplazarlo sino extenderlo: adaptar sus funcionalidades al contexto universitario mediante extensiones y personalizaciones que se construyen sobre una base probada, sin comprometer la solidez del núcleo.

Siguiendo el proceso establecido por el ecosistema de automatización del laboratorio, el primer paso es registrar el equipo responsable del servicio mediante Stack Provisioner. El grupo se crea

con el nombre autenticador, con descripción de servicio de autenticador y bajo la responsabilidad del propietario correspondiente. Con un único commit a la rama principal, el sistema aprovisiona automáticamente el grupo junto con todas las variables necesarias para su operación. Es importante destacar que en ningún momento se exponen claves SSH privadas en este proceso; la gestión segura de credenciales sensibles queda delegada al repositorio central de promoción, que es el único autorizado para operar con ellas.

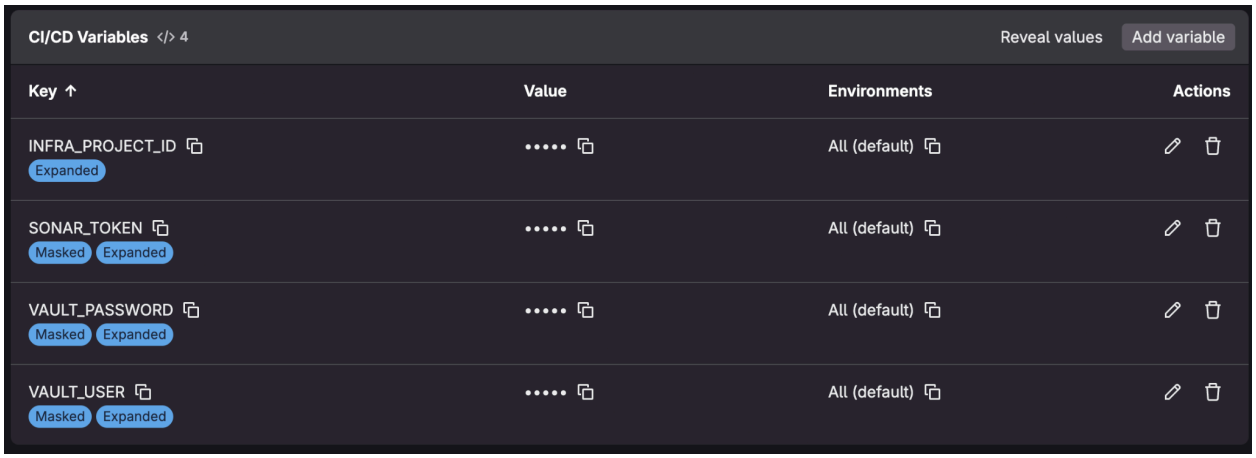
Figura 47

Ejemplo de team.yaml

```
team: autenticador
description: servicio de autenticador
owner: cdfuentesd
```

Figura 48

Variables de aprovisionamiento



Con el equipo registrado, se procede a crear el proyecto keycloak-manager, que actúa como el repositorio central de toda la lógica de Keycloak en el laboratorio. Este proyecto es un

mono-repositorio con una separación clara entre dos responsabilidades: el backend y el frontend de Keycloak.

Figura 49

Estructura del keycloak-manager

```
keycloak-manager/
├── .gitlab-ci.yml
├── docker-compose.yaml
├── Dockerfile
├── README.md
├── extensions/
│   ├── pom.xml
│   ├── authentication/
│   ├── email-sender/
│   └── user-profile/
└── themes/
    └── tema-uis/
```

La carpeta extensions/ contiene bloques de código Java que extienden las funcionalidades del núcleo de Keycloak. A través de este mecanismo oficial de extensibilidad, es posible agregar comportamientos personalizados en áreas como autenticación, envío de correos electrónicos y gestión de perfiles de usuario, sin modificar el código fuente original del proyecto. Cada extensión vive en su propio módulo dentro de un proyecto Maven unificado, lo que permite compilarlas y empaquetarlas de forma independiente o conjunta según se requiera.

La carpeta themes/ contiene las plantillas visuales del sistema de autenticación. Estas plantillas utilizan el formato FTL, un lenguaje que combina HTML con lógica de presentación que Keycloak resuelve en tiempo de renderizado. A través de estas plantillas se personaliza completamente la interfaz que los usuarios de la UIS verán al autenticarse: la pantalla de inicio

de sesión, el formulario de registro, los mensajes de verificación y la entrega de códigos OTP. Esto permite que la experiencia de autenticación refleje la identidad visual institucional sin depender de las pantallas genéricas que Keycloak ofrece por defecto.

Cada proyecto del laboratorio debe contar con un Dockerfile como pieza obligatoria de su estructura, ya que es el artefacto que define cómo se construye la imagen que será desplegada en cada ambiente. En el caso de keycloak-manager, este archivo tiene una responsabilidad particular: no construir una aplicación desde cero, sino ensamblar una imagen que combine el núcleo oficial de Keycloak con todos los desarrollos propios del laboratorio.

Figura 50

Dockerfile de Keycloak

```
FROM quay.io/keycloak/keycloak:24.0.0
COPY ./themes/ /opt/keycloak/themes/
COPY ./extensions/email-sender/target/*.jar /opt/keycloak/providers/
COPY ./extensions/user-profile/target/*.jar /opt/keycloak/providers/
COPY ./extensions/authentication/target/*.jar
/opt/keycloak/providers/

ENV KC_DB=postgres
RUN /opt/keycloak/bin/kc.sh build
ENV KC_SPI_AUTHENTICATION_PROVIDER=otp-email-verification

# levantar el contenedor
CMD ["start", "--optimized"]
```

La imagen base utilizada es keycloak:24.0.0, una versión estable y específicamente fijada. Esta decisión no es trivial: anclar la imagen a una versión concreta garantiza que los ambientes de desarrollo, staging y producción sean idénticos entre sí y que una actualización inesperada del proveedor no rompa el comportamiento del sistema. Sobre esta base oficial, el

Dockerfile copia los archivos compilados de las extensiones Java hacia el directorio interno donde Keycloak detecta automáticamente los providers disponibles, y de igual forma transfiere las plantillas de temas al directorio correspondiente de themes.

Este mecanismo de copia es el puente entre el desarrollo propio y el núcleo de Keycloak. Al iniciar el contenedor, Keycloak escanea esas rutas y registra automáticamente tanto las extensiones como los temas encontrados, haciéndolos disponibles para su configuración desde la consola de administración. El resultado es una imagen completamente autocontenida: cualquier ambiente que la ejecute tendrá disponibles las extensiones de autenticación, el tema visual institucional y toda la configuración personalizada, sin pasos manuales adicionales ni dependencias externas al contenedor.

Entender cómo se extiende Keycloak es comprender cómo está construido un software de nivel empresarial. Keycloak no es una caja negra: está diseñado deliberadamente para ser extendido mediante un sistema de providers e interfaces bien definidas, siguiendo principios de ingeniería de software que permiten sustituir comportamientos internos sin tocar su código fuente. Aprovechar esta arquitectura correctamente es lo que distingue una integración sólida de un parche frágil.

Dado que el laboratorio requiere múltiples extensiones independientes, la carpeta extensions/ se organiza como un proyecto Maven multi-módulo. Existe un archivo pom.xml padre en la raíz que actúa como coordinador central: en él se declara la versión de Java a utilizar, concretamente Java 17, el groupId del proyecto bajo el espacio de nombres com.eisi.keycloak, y la lista de todos los módulos hijos que componen el conjunto de extensiones: email-sender, user-profile y authentication. Con un único comando de compilación ejecutado desde la raíz,

Maven construye todas las extensiones en el orden correcto, resolviendo dependencias compartidas una sola vez.

Un aspecto técnico crítico del pom.xml padre es la sección dependencyManagement, donde se declaran las bibliotecas de Keycloak con scope provided. Este scope no es un detalle menor: le indica a Maven que esas bibliotecas, como keycloak-services, keycloak-core, keycloak-server-spi y keycloak-model-jpa, no deben empaquetarse dentro del JAR de la extensión porque ya existen en el contenedor de Keycloak en tiempo de ejecución. Incluirlas en el JAR sería redundante y podría generar conflictos de versiones al momento de cargar la extensión. De esta forma, los módulos hijos heredan esta configuración y solo declaran qué dependencias necesitan, sin preocuparse por sus versiones ni por cómo se empaquetan.

Figura 51

Archivo pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.eisi.keycloak</groupId>
    <artifactId>extensions-parent</artifactId>
    <version>1.0.0</version>
    <packaging>pom</packaging> <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <keycloak.version>24.0.0</keycloak.version>
    </properties>

    <modules>
        <module>email-sender</module>
```

```
        <module>user-profile</module>
        <module>authentication</module>
    </modules>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.keycloak</groupId>
                <artifactId>keycloak-services</artifactId>
                <version>${keycloak.version}</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>org.keycloak</groupId>
                <artifactId>keycloak-core</artifactId>
                <version>${keycloak.version}</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>org.keycloak</groupId>
                <artifactId>keycloak-server-spi</artifactId>
                <version>${keycloak.version}</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>org.keycloak</groupId>
                <artifactId>keycloak-server-spi-private</artifactId>
                <version>${keycloak.version}</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>org.keycloak</groupId>
                <artifactId>keycloak-model-jpa</artifactId>
                <version>${keycloak.version}</version>
                <scope>provided</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

Figura 52

Archivo /email-sender/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>com.eisi.keycloak</groupId>
        <artifactId>extensions-parent</artifactId>
        <version>1.0.0</version>
    </parent>

    <artifactId>email-sender</artifactId>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-server-spi</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-server-spi-private</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-services</artifactId>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <resources>
            <resource>
```

```
            <directory>src/main/resources</directory>
        </resource>
    </resources>
</build>

</project>
```

Para desarrollar una extensión es necesario comprender el modelo de providers sobre el cual Keycloak construye toda su arquitectura interna. Este modelo está fundamentado en el quinto principio SOLID, inversión de dependencias, que establece que los módulos de alto nivel no deben depender de implementaciones concretas sino de abstracciones. En la práctica, Keycloak define interfaces para cada una de sus capacidades internas y trabaja exclusivamente contra esas interfaces, lo que significa que para reemplazar o extender un comportamiento el desarrollador provee una nueva implementación de la interfaz correspondiente y Keycloak la detecta y utiliza automáticamente. No se modifica una sola línea del código fuente original.

El módulo email-sender ilustra este proceso de principio a fin. El punto de partida es la interfaz EmailSenderProvider, que Keycloak define con dos métodos: uno que recibe un objeto UserModel y extrae su correo internamente, y otro más genérico que recibe directamente una dirección de correo. Estos dos métodos representan el contrato que cualquier mecanismo de envío debe cumplir.

Figura 53

Archivo EmailSenderProvider.Java

```
public interface EmailSenderProvider extends Provider {
    default void send(Map<String, String> config, UserModel user,
String subject, String textBody, String htmlBody) throws
EmailException {
        send(config, user.getEmail(), subject, textBody, htmlBody);
    }
    void send(Map<String, String> config, String address, String
subject, String textBody, String htmlBody) throws EmailException;
}
```

En lugar de usar la implementación por defecto basada en SMTP, el laboratorio requiere enviar correos a través de un servicio REST externo, lo que proporciona mayor control, trazabilidad y flexibilidad sobre las comunicaciones del sistema de autenticación. Para esto se crea la clase concreta RestEmailSenderProvider que implementa ambos métodos de la interfaz. Internamente, ambos métodos delegan en un método privado executeSend que construye el payload JSON con el destinatario, asunto y cuerpo del mensaje, y lo envía mediante un HttpClient de Java con un timeout de conexión de 15 segundos. Si el servicio externo responde con un código de error, la implementación lanza una EmailException que Keycloak intercepta y maneja según su propia lógica de reintentos y notificaciones. La URL del servicio externo se obtiene de la variable de entorno “Email Rest URL”, manteniéndola fuera del código y alineada con el modelo de inyección de secretos en tiempo de ejecución que usa el laboratorio.

Figura 54

Archivo RestEmailSenderProvider.java

```
public class RestEmailSenderProvider implements EmailSenderProvider {

    private static final HttpClient httpClient =
HttpClient.newBuilder()
            .connectTimeout(Duration.ofSeconds(15))
            .build();
```

```
    private final String apiUrl = System.getenv("EMAIL_REST_URL");

    @Override
    public void send(Map<String, String> config, UserModel user,
String subject, String textBody, String htmlBody) throws
EmailException {
        executeSend(user.getEmail(), subject, textBody, htmlBody);
    }

    @Override
    public void send(Map<String, String> config, String address,
String subject, String textBody, String htmlBody) throws
EmailException {
        executeSend(address, subject, textBody, htmlBody);
    }

    private void executeSend(String emailTo, String subject, String
textBody, String htmlBody) throws EmailException {
        try {
            String content = (htmlBody != null &&
!htmlBody.isEmpty()) ? htmlBody : textBody;
            String jsonBody = buildJson(emailTo, subject, content);

            HttpRequest request = HttpRequest.newBuilder()
                    .uri(URI.create(apiUrl))
                    .header("Content-Type", "application/json")

.POST(HttpRequest.BodyPublishers.ofString(jsonBody))
                    .build();

            HttpResponse<String> response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());

            if (response.statusCode() >= 300) {
                throw new EmailException("Error API externa: " +
response.statusCode() + " - " + response.body());
            }
        } catch (Exception e) {
```

```
            throw new EmailException("Fallo al enviar email via REST
a: " + emailTo, e);
        }
    }

    private String buildJson(String email, String subject, String
body) {
        return "{" +
                "\"email\": \"" + escapeJson(email) + "\"," +
                "\"subject\": \"" + escapeJson(subject) + "\"," +
                "\"body\": \"" + escapeJson(body) + "\"" +
                "}";
    }

    private String escapeJson(String raw) {
        if (raw == null) return "";
        return raw.replace("\\", "\\\\")
                  .replace("\"", "\\\"")
                  .replace("\b", "\\b")
                  .replace("\f", "\\f")
                  .replace("\n", "\\n")
                  .replace("\r", "\\r")
                  .replace("\t", "\\t");
    }

    @Override
    public void close() {
    }


}
```

Tener la implementación concreta no es suficiente: Keycloak necesita saber que existe y cómo instanciarla. Para esto utiliza un sistema de factories donde cada componente debe acompañarse de una clase que implemente la interfaz ProviderFactory correspondiente. La clase RestEmailSenderProviderFactory cumple este rol: implementa EmailSenderProviderFactory y en su método create retorna una nueva instancia de RestEmailSenderProvider cada vez que

Keycloak necesita el componente. El método getId retorna el identificador único rest-email, que es el nombre con el que esta implementación quedará registrada y seleccionable dentro de la consola de administración de Keycloak. Los métodos init y postInit permiten ejecutar lógica de inicialización cuando el servidor arranca, útiles en extensiones que requieren configuración previa o conexiones a recursos externos.

Figura 55

Implementación de RestEmailSenderProviderFactory

```
public class RestEmailSenderProviderFactory implements
EmailSenderProviderFactory {

    @Override
    public EmailSenderProvider create(KeycloakSession session) {
        return new RestEmailSenderProvider();
    }

    @Override
    public void init(Config.Scope config) {
    }

    @Override
    public void postInit(KeycloakSessionFactory factory) {
    }

    @Override
    public void close() {
    }

    @Override
    public String getId() {
        return "rest-email";
    }
}
```

Para que Keycloak pueda descubrir la factory en tiempo de arranque, es necesario un último paso que utiliza el mecanismo estándar de descubrimiento de servicios de Java, conocido como Java SPI. Dentro del módulo se crea el archivo en la ruta 'src/main/resources/META-INF/services/org.keycloak.email.EmailSenderProviderFactory'. El nombre de este archivo no es arbitrario ni configurable: debe coincidir exactamente con el nombre completo del paquete e interfaz que se está implementando, porque es precisamente así como Java localiza en tiempo de ejecución todas las implementaciones disponibles para una interfaz dada. Dentro de ese archivo se coloca una única línea con la ruta completa de la factory del laboratorio: com.eisi.keycloak.email.RestEmailSenderProviderFactory. Con esto, al iniciar el contenedor Keycloak escanea ese directorio del JAR, encuentra la declaración, carga la factory y registra la extensión como un provider disponible.

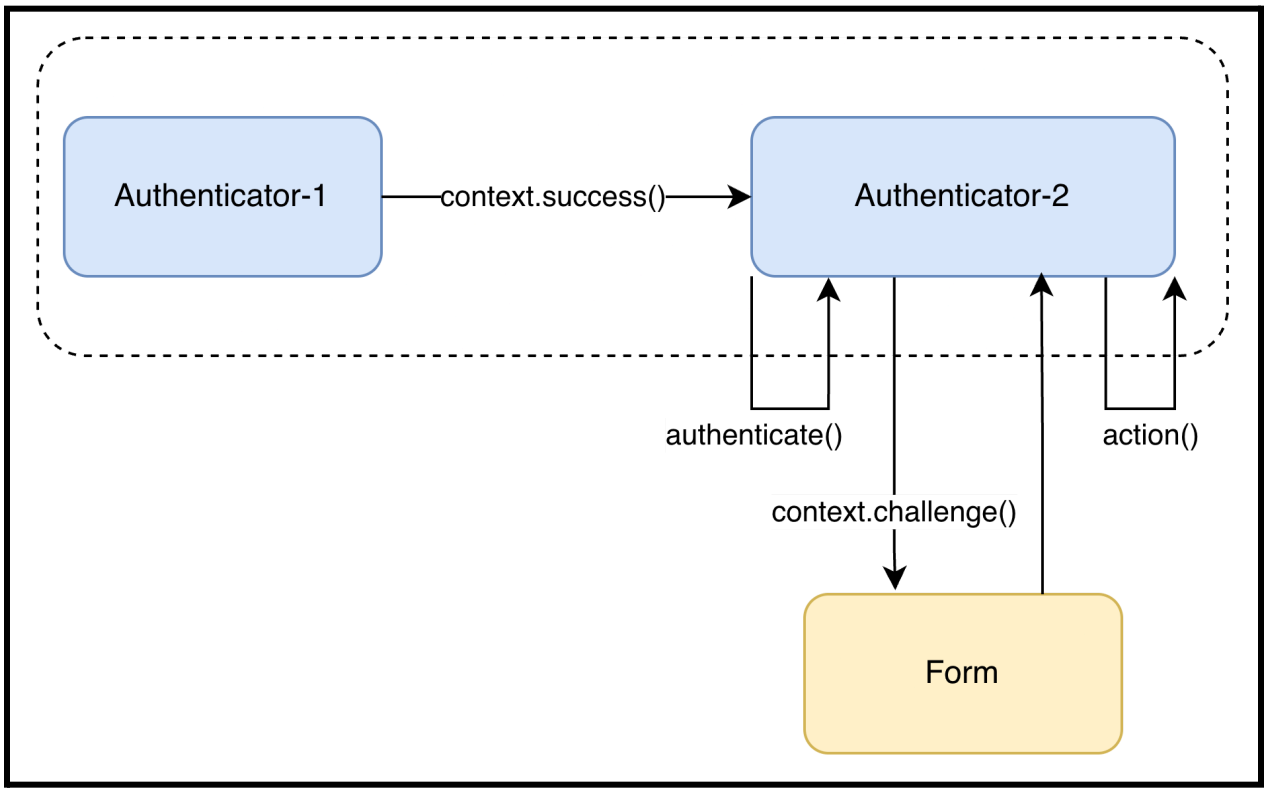
Este mecanismo, combinado con la estructura multi-módulo y el Dockerfile que copia los JARs compilados dentro de la imagen oficial, forma el ciclo completo de extensibilidad: se desarrolla sobre interfaces estables del núcleo, se empaqueta respetando las convenciones del ecosistema Java, y se despliega sin modificar una sola línea de Keycloak. Cualquier nueva extensión que el laboratorio necesite en el futuro sigue exactamente el mismo patrón: una interfaz de Keycloak como contrato, una implementación concreta con la lógica requerida, una factory que la registra, y un archivo de metainformación que la hace descubrible.

Una de las extensiones más relevantes del proyecto es la verificación de identidad mediante un código numérico enviado al correo electrónico del usuario. Aunque Keycloak ofrece nativamente un mecanismo de verificación por correo basado en magic link, donde el usuario recibe un enlace y simplemente hace clic para autenticarse, este enfoque resultaba incompatible con las políticas del correo institucional de la UIS, que bloquea o filtra este tipo de mensajes. La

solución fue desarrollar una extensión propia que envía un código numérico de seis dígitos, más tolerante a los filtros de seguridad del correo universitario y alineado con la experiencia de verificación en dos pasos que los usuarios ya conocen.

Figura 56

Autenticación en Keycloak



Para entender esta extensión es necesario comprender cómo Keycloak organiza sus flujos de autenticación, porque es un conocimiento que no es intuitivo y que requiere lectura profunda de su documentación. Keycloak implementa el patrón de diseño Chain of Responsibility para gestionar la autenticación: un flujo es una cadena de pasos secuenciales, donde cada paso es un Authenticator independiente. Cada Authenticator recibe un objeto context que representa el

estado completo de la sesión de autenticación en ese momento, y a través de ese objeto decide qué ocurre a continuación.

El ciclo de vida de un Authenticator funciona así: primero se invoca el método authenticate(), que evalúa las condiciones iniciales y decide si debe presentar al usuario un desafío, que es un formulario FTL personalizable. Una vez que el usuario completa este formulario y hace submit, Keycloak invoca el método action(), donde se validan los datos ingresados. Si la validación falla, se llama a context.failureChallenge() para devolver el formulario con el mensaje de error correspondiente. Si todo es correcto, se llama a context.success() y la cadena avanza al siguiente Authenticator. Esta separación entre el momento de presentar el desafío y el momento de procesar la respuesta del usuario es lo que permite construir flujos de autenticación complejos de forma modular y controlada.

Dado que la lógica de generación y validación del OTP es necesaria en múltiples puntos del sistema, se centralizó en una clase de utilidad llamada OtpEmailCommonUtil. Esta clase es de tipo estático y no instanciable, un patrón de diseño que comunica explícitamente que no representa un objeto con estado sino un conjunto de operaciones reutilizables. El constructor privado lanza una excepción si alguien intentara instanciarla por reflexión, cerrando completamente esa posibilidad.

La generación del código utiliza SecureRandom, que a diferencia de Random está diseñado específicamente para contextos de seguridad, produciendo valores impredecibles que no pueden ser anticipados por un atacante. El código resultante es un número de seis dígitos generado en el rango entre 100.000 y 999.999, garantizando siempre seis dígitos. Una vez generado, el código y su tiempo de expiración se almacenan en la AuthenticationSession, que es

el espacio de memoria temporal que Keycloak mantiene durante el proceso de autenticación activo. Este almacenamiento es importante porque es completamente efímero: desaparece cuando la sesión termina, ya sea por autenticación exitosa, por expiración o por abandono, sin dejar rastros en base de datos ni en logs.

La expiración está configurada en 300 segundos, cinco minutos, y existe un cooldown de 60 segundos entre reenvíos para evitar que un usuario o un atacante genere solicitudes masivas de códigos. La validación verifica tres condiciones en orden: que el código ingresado no esté vacío, que no haya expirado comparando el tiempo actual con el tiempo de expiración almacenado, y que coincida exactamente con el código guardado en sesión. Si cualquiera de estas condiciones falla, retorna el mensaje de error correspondiente; si todas pasan, retorna null como señal de validación exitosa.

El envío del correo se delega en el EmailTemplateProvider de Keycloak, que es precisamente la interfaz que la extensión RestEmailSenderProvider reemplaza. Esto significa que toda la cadena está conectada: la utilidad común pide a Keycloak que envíe un correo, Keycloak busca su provider de correo registrado, y encuentra la implementación REST del laboratorio en lugar del SMTP por defecto. La plantilla usada es email-template-verification-otp.ftl, que recibe el código y el tiempo de expiración en minutos como atributos para renderizarlos en el mensaje.

La clase ResetPasswordOtpAuthenticator implementa la interfaz Authenticator de Keycloak y representa el paso de verificación OTP dentro de un flujo de autenticación configurable. Cuando Keycloak llega a este paso, invoca authenticate(), que inmediatamente genera y envía el OTP al correo del usuario mediante la utilidad común, y luego presenta el formulario email-otp-verification.ftl como desafío. Si el envío falla por un error en el servicio de

correo, el flujo interrumpe el desafío normal y presenta una página de error, evitando que el usuario quede en un estado indefinido esperando un código que nunca llegará.

Cuando el usuario envía el formulario, Keycloak invoca action(). Este método primero revisa si la acción enviada es un reenvío, en cuyo caso genera un nuevo código y vuelve a presentar el formulario con un mensaje informativo. Si es una validación normal, delega en OtpEmailCommonUtil.validateOtp() y según el resultado vuelve a presentar el formulario con el error o llama a context.success() para avanzar en la cadena. Como efecto adicional, si el usuario completa la verificación exitosamente y su correo no estaba marcado como verificado en Keycloak, este paso lo marca automáticamente, unificando la verificación de autenticación con la verificación de cuenta.

Junto al authenticator existe una segunda extensión complementaria: VerifyOtpEmail, que implementa RequiredActionProvider. En Keycloak, una Required Action es un componente diferente a un Authenticator: no vive dentro del flujo de autenticación sino que se ejecuta después de que el usuario se autentica correctamente pero antes de que se le conceda acceso a la aplicación. Es una acción pendiente que el usuario debe completar obligatoriamente.

El método evaluateTriggers() se ejecuta en cada autenticación para determinar si esta acción debe activarse. En este caso, si el realm tiene configurada la verificación de correo y el usuario aún no tiene su email verificado, la acción se agrega automáticamente a su lista de pendientes. El método requiredActionChallenge() presenta el mismo formulario de ingreso de OTP, pero antes verifica si ya existe un código en la sesión para no generar uno nuevo innecesariamente en caso de que la página se recargue. El procesamiento de la respuesta en

processAction() sigue la misma lógica que el authenticator: reenvío o validación, con el mismo mecanismo de éxito que marca el correo como verificado.

Esta separación entre ResetPasswordOtpAuthenticator y VerifyOtpEmail responde a dos escenarios distintos dentro del ciclo de autenticación, pero ambos comparten exactamente la misma lógica de generación, envío y validación a través de OtpEmailCommonUtil. Esta decisión de centralizar la lógica común evita duplicación de código y garantiza que cualquier cambio en la política de OTP, como modificar el tiempo de expiración o el formato del código, se aplique de forma consistente en todos los puntos del sistema donde se utiliza.

La capa visual del sistema de autenticación es tan importante como su lógica de backend. Keycloak permite reemplazar completamente sus pantallas predeterminadas mediante un sistema de plantillas basado en FreeMarker Template Language, conocido como FTL. Este lenguaje de plantillas es interpretado: el archivo no es HTML estático sino una combinación de HTML con directivas especiales que el motor de Keycloak procesa en el servidor antes de enviar la respuesta al navegador. Estas directivas, reconocibles por el prefijo #, permiten evaluar condiciones, iterar listas, acceder a variables del contexto de autenticación y componer estructuras reutilizables, todo en tiempo de renderizado sin que el usuario final lo perciba.

La pieza central del sistema de plantillas es template.ftl, que define el layout base mediante una macro reutilizable llamada registrationLayout. En FTL, una macro es un bloque de código que puede ser invocado desde otras plantillas, similar a un componente. Todas las pantallas del sistema, como el login, el registro y la verificación OTP, importan este template y lo utilizan como envoltorio, garantizando que el encabezado HTML, las hojas de estilo, los scripts y la estructura general sean consistentes en toda la experiencia de autenticación.

Para el estilo visual se adoptó Beer CSS, una biblioteca de componentes ligera basada en Material Design 3. Esta elección no fue arbitraria: la Universidad Industrial de Santander utiliza Material Design como su sistema de diseño oficial, por lo que usar Beer CSS permite que las pantallas de autenticación sean coherentes con la identidad visual institucional sin necesidad de construir un sistema de estilos desde cero. Además, su peso reducido es especialmente relevante en pantallas de autenticación, que deben cargar rápidamente incluso en conexiones lentas.

El template.ftl también centraliza la lógica de notificaciones globales mediante un componente snackbar. Cuando Keycloak necesita mostrar un mensaje al usuario, ya sea un error de credenciales, una confirmación de envío de correo o cualquier notificación informativa, el template detecta el tipo de mensaje y aplica la clase CSS correspondiente, mostrando el snackbar automáticamente durante cinco segundos. Esta lógica vive en un único lugar y se aplica a todas las pantallas sin que cada plantilla individual tenga que implementarla.

Figura 57

Archivo template.ftl

```
<#macro registrationLayout bodyClass="" displayInfo=false
displayMessage=true displayRequiredFields=false
showAnotherWayIfPresent=true>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
    <meta name="robots" content="noindex, nofollow">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0, maximum-scale=1.0, user-scalable=no">

    <link
```

```
href="https://cdn.jsdelivr.net/npm/beercss@3.13.1/dist/cdn/beer.min.c
ss" rel="stylesheet">
    <script type="module"
src="https://cdn.jsdelivr.net/npm/beercss@3.13.1/dist/cdn/beer.min.js
"></script>
    <script type="module"
src="https://cdn.jsdelivr.net/npm/material-dynamic-colors@1.1.2/dist/
cdn/material-dynamic-colors.min.js"></script>

    <link rel="icon"
href="${url.resourcesPath}/img/favicon.ico?v=999" />

    <title>${msg("loginTitle", "UIS - Inicio de Sesión")}</title>

    <#if properties.styles?has_content>
        <#list properties.styles?split(' ') as style>
            <link href="${url.resourcesPath}/${style}"
rel="stylesheet" />
        </#list>
    </#if>

    <#if properties.scripts?has_content>
        <#list properties.scripts?split(' ') as script>
            <script src="${url.resourcesPath}/${script}"
type="text/javascript"></script>
        </#list>
    </#if>
</head>

<body>
    <main class="principal-container">
        <#nested "form">
    </main>

    <#if displayMessage && message?has_content && (message.type !=
'warning')>

        <#assign messageClass = "">
        <#if message.type = 'error'>
```

```
            <#assign messageClass = "error">
        <#elseif message.type = 'success'>
            <#assign messageClass = "success">
        <#else>
            <#assign messageClass = "primary">
        </#if>

        <div class="snackbar ${messageClass}" id="global-snackbar">
            <#if message.type = 'error'><i
class="front">error</i></#if>
            <#if message.type = 'success'><i
class="front">check_circle</i></#if>
            <span>${message.summary?no_esc}</span>
        </div>

        <script>
            window.addEventListener("load", function() {
                ui("#global-snackbar", 5000); /            });
        </script>
    </#if>
</body>
</html>
</#macro>
```

El formulario de registro, definido en register.ftl, ilustra cómo las plantillas FTL pueden ir mucho más allá de HTML estático. En lugar de presentar un único formulario largo, la pantalla está dividida en cuatro pasos secuenciales: información de nombres, datos institucionales, información personal y contraseña. Cada paso es una sección del mismo formulario HTML con visibilidad controlada por JavaScript, lo que significa que todos los datos viajan en un único submit al servidor sin necesidad de múltiples peticiones.

Un aspecto técnico relevante es el manejo de nombres y apellidos. El formulario divide visualmente el primer nombre, segundo nombre, primer apellido y segundo apellido en campos independientes para facilitar el llenado, pero Keycloak espera recibir firstName y lastName

como campos únicos. La plantilla resuelve esto con campos ocultos que se construyen dinámicamente combinando los valores temporales antes del envío, y en sentido inverso, cuando hay errores de validación y el formulario se pre-rellena con los valores anteriores, usa las funciones FTL keep_before y keep_after para separar nuevamente los valores por el espacio y distribuirlos en los campos correctos.

La lógica de los programas académicos muestra otra capacidad importante de las plantillas FTL: recibir datos dinámicos del backend. El selector de programa académico solo se renderiza si la variable 'programasAcademicos' está disponible y tiene contenido, lo cual se verifica con la directiva ?? de FTL. Si el backend no pudo cargar la lista, el campo simplemente no aparece en lugar de mostrar un selector vacío o roto. Cada opción del selector muestra solo el nombre del programa al usuario, pero envía el valor completo del atributo al servidor, separando la presentación del dato técnico.

Cuando Keycloak procesa el formulario y detecta errores de validación, devuelve la respuesta indicando qué campos fallaron. La plantilla usa messagesPerField para evaluar en qué pasó ocurrió el error y establece el paso inicial de la pantalla en consecuencia, con prioridad hacia los pasos anteriores. Si hay errores en el paso 4 pero también en el paso 2, el formulario se abre en el paso 2 porque es más lógico que el usuario corrija desde el principio. Esta variable se pasa como dato al JavaScript de la página para que controle qué sección mostrar al cargar.

Figura 58

Archivo register.ftl

```
<#import "./template.ftl" as layout>
<@layout.registrationLayout displayInfo=false; section>
```

```
    <#if section = "form">

        <div class="absolute right bottom uis-icon">
            <img src="${url.resourcesPath}/img/logo-uis.png"
alt="Logo EISI" class="extra"/>
        </div>

        <article class="center">

        <!-- Logo y titulo -->
        <div class="row">
            <div>
                <img src="${url.resourcesPath}/img/logo-eisi.png"
alt="Logo EISI" class="logo-eisi"/>
            </div>
            <div class="max">
                <h5 class="bold small">Escuela de Ingeniería de
Sistemas e Informática</h5>
            </div>
        </div>

        <div class="space"></div>

        <form action="${url.loginAction}" method="post">

            <!-- Step 1 - informacion básica -->
            <div class="form-step active" id="step1">

                <!-- Titulo y subtitulo -->
                <h5 class="small bold">Crea una cuenta</h5>
                <span>Ingresa tus nombres y apellidos</span>

                <input type="hidden" id="realFirstName"
name="firstName" value="${(register.formData.firstName!'')}"/>
                <input type="hidden" id="realLastName"
name="lastName" value="${(register.formData.lastName!'')}"/>

                <div class="space"></div>
```

```
                <div class="field label required-field">
                    <input id="temp-firstName"
value="${(register.formData.firstName!'')?keep_before(' ')}"
type="text" autocomplete="given-name"/>
                    <label>Primer nombre*</label>
                    <output class="invisible
invalid">Obligatorio</output>
                </div>

                <div class="field label">
                    <input id="temp-middleName"
value="${(register.formData.firstName!'')?keep_after(' ')}"
type="text" autocomplete="family-name"/>
                    <label>Segundo nombre</label>
                    <output class="invisible
invalid">Obligatorio</output>
                </div>

                <div class="field label required-field">
                    <input id="temp-lastName"
value="${(register.formData.lastName!'')?keep_before(' ')}"
type="text" autocomplete="additional-name"/>
                    <label>Primer apellido*</label>
                    <output class="invisible
invalid">Obligatorio</output>
                </div>

                <div class="field label">
                    <input id="temp-secondLastName"
value="${(register.formData.lastName!'')?keep_after(' ')}"
type="text" autocomplete="additional-name"/>
                    <label>Segundo apellido</label>
                    <output class="invisible
invalid">Obligatorio</output>
                </div>

            </div>

            <!-- Step 2 - informacion básica -->
```

```
            <div class="form-step" id="step2">

                <div class="row step-header">
                    <button type="button" class="transparent circle
left-round small btn-back-arrow">
                        <i>arrow_back</i>
                    </button>
                    <h5 class="small bold">Información
institucional</h5>
                </div>
                <span>Ingresa tu correo institucional y programa
académico</span>

                <div class="space"></div>

                 <div class="space"></div>

                    <div class="field label required-field <#if
messagesPerField.existsError('email')>invalid</#if>">
                        <input id="email" name="email"
value="${(register.formData.email!'')}" type="email"
autocomplete="email"/>
                        <label>Correo institucional*</label>
                        <output class="<#if
!messagesPerField.existsError('email')>invisible</#if> invalid">
                            <#if
!messagesPerField.existsError('email')>
                                Obligatorio
                            <#else>

${messagesPerField.get('email')!'Obligatorio'}
                            </#if>
                        </output>
                    </div>

                    <#if programasAcademicos?? &&
programasAcademicos?has_content>
                        <div class="field responsive label suffix max
required-field">
```

```
                            <select id="academicProgram"
name="user.attributes.academicProgram" autocomplete="off">
                                <option disabled
value="">Selecciona</option>
                                <#list programasAcademicos as item>
                                    <option
value="${item}">${item?split(':')[1]?trim}</option>
                                </#list>
                            </select>
                            <label>Programa Académico*</label>
                            <i>arrow_drop_down</i>
                            <output class="invisible
invalid">Obligatorio</output>
                        </div>
                    </#if>

                    <div class="field label required-field <#if
messagesPerField.existsError('studentCode')>invalid</#if>">
                        <input id="studentCode"
name="user.attributes.studentCode"
value="${(register.formData['user.attributes.studentCode']!'')}"
type="number" autocomplete="off"/>
                        <label>Código estudiantil*</label>
                        <output class="<#if
!messagesPerField.existsError('studentCode')>invisible</#if>
invalid">
                            <#if
!messagesPerField.existsError('studentCode')>
                                Obligatorio
                            <#else>

${messagesPerField.get('studentCode')!'Obligatorio'}
                            </#if>
                        </output>
                    </div>

            </div>

            <!-- Step 3 - informacion personal -->
```

```
            <div class="form-step" id="step3">

                <!-- Titulo y subtitulo -->
                <div class="row step-header">
                    <button type="button" class="transparent circle
left-round small btn-back-arrow">
                        <i>arrow_back</i>
                    </button>
                    <h5 class="small bold">Información personal</h5>
                </div>
                <span>Ingresa tu datos personales</span>

                <div class="space"></div>

                    <div class="field label required-field <#if
messagesPerField.existsError('email')>invalid</#if>">
                        <input id="email"
name="user.attributes.personalEmail"
value="${(register.formData.email!'')}" type="email"
autocomplete="email"/>
                        <label>Correo personal*</label>
                        <output class="<#if
!messagesPerField.existsError('email')>invisible</#if> invalid">
                            <#if
!messagesPerField.existsError('email')>
                                Obligatorio
                            <#else>

${messagesPerField.get('email')!'Obligatorio'}
                            </#if>
                        </output>
                    </div>

                    <div class="field label required-field">
                        <input id="phone"
name="user.attributes.phone"
value="${(register.formData['user.attributes.phone']!'')}"
type="text" autocomplete="tel"/>
                        <label>Teléfono*</label>
```

```
                        <output class="invisible
invalid">Obligatorio</output>
                    </div>

                    <div class="s7 m8 field label max
required-field">
                        <input id="document"
name="user.attributes.documentNumber"
value="${(register.formData['user.attributes.documentNumber']!'')}"
type="number" autocomplete="off"/>
                        <label>Documento*</label>
                        <output class="invisible
invalid">Obligatorio</output>
                    </div>
            </div>

            <!-- Step 4 - Contraseña -->
            <div class="form-step" id="step4">

                <div class="row step-header">
                    <button type="button" class="transparent circle
left-round small btn-back-arrow">
                        <i>arrow_back</i>
                    </button>
                    <h5 class="small bold">Contraseña</h5>
                </div>
                <span>Ingresa tu contraseña</span>

                <div class="space"></div>

                    <div class="field label required-field <#if
messagesPerField.existsError('password')>invalid</#if>">
                        <input id="password" name="password"
value="${(register.formData.password!'')}" type="password"
autocomplete="new-password"/>
                        <label>Contraseña*</label>
                        <output class="<#if
!messagesPerField.existsError('password')>invisible</#if> invalid">
```

```
${messagesPerField.get('password')!'Obligatorio'}
                        </output>
                    </div>

                    <div class="field label required-field <#if
messagesPerField.existsError('password-confirm')>invalid</#if>">
                        <input id="password-confirm"
name="password-confirm"
value="${(register.formData.passwordConfirm!'')}" type="password"
autocomplete="new-password"/>
                        <label>Confirmar contraseña*</label>
                        <output class="<#if
!messagesPerField.existsError('password-confirm')>invisible</#if>
invalid">

${messagesPerField.get('password-confirm')!'Obligatorio'}
                        </output>
                    </div>

            </div>


            <div class="space"></div>

            <!-- Botones -->
             <div class="row center-align large-space mt-2">

                <div class="max">
                    <button type="button"
onclick="window.location.href='${url.loginUrl}'" class="border
primary-border primary-text responsive">Cancelar</button>
                </div>

                <div class="max">
                    <button id="nextBtn" class="primary responsive"
type="button">
                        <span id="nextBtnText">Siguiente</span>
                        <progress class="circle small white-text
indeterminate d-none"></progress>
```

```
                    </button>
                </div>

            </div>

        </form>

        </article>

        <#assign initialStep = 1>

             <!-- Verificamos errores de atrás hacia adelante o por
prioridad -->

            <!-- Paso 4: Contraseñas -->
            <#if messagesPerField.existsError('password',
'password-confirm')>
                <#assign initialStep = 4>
            </#if>

            <#-- Paso 3: Contacto y Documento (Sobrescribe si hay
error aquí, pues es anterior) -->
            <#if
messagesPerField.existsError('user.attributes.phone',
'user.attributes.documentNumber', 'user.attributes.personalEmail')>
                <#assign initialStep = 3>
            </#if>

            <#-- Paso 2: Info Básica -->
            <#if messagesPerField.existsError('email',
'user.attributes.studentCode', 'user.attributes.academicProgram')>
                <#assign initialStep = 2>
            </#if>

            <#-- Paso 1: Nombres (Prioridad máxima: si falla esto,
volvemos al inicio) -->
            <#if messagesPerField.existsError('firstName',
'lastName')>
                <#assign initialStep = 1>
```

```
            </#if>

        <script>
            window.serverInitialStep = ${initialStep};
        </script>
    </#if>
</@layout.registrationLayout>
```

Con las extensiones compiladas y las plantillas construidas, el último paso es conectar el proyecto con el ecosistema de automatización del laboratorio para que el código llegue a los ambientes de ejecución de forma controlada y repetible. Este proceso se define en el archivo .gitlab-ci.yml del proyecto, que no construye su pipeline desde cero sino que consume las plantillas centrales del repositorio gitlab-ci-templates mediante la directiva include.

La directiva include permite referenciar archivos de configuración que viven en otros repositorios de GitLab, especificando el proyecto, la ruta del archivo y la referencia desde la cual tomarlo. Aunque es posible apuntar a una rama como main o dev, el laboratorio adopta como buena práctica el uso de tags de versionado semántico en esa referencia. Esta decisión garantiza trazabilidad total: si el comportamiento del pipeline cambia en una versión futura de las plantillas, los proyectos que apuntan a un tag específico no se ven afectados hasta que deliberadamente actualicen esa referencia. De esta forma, una mejora en las plantillas centrales no puede romper accidentalmente un proyecto que no esperaba.

Las plantillas incluidas cubren las tres etapas del ciclo: el análisis de calidad con SonarQube, la construcción y publicación de la imagen Docker, y el trigger de despliegue hacia la infraestructura. Para el caso particular de keycloak-manager, la plantilla de construcción

Maven requiere una personalización: dado que el código Java no está en la raíz del repositorio sino dentro de la carpeta extensions/, se sobreescribe la variable “Maven Project Dir” para indicarle a la plantilla dónde encontrar el pom.xml padre. Esta capacidad de sobrescribir variables de las plantillas es lo que las hace verdaderamente reutilizables: el comportamiento base es compartido pero cada proyecto puede adaptarlo a su propia estructura sin modificar la plantilla original.

El bloque trigger_deploy es el más significativo en términos de configuración, porque es donde se define cómo debe ejecutarse el contenedor en el servidor de destino. La variable “App Networks”especifica a qué redes Docker estará conectado el contenedor, lo que determina con qué otros servicios del ecosistema podrá comunicarse. La variable “Secret Name” define el nombre del secreto dentro de la bóveda de HashiCorp Vault, siguiendo la convención de rutas del laboratorio que sigue el formato “Grupo/Ambiente/Nombre Secreto”. Esta convención permite que un mismo grupo de servicios tenga secretos independientes por ambiente, de forma que las credenciales de desarrollo nunca se mezclen con las de producción.

La variable “App Vars” es especialmente crítica: contiene los nombres de las claves que el motor de despliegue debe buscar dentro del secreto descargado de Vault y establecer como variables de entorno dentro del contenedor. Si alguna de las claves declaradas en “App Vars” no existe en el secreto correspondiente de la bóveda, el pipeline falla en su etapa de Plan antes de tocar cualquier contenedor en producción. Este comportamiento no es un defecto sino una garantía deliberada: es preferible que el pipeline se detenga con un error claro durante la validación a que el contenedor arranque con variables faltantes y falle silenciosamente en tiempo de ejecución. Una vez que la etapa de Plan confirma que todos los secretos están disponibles, los

descarga y los inyecta directamente como variables de entorno dentro del contenedor, sin que sus valores aparezcan en ningún log del pipeline ni en ningún archivo de configuración intermedio.

Figura 59

Archivo .gitlab-ci.yml

```yaml
image: docker:27.3.1
services:
  - docker:dind

stages:
  - build
  - push
  - deploy

# ========================================================
# INCLUSIÓN DE TEMPLATES (Librería de Infra)
# ========================================================
include:
  - project: "infraestructure/gitlab-ci-templates"
    file: "/build/maven-build.yml"
    ref: v4.0.5

  - project: "infraestructure/gitlab-ci-templates"
    file: "/security/sonar-scanner-extension.yml"
    ref: v4.0.5

  - project: "infraestructure/gitlab-ci-templates"
    file: "/docker/docker-push.yml"
    ref: v4.0.5

  - project: "infraestructure/gitlab-ci-templates"
    file: "/deploy/trigger-infra.yml"
    ref: v4.0.5

# ========================================================
# CONFIGURACIÓN ESPECÍFICA DEL PROYECTO
# ========================================================
```

```
maven_build:
  variables:
    MAVEN_PROJECT_DIR: "extensions"

trigger_deploy:
  extends: .trigger-infra
  variables:
    APP_NETWORKS: "eisi-core-net nginx-network"
    SECRET_NAME: "KEYCLOAK-SECRET"
    APP_VARS: |
      - KC_DB
      - KC_DB_URL
      - KC_DB_USERNAME
      - KC_DB_PASSWORD
      - KEYCLOAK_ADMIN
      - KEYCLOAK_ADMIN_PASSWORD
      - KC_HOSTNAME_STRICT
      - KC_PROXY
      - KC_PROXY_HEADERS
      - KC_HTTP_ENABLED
      - KC_HOSTNAME
      - EMAIL_REST_KEY
      - EMAIL_REST_URL
```

Figura 60

Visualización de secretos del Vault

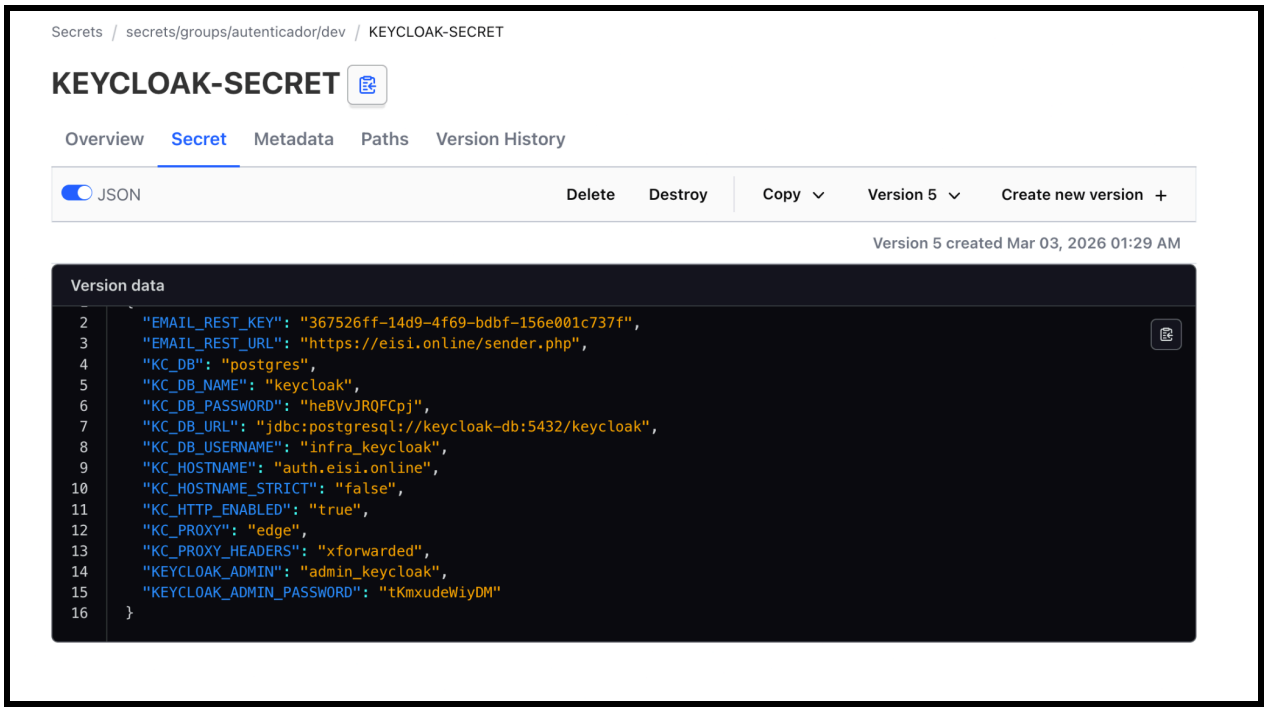


Figura 61

Verificación de los pipelines

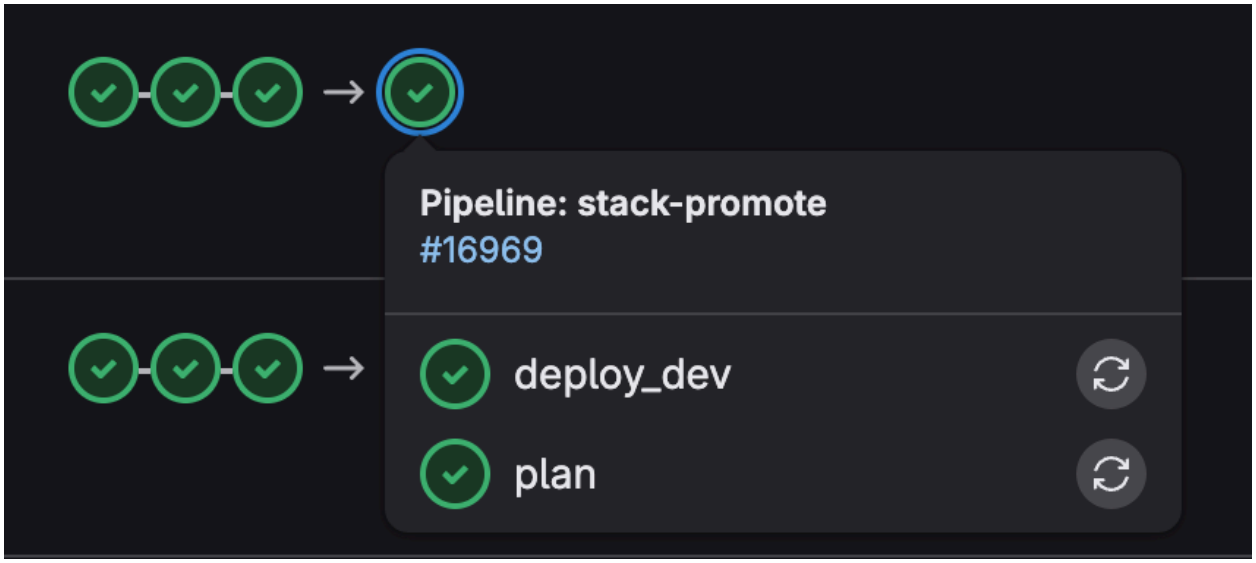


Figura 62

Estructura del api-gateway-nginx

```
├── .gitlab-ci.yml
├── Dockerfile
├── entrypoint.sh
│
├── projects/
│   ├── equipo-saam/
│   │   └── saam.yaml
│   ├── equipo-autenticador/
│   └── ...
│
├── scripts/
│   └── generator.py
│
└── templates/
    └── nginx_block.j2
```

Todo sistema de autenticación centralizado necesita un punto de entrada igualmente centralizado: un componente que reciba todas las peticiones entrantes, verifique que el usuario tiene una sesión válida y las enrute hacia el servicio correcto. Este es el rol del API Gateway del laboratorio, construido sobre OpenResty, una distribución de Nginx que incorpora un motor Lua embebido. Esta elección no es arbitraria: Nginx por sí solo es un proxy inverso extraordinariamente eficiente, pero no puede ejecutar lógica de autenticación compleja. OpenResty extiende Nginx con la capacidad de ejecutar scripts Lua en cada fase del ciclo de vida de una petición, lo que permite implementar flujos OIDC completos, gestión de sesiones y generación de tokens internos sin necesidad de un servicio auxiliar separado.

Arquitectura de configuración dinámica: El desafío central del gateway es que debe servir a múltiples proyectos del laboratorio, cada uno con su propio dominio, sus propios servicios y sus propias reglas de seguridad, sin que agregar un nuevo proyecto requiera modificar manualmente los archivos de configuración de Nginx. Para resolver esto, el gateway adopta un

modelo de configuración declarativa: cada proyecto describe sus necesidades en un archivo YAML, y un script de Python llamado generator.py transforma esas declaraciones en bloques de configuración Nginx listos para usar antes de que el servidor arranque.

Cada archivo YAML de proyecto declara su grupo de pertenencia, los dominios por ambiente, si el proyecto requiere autenticación, las redes a las que pertenece, las reglas de control de acceso por IP y la lista de servicios que lo componen, distinguiendo entre frontends y backends. El generador recorre todos los archivos YAML en el directorio de proyectos, valida su contenido, y para cada proyecto que tenga un dominio definido en el ambiente activo genera un archivo .conf de Nginx usando una plantilla Jinja2 llamada nginx_block.j2. Si durante la validación se detecta algún error, como un proyecto marcado como protegido al que le faltan variables de seguridad críticas, el generador termina con error y el gateway no arranca, evitando un despliegue parcialmente configurado.

La imagen base es openresty/openresty:alpine-fat, una variante que incluye LuaRocks, el gestor de paquetes para Lua. Sobre esta base el Dockerfile instala las bibliotecas que implementan toda la inteligencia del gateway: lua-resty-openidc gestiona el flujo de autenticación con Keycloak siguiendo el protocolo OpenID Connect, lua-resty-session maneja las sesiones de usuario almacenadas en Redis, lua-resty-jwt permite firmar y verificar tokens JWT internos, y lua-resty-redis provee la conexión al almacén de sesiones. Adicionalmente instala Python con las bibliotecas pyyaml y jinja2 necesarias para ejecutar el generador de configuraciones. El resultado es una imagen completamente autocontenida que no necesita servicios auxiliares de proceso para funcionar.

El archivo entrypoint.sh define la secuencia de arranque del contenedor. Primero limpia cualquier configuración residual de ejecuciones anteriores para garantizar que el estado sea siempre limpio. Luego ejecuta generator.py, que lee los archivos YAML de proyectos y genera los bloques de configuración Nginx dinámicamente para el ambiente activo, determinado por la variable de entorno Target Env. Una vez generadas las configuraciones dinámicas, copia las configuraciones estáticas del directorio custom/, que contienen bloques de Nginx para servicios core que no siguen el modelo declarativo. Finalmente valida la sintaxis de toda la configuración con 'openresty -t' antes de iniciar el servidor. Si la validación falla, el contenedor se detiene con error antes de servir cualquier petición. Esta secuencia garantiza que Nginx nunca arranque con una configuración inválida o incompleta.

La plantilla Jinja2 genera para cada proyecto un bloque de servidor Nginx con toda su lógica de seguridad. Cada dominio recibe dos bloques: uno en el puerto 80 que únicamente redirige a HTTPS y gestiona la renovación de certificados TLS, y uno en el puerto 443 con la configuración completa de SSL y toda la lógica de autenticación.

Para los proyectos marcados como protegidos, la plantilla inyecta un bloque access_by_lua_block que se ejecuta en cada petición antes de que Nginx la enrute al servicio de destino. Este bloque invoca lua-resty-openidc para verificar si el usuario tiene una sesión activa en Redis. Si no la tiene, lo redirige al flujo de autenticación de Keycloak. Si la sesión es válida, extrae la información del usuario del id token y el access token y construye lo que el laboratorio denomina un Phantom Token: un JWT de corta duración, firmado con un secreto interno del gateway, que contiene los datos del usuario ya procesados, sus roles globales y sus roles por cliente. Este token reemplaza completamente la cabecera Authorization original y la cookie de sesión antes de que la petición llegue al backend, de forma que los servicios internos nunca ven

credenciales de Keycloak sino tokens simplificados y de vida corta que el propio gateway controla.

La limpieza de campos técnicos del id token antes de construir el Phantom Token es deliberada: campos como nonce, jti, aud o session state son relevantes para el protocolo OIDC pero no tienen utilidad para un backend de aplicación, y exponerlos innecesariamente ampliará la superficie de información sensible que circula por la red interna.

La plantilla genera también dos endpoints especiales para proyectos protegidos. El endpoint /auth/userinfo permite que los backends consulten la información del usuario autenticado enviando su ‘Client Id’ y ‘Client Secret’ junto con el Phantom Token recibido. El gateway valida las credenciales del backend contra sus variables de entorno, verifica la firma del token y devuelve los datos del usuario en formato JSON. Este mecanismo desacopla completamente a los backends de Keycloak: ningún servicio interno necesita implementar OIDC ni conocer la existencia de Keycloak, solo necesita confiar en el gateway. El endpoint /auth/slo/logout gestiona el cierre de sesión global, destruyendo la sesión local en Redis y redirigiendo al endpoint de logout de Keycloak con el id token como hint para que Keycloak también invalide la sesión en su lado.

El enrutamiento de servicios distingue entre frontends y backends. Para los frontends, los archivos estáticos como JavaScript, CSS e imágenes se sirven con caché desactivada explícitamente para evitar que el usuario reciba versiones obsoletas de la aplicación. Para los backends, cada ruta declarada en el YAML genera un bloque de localización en Nginx que reescribe la URL eliminando el prefijo del API antes de hacer el proxy hacia el upstream

correspondiente, de forma que el backend recibe la petición en su ruta original sin los prefijos de enrutamiento del gateway.

Al final del proceso de generación, generator.py produce un archivo Lua adicional llamado ‘secrets_registry.lua’ que construye un mapa en memoria de todos los identificadores de clientes de proyectos protegidos hacia sus credenciales correspondientes. Este registro permite que la lógica Lua del gateway resuelva en tiempo de ejecución qué secreto corresponde a cada cliente sin necesidad de búsquedas en variables de entorno por nombre de proyecto, centralizando el acceso a credenciales en un único artefacto generado automáticamente a partir de las declaraciones YAML de los proyectos.

Con el gateway configurado y operativo, conectar Keycloak al ecosistema de enrutamiento es tan simple como agregar un archivo de declaración YAML al directorio de proyectos. Esta es precisamente la filosofía del modelo declarativo del gateway: incorporar un nuevo servicio no requiere tocar ningún archivo de Nginx ni reiniciar manualmente nada, solo describir el servicio en el formato esperado y el generador se encarga del resto en el próximo arranque.

La configuración de Keycloak declara el grupo autenticador, que es el mismo equipo registrado en Stack Provisioner al inicio del proyecto, manteniendo consistencia en la nomenclatura a lo largo de todo el ecosistema. Los dominios definen que en el ambiente de desarrollo el servicio estará disponible en auth.eisi.online y en producción en auth.uis.edu.co, mientras que el ambiente de staging no tiene dominio asignado, lo que le indica al generador que simplemente omita este proyecto para ese ambiente sin generar error.

El control de acceso por IP está configurado como all, lo que significa que Keycloak es accesible desde cualquier dirección. Esto es correcto para un sistema de autenticación: la pantalla de login debe ser alcanzable desde cualquier red ya que es el punto de entrada público del sistema. Si estuviera restringido a IPs internas, los usuarios externos nunca podrían autenticarse.

El campo protegido: false es igualmente deliberado y responde a una razón lógica fundamental: no tiene sentido proteger con autenticación el servicio que provee la autenticación misma. Si el gateway intentará verificar la sesión del usuario antes de llegar a Keycloak, los usuarios no autenticados quedarían en un ciclo infinito de redirecciones sin poder iniciar sesión nunca. Al marcarlo como no protegido, el gateway enruta las peticiones directamente a Keycloak sin pasar por el bloque Lua de verificación OIDC.

Finalmente, el servicio se declara de tipo front apuntando al contenedor keycloak-manager en el puerto 8080, que es el puerto en el que Keycloak expone su interfaz HTTP por defecto. Al ser de tipo front, el gateway aplica las cabeceras de control de caché correspondientes y enruta todo el tráfico del dominio hacia ese upstream, incluyendo tanto la consola de administración como los flujos de autenticación y las pantallas personalizadas que construimos con las plantillas FTL.

Figura 63

Configuración de exposición de Keycloak

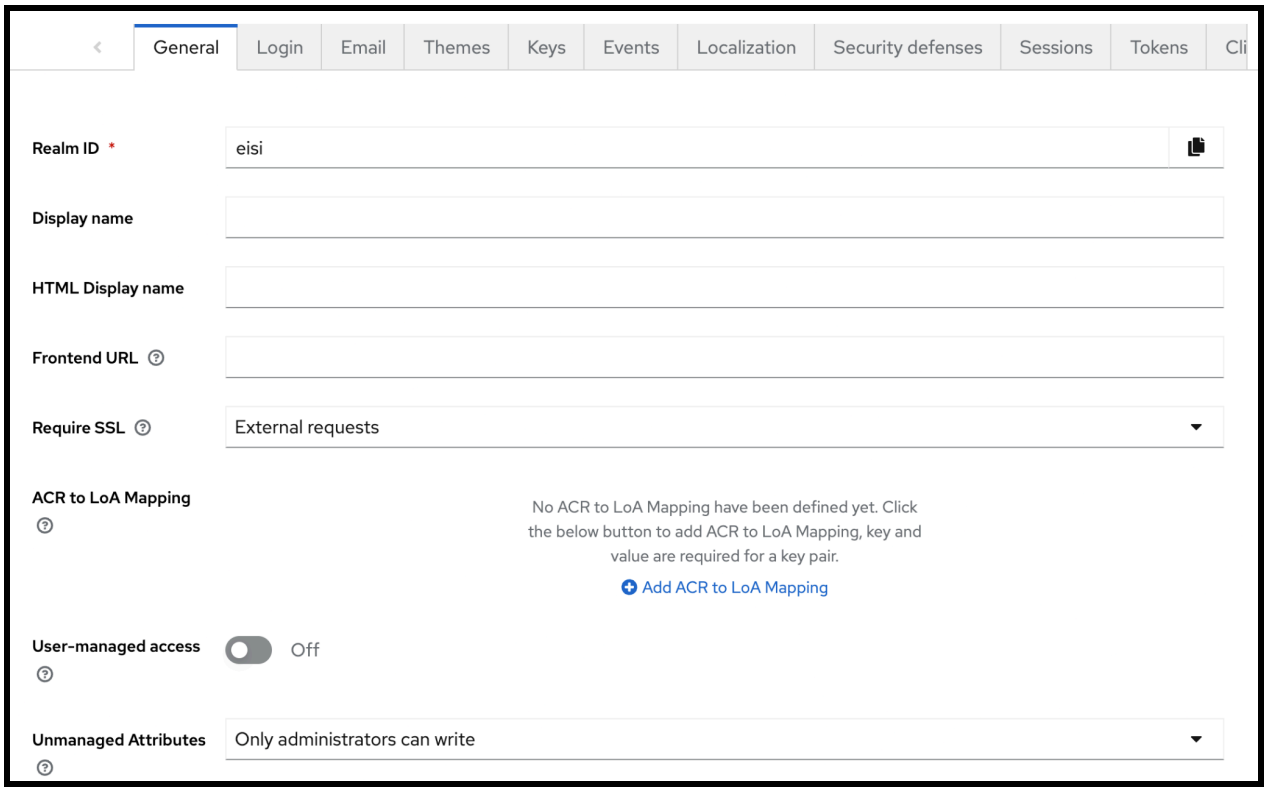
```
grupo: "autenticador"
dominios:
  dev: "auth.eisi.online"
  stg: ""
  prod: "auth.uis.edu.co"
allow_ips: "all"
protegido: false

servicios:
  - proyecto: "keycloak-manager"
    tipo: "front"
    upstream_port: 8080
```

Con Keycloak desplegado y accesible a través del gateway, el siguiente paso es configurar el espacio de identidades que albergará a todos los usuarios de la Escuela de Ingeniería de Sistemas e Informática. En Keycloak este espacio se denomina Realm, y es el contenedor lógico que agrupa usuarios, aplicaciones, roles, políticas de seguridad y flujos de autenticación bajo una identidad institucional común. Crear un Realm dedicado para la escuela garantiza aislamiento total respecto a otros posibles Realms del servidor y permite configurar cada aspecto del sistema de identidades de forma independiente.

Figura 64

Keycloak: Configuración del Realm



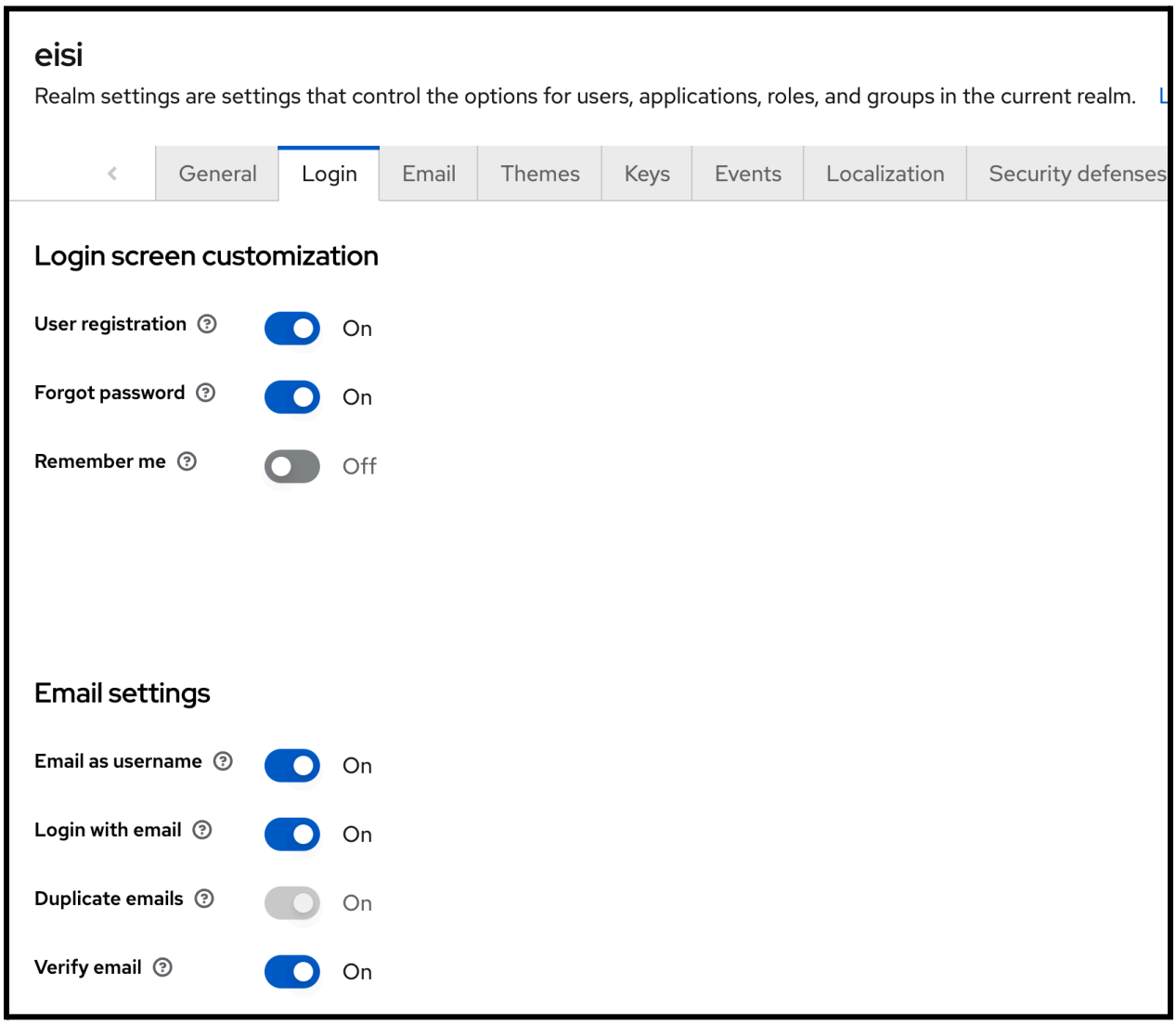
Lo primero es ajustar el comportamiento de los atributos de usuario. En la sección General de Realm Settings, la opción Unmanaged Attributes se establece en "Only administrator can write". Los atributos no gestionados son campos adicionales que no están definidos formalmente en el perfil de usuario pero que pueden ser necesarios en operaciones especiales, como migraciones masivas de estudiantes donde se requiere trasladar datos históricos. Restringir su escritura únicamente a administradores evita que usuarios o aplicaciones externas puedan modificar campos arbitrarios en los perfiles, manteniendo la integridad de los datos institucionales.

En la pestaña de Login se habilitan varias opciones que definen el comportamiento del sistema de autenticación para la escuela. Se activa Email as username porque el identificador de los usuarios será su correo institucional, eliminando la necesidad de gestionar nombres de usuario independientes. Se habilita User registration para que los estudiantes puedan crear su

cuenta desde la pantalla de login. Se activa Forgot password para el flujo de restablecimiento de contraseña. Se habilita Login with email y Verify email, esta última para garantizar que cada usuario que se registre confirme la titularidad de su correo antes de poder acceder.

Figura 65

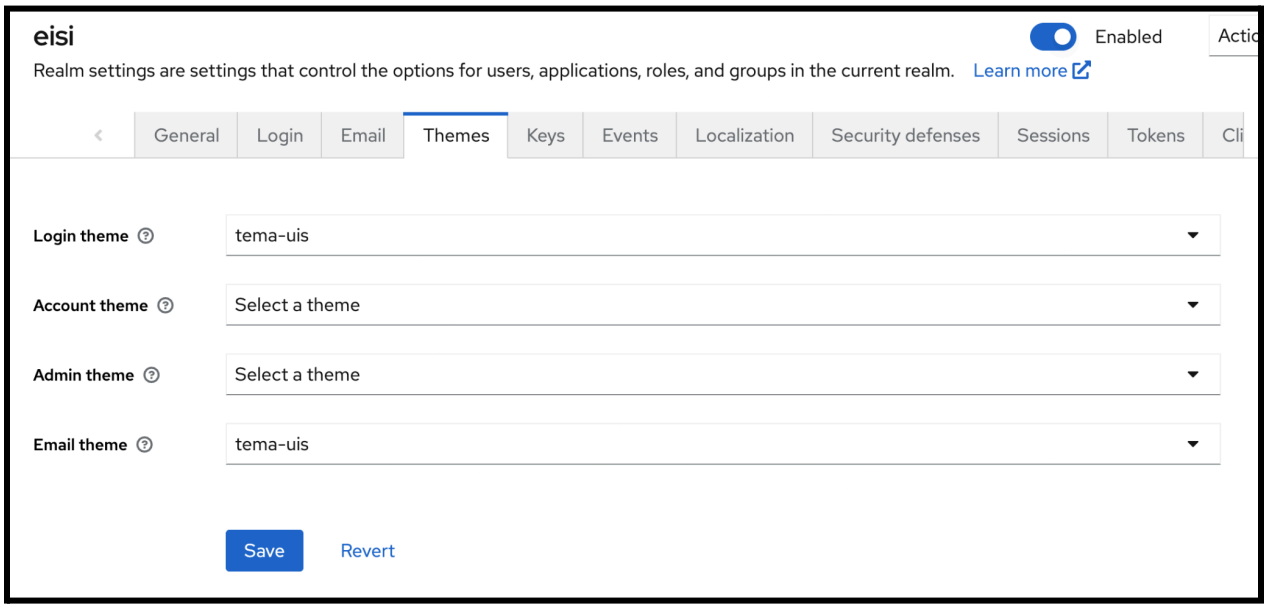
Keycloak Realms: Login



En la pestaña Theme se selecciona tema-uis tanto para el Login Theme como para el Email Theme, aplicando la identidad visual institucional que construimos con las plantillas FTL a todas las pantallas que el usuario verá durante su interacción con el sistema de autenticación.

Figura 66

Keycloak Realms: Themes



En la sección User Profile se carga el esquema JSON con los atributos que conforman el perfil de un usuario institucional: correo institucional, correo personal, código estudiantil, programa académico, teléfono y número de documento. Definir estos atributos formalmente en el perfil de usuario de Keycloak tiene una ventaja importante: permite adjuntarles validaciones, marcarlos como obligatorios o visibles solo para ciertos roles, y garantizar que el formulario de registro los solicite y valide correctamente antes de crear la cuenta.

Figura 67

Keycloak Users

```
{
  "attributes": [
    {
      "name": "username",
      "displayName": "${username}",
      "validations": {
        "length": {
          "min": 3,
          "max": 255
        },
        "username-prohibited-characters": {},
        "up-username-not-idn-homograph": {}
      },
      "permissions": {
        "view": [
          "admin",
          "user"
        ],
        "edit": [
          "admin",
          "user"
        ]
      },
      "multivalued": false
    },
    {
      "name": "email",
      "displayName": "${email}",
      "validations": {
        "email": {},
        "length": {
          "max": 255
        }
      },
      "required": {
        "roles": [
          "user"
        ]
      },
      "permissions": {
        "view": [
          "admin",
          "user"
        ],
```

```
      "edit": [
        "admin",
        "user"
      ]
    },
    "multivalued": false
  },
  {
    "name": "firstName",
    "displayName": "${firstName}",
    "validations": {
      "length": {
        "max": 255
      },
      "person-name-prohibited-characters": {}
    },
    "required": {
      "roles": [
        "user"
      ]
    },
    "permissions": {
      "view": [
        "admin",
        "user"
      ],
      "edit": [
        "admin",
        "user"
      ]
    },
    "multivalued": false
  },
  {
    "name": "lastName",
    "displayName": "${lastName}",
    "validations": {
      "length": {
        "max": 255
      },
      "person-name-prohibited-characters": {}
    },
    "required": {
      "roles": [
```

```
        "user"
      ]
    },
    "permissions": {
      "view": [
        "admin",
        "user"
      ],
      "edit": [
        "admin",
        "user"
      ]
    },
    "multivalued": false
  },
  {
    "name": "studentCode",
    "displayName": "Código estudiantil",
    "validations": {},
    "annotations": {},
    "required": {
      "roles": [
        "user"
      ]
    },
    "permissions": {
      "view": [
        "admin",
        "user"
      ],
      "edit": [
        "admin",
        "user"
      ]
    },
    "multivalued": false
  },
  {
    "name": "academicProgram",
    "displayName": "Programa académico",
    "validations": {
      "options": {
        "options": [
          "10:BIOLOGIA",
```

```
          "11:INGENIERIA DE SISTEMAS",
          "14:QUIMICA",
          "16:LICENCIATURA EN MATEMATICAS",
          "21:INGENIERIA CIVIL",
          "23:INGENIERIA INDUSTRIAL",
          "24:INGENIERIA MECANICA",
          "27:DISEÑO INDUSTRIAL",
          "30:LICENCIATURA EN MUSICA",
          "32:INGENIERIA DE PETROLEOS",
          "33:INGENIERIA QUIMICA",
          "39:MATEMATICAS",
          "40:FISICA",
          "47:INGENIERÍA EN INTELIGENCIA ARTIFICIAL",
          "50:INGENIERIA EN CIENCIA DE DATOS",
          "56:FISIOTERAPIA",
          "57:NUTRICION Y DIETETICA",
          "58:MICROBIOLOGIA Y BIOANALISIS",
          "69:INGENIERIA BIOMEDICA"
        ]
      }
    },
    "annotations": {
      "inputType": "select"
    },
    "required": {
      "roles": [
        "admin",
        "user"
      ]
    },
    "permissions": {
      "view": [
        "admin",
        "user"
      ],
      "edit": [
        "admin",
        "user"
      ]
    },
    "multivalued": false
  },
  {
    "name": "documentNumber",
```

```
    "displayName": "Número de documento",
    "validations": {},
    "annotations": {},
    "required": {
      "roles": [
        "user"
      ]
    },
    "permissions": {
      "view": [
        "admin",
        "user"
      ],
      "edit": [
        "admin",
        "user"
      ]
    },
    "multivalued": false
  },
  {
    "name": "personalEmail",
    "displayName": "Email personal",
    "validations": {
      "email": {
        "max-local-length": "64"
      }
    },
    "annotations": {},
    "required": {
      "roles": [
        "user"
      ]
    },
    "permissions": {
      "view": [
        "admin",
        "user"
      ],
      "edit": [
        "admin",
        "user"
      ]
    },
```

```
      "multivalued": false
    },
    {
      "name": "phone",
      "displayName": "Telefono",
      "validations": {},
      "annotations": {},
      "required": {
        "roles": [
          "user"
        ]
      },
      "permissions": {
        "view": [
          "admin",
          "user"
        ],
        "edit": [
          "admin",
          "user"
        ]
      },
      "multivalued": false
    }
  ],
  "groups": [
    {
      "name": "user-metadata",
      "displayHeader": "User metadata",
      "displayDescription": "Attributes, which refer to user metadata"
    }
  ],
  "unmanagedAttributePolicy": "ADMIN_EDIT"
}
```


Figura 68

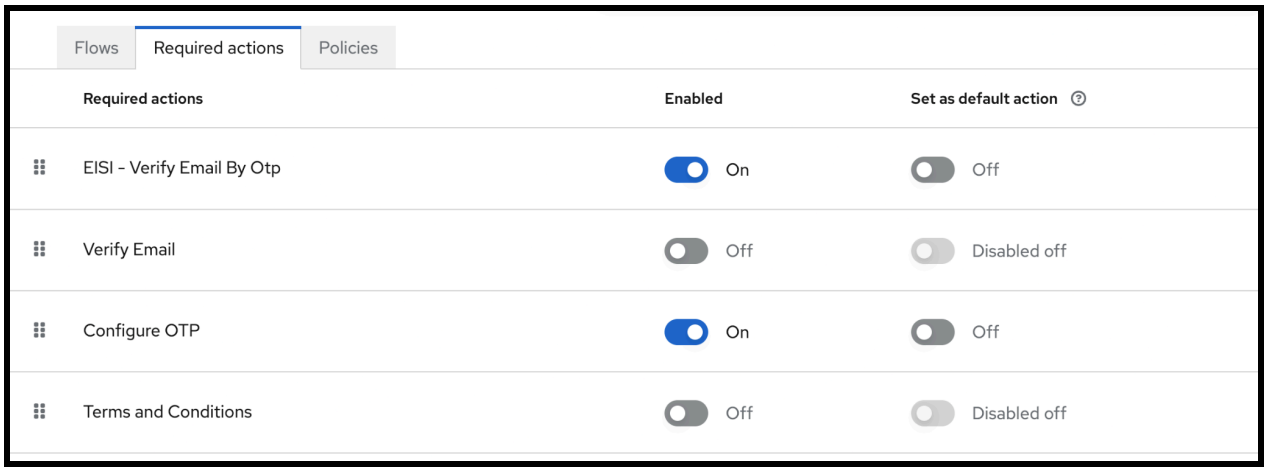
Keycloak Users: Visualización post-configuración

Attribute [Name]	Display name	Attribute group
username	${username}	
email	${email}	
firstName	${firstName}	
lastName	${lastName}	
studentCode	Código estudiantil	
academicProgram	Programa académico	
documentNumber	Número de documento	
personalEmail	Email personal	
phone	Telefono	

En Authentication > Required Actions se gestiona qué acciones debe completar un usuario antes de poder acceder al sistema. Por defecto Keycloak trae habilitada su propia implementación de verificación de correo basada en magic link. Esta implementación se deshabilita y en su lugar se activa la extensión EISI library - Verify Email By Otp que desarrollamos, reemplazando el mecanismo de verificación en todo el Realm de forma transparente.

Figura 69

Required Actions: Flujos necesarios para usuarios

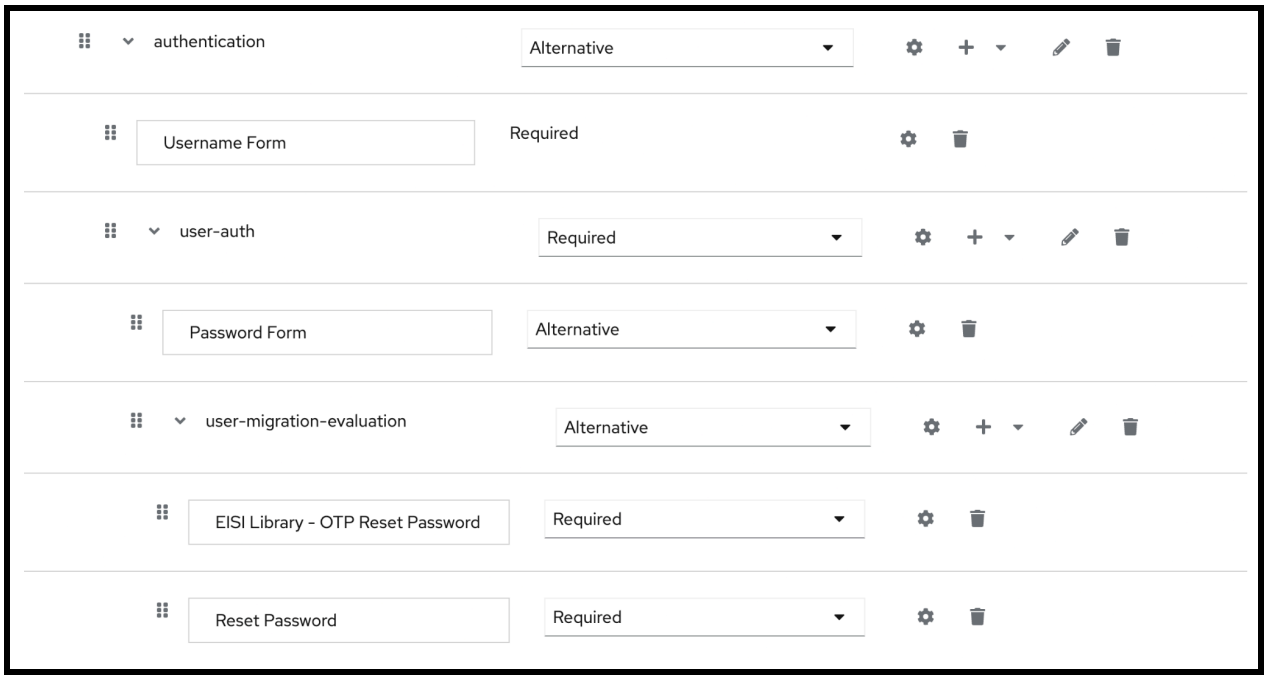


Antes de describir las modificaciones a los flujos, es importante entender qué son y cómo funcionan. Un flujo de autenticación en Keycloak es una cadena de pasos configurables que define exactamente qué debe ocurrir para que una acción, como iniciar sesión, registrarse o restablecer la contraseña, sea considerada exitosa. Cada paso de la cadena puede ser de tipo Required, lo que significa que siempre debe ejecutarse y superarse, o Alternative, lo que significa que si ese paso tiene éxito la cadena puede avanzar sin necesidad de que los pasos alternativos al mismo nivel se ejecuten. Los pasos pueden agruparse en subflows, que son cadenas anidadas con su propia lógica interna de Required y Alternative. Esta arquitectura es la que permite construir flujos complejos como el que necesita la escuela, donde el camino que sigue la autenticación depende del estado del usuario que intenta ingresar.

Keycloak no permite modificar sus flujos predeterminados directamente, por lo que el primer paso es crear copias de los flujos browser, registration y reset credentials. Estas copias son las que se modifican, preservando los flujos originales como referencia.

Figura 70

Flujo de Browser



El flujo de browser es el que gestiona el inicio de sesión regular. La modificación más significativa es la adopción del enfoque Identity First, que separa la solicitud de correo y la solicitud de contraseña en pasos distintos en lugar de presentarlos en un único formulario. Esta separación no es solo una decisión de experiencia de usuario sino una necesidad técnica derivada del proceso de migración de estudiantes.

El laboratorio debe migrar una gran cantidad de estudiantes cuyas cuentas serán creadas administrativamente sin contraseña inicial. Cuando uno de estos usuarios intentan ingresar por primera vez, el sistema necesita reconocer que el correo existe pero que no tiene contraseña asignada, y en lugar de mostrar un error debe guiarlo por el flujo de verificación y creación de contraseña. Esto solo es posible si primero se identifica al usuario y luego se evalúa su estado antes de decidir qué paso sigue.

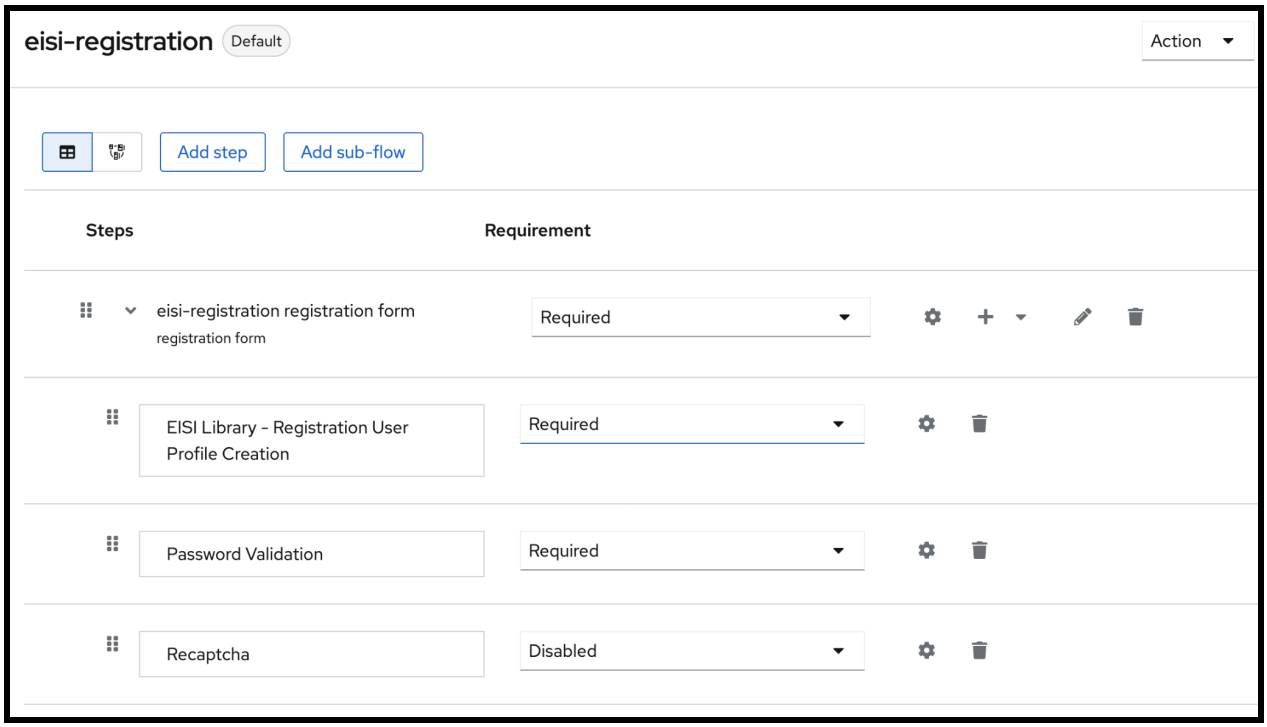
La estructura del flujo modificado funciona de la siguiente manera. El nivel superior mantiene el paso de Cookie como Alternative, que permite que usuarios con sesión activa

ingresen directamente sin volver a autenticarse. Si no hay cookie, el flujo entra a un subflow Alternative llamado authentication. Dentro de este subflow, el primer paso es el formulario Username Form de Keycloak configurado como Required, que solicita únicamente el correo institucional. Una vez identificado el usuario, se entra a un segundo subflow Required llamado user-auth que contiene la lógica de bifurcación.

Dentro de user-auth, el Password Form está configurado como Alternative. Si el usuario tiene contraseña, la ingresa y el flujo termina exitosamente. Si no tiene contraseña porque fue migrado, el Password Form no puede completarse y Keycloak evalúa el siguiente paso Alternative, que es el Authenticator EISI library - OTP Reset Password que desarrollamos. Este Authenticator envía un código OTP al correo del usuario para verificar su identidad. Si la verificación es exitosa, entra el paso Required de Reset Password de Keycloak, que le permite al usuario establecer su contraseña por primera vez. A partir de ese momento, en los siguientes inicios de sesión el usuario simplemente ingresa su correo y contraseña sin pasar por el flujo de verificación.

Figura 71

Flujo de registro

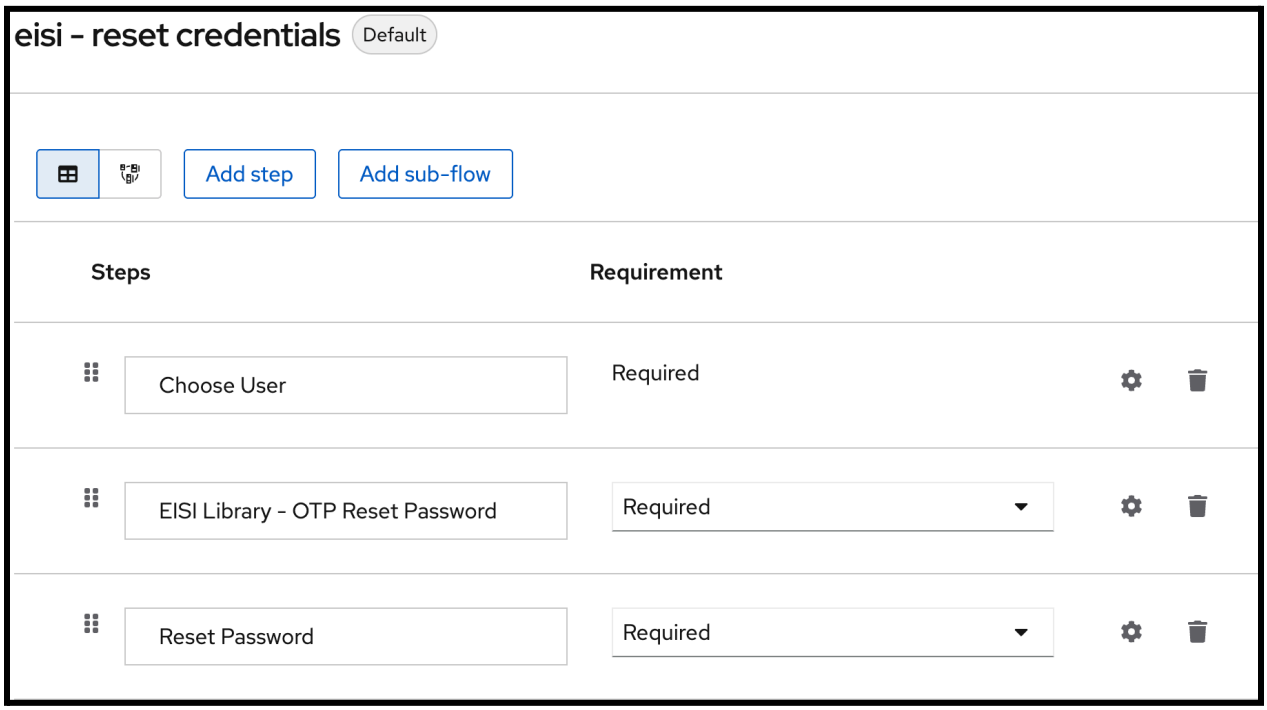


El flujo de registro requiere un cambio puntual pero crítico. Por defecto, Keycloak bloquea el registro si el correo ya existe en el sistema. Este comportamiento representa una vulnerabilidad en el contexto de la escuela: si un estudiante migrado intenta registrarse sin saber que ya tiene cuenta, Keycloak le indica que el correo está en uso pero sin permitirle continuar. Un actor malicioso podría explotar esto realizando registros masivos con correos institucionales conocidos para bloquear el acceso a estudiantes reales antes de que ellos mismos lo intenten.

Para resolver esto, el paso Registration User Profile Creation del flujo se reemplaza por la extensión propia del laboratorio, que permite que un usuario cuyo correo ya existe en el sistema pueda continuar el flujo de registro. El sistema reconoce que la cuenta ya existe, la vincula y guía al usuario por el proceso de verificación y configuración de contraseña en lugar de bloquearlo.

Figura 72

Flujo de reset de credenciales



El flujo de restablecimiento de contraseña original de Keycloak funciona enviando al usuario un correo con un enlace en el que debe hacer clic para acceder a la pantalla de nueva contraseña. Como se explicó anteriormente, este mecanismo es incompatible con las políticas del correo institucional de la UIS que filtra este tipo de mensajes. El cambio en este flujo es reemplazar el paso intermedio de envío de enlace por el Authenticator de OTP que desarrollamos, de forma que el usuario reciba un código numérico en su correo en lugar de un enlace, verifique su identidad ingresando el código y proceda inmediatamente a establecer su nueva contraseña. El resultado final para el usuario es funcionalmente equivalente pero compatible con los filtros del correo institucional.

El proxy admin es el componente que expone el directorio de usuarios de Keycloak a las aplicaciones del ecosistema de forma segura y gobernada. Su propósito es evitar que las aplicaciones interactúen directamente con la API de administración de Keycloak, centralizando y

controlando qué operaciones puede ejecutar cada aplicación y bajo qué condiciones, según el rol del usuario autenticado que realiza la solicitud.

El proxy está construido con FastAPI y se organiza en capas con responsabilidades claramente separadas: seguridad, gobernanza, integración con Keycloak y exposición de endpoints. Esta separación garantiza que cada componente pueda evolucionar de forma independiente sin afectar al resto.

Toda petición que llega al proxy pasa obligatoriamente por dos validaciones de seguridad antes de ejecutar cualquier lógica de negocio.

La primera es la verificación del proxy enrutador. Cada petición debe incluir las cabeceras X-Proxy-Client-Id y X-Proxy-Client-Secret, que identifican al proxy Nginx como el origen autorizado de la solicitud. La comparación de estas credenciales se realiza mediante secrets.compare_digest, una función diseñada específicamente para prevenir ataques de temporización, donde un atacante podría inferir si una credencial es correcta midiendo el tiempo que tarda el servidor en responder. Si estas cabeceras están ausentes o son incorrectas, la petición es rechazada con un error 401 y el intento queda registrado en el log con la IP de origen, el path y el método utilizado.

La segunda validación es la verificación de permisos mediante el decorador RequirePermission. Este componente extrae la identidad del usuario autenticado desde la cabecera X-Authenticated-User, que contiene el token interno inyectado por Nginx con los datos del usuario, obtiene sus roles y los cruza contra el Policy Engine para determinar qué capacidades tiene ese usuario dentro de la aplicación que realiza la solicitud. Si el usuario no posee la capacidad requerida para ejecutar la operación solicitada, la petición es bloqueada con

un error 403 y el intento queda registrado con todos los detalles relevantes: nombre, email, roles, capacidad requerida y endpoint solicitado.

El Policy Engine es el componente que implementa el modelo de control de acceso basado en políticas, conocido como PBAC. Al iniciar el servicio, carga los archivos YAML de configuración del directorio de gobernanza, donde cada archivo define las reglas de acceso para una aplicación específica identificada por su 'client_id'.

Figura 73

Permisos de usuarios en Keycloak: Governance y Capabilities

```yaml
client_id: "SAAM_EISI_PROJECT"
rol_prefix: ""

roles_gestionados:
  - rol_keycloak: "ROOT"
    descripcion: "Administrador de la app SAAM"
    capabilities:
      - "users:read_detail"
      - "roles:read_members"

  - rol_keycloak: "STUDENT"
    descripcion: "Estudiante de la app SAAM"
    capabilities:
      - "users:read_detail"
```

Cada archivo define qué roles de Keycloak existen para esa aplicación y qué capacidades tiene cada uno. Cuando el Policy Engine recibe una consulta de permisos, no evalúa si el usuario es un estudiante o un administrador: simplemente pregunta si el usuario tiene la capacidad requerida para esa operación en esa aplicación. Esta abstracción desacopla completamente la lógica de autorización de los nombres de roles específicos, permitiendo que cada aplicación defina su propia jerarquía de acceso sin modificar ningún código.

El motor también gestiona el prefijo de roles, lo que permite que múltiples aplicaciones compartan el mismo realm de Keycloak sin colisión de nombres, y soporta roles ad-hoc que pueden crearse dinámicamente en Keycloak cuando una aplicación los necesita y no estaban definidos previamente en el YAML.

La comunicación con Keycloak se centraliza en el KeycloakService, un adaptador sobre la librería python-keycloak que simplifica las operaciones del servidor de identidad y abstrae su complejidad hacia el resto del sistema.

La búsqueda de usuarios soporta tres modalidades que pueden combinarse: búsqueda textual general por nombre o usuario, filtros sobre campos base como el correo electrónico, y búsqueda por atributos personalizados usando la sintaxis nativa de Keycloak. Esto permite que las aplicaciones ejecuten consultas complejas sobre el directorio sin conocer los detalles de la API de Keycloak.

Para la consulta de detalle de un usuario, el servicio enriquece la respuesta básica de Keycloak agregando los roles de realm asignados al usuario, consolidando en una sola respuesta toda la información que la aplicación necesita. La asignación de roles distingue entre roles gestionados, definidos en el YAML de gobernanza, y roles ad-hoc, que el servicio crea automáticamente en Keycloak si no existen antes de asignarlos, con manejo de conflictos en caso de que ya existan.

El proxy expone cuatro operaciones sobre el directorio, cada una protegida por una capacidad específica que el usuario debe tener según la configuración del Policy Engine de su aplicación. La operación GET /directory/users lista usuarios con paginación y soporte de filtros, requiriendo la capacidad directory:read. La operación GET /directory/users/{uuid} retorna el

detalle completo de un usuario, requiriendo users:read_detail. La operación GET /directory/roles/{role}/users lista los miembros de un rol específico, requiriendo roles:read_members. Finalmente, POST /directory/roles/assign asigna un rol a un usuario, requiriendo roles:assign_managed para roles gestionados o roles:assign_adhoc para roles ad-hoc.

Todas las respuestas pasan por una función de sanitización que elimina los campos técnicos internos de Keycloak antes de devolverlos a la aplicación solicitante, reduciendo el volumen de datos transferidos y evitando exponer metadatos internos del servidor de identidad.

Adicionalmente el proxy expone un endpoint de envío de correo institucional mediante SMTP conectado a Azure, con validación de dominio para distinguir entre destinatarios internos de la universidad y externos, protegido por las mismas credenciales de proxy que el resto de los endpoints.

5.3.3 *Interfaces de usuario funcionales y Sistema integrado y validado*

5.3.3.1 Integración de SAAM con el autenticador. Con el Realm configurado y el gateway operativo, el siguiente paso es demostrar la integración completa conectando una aplicación real al ecosistema de autenticación. Para esto se utiliza SAAM, que será la primera aplicación del laboratorio en consumir el sistema de identidades institucionales. El proceso de integración involucra tres actores que deben coordinarse: el repositorio de la aplicación, el API Gateway y Keycloak.

5.3.3.2 Declaración en el API Gateway. Configuración de Gateway:

Figura 74

Configuración de SAAM en el Gateway

```yaml
grupo: "SAAM"
dominios:
  dev: "saam.eisi.online"
  stg: ""
  prod: "saam.uis.edu.co"
allow_ips: "all"
protegido: true

servicios:

  - proyecto: "saam_frontend"
    tipo: "front"
    upstream_port: 80

  - proyecto: "saam_backend"
    tipo: "back"
    api_root: "/saam"
    upstream_port: 3000
    tls: false
    rutas:
      #default
      - "/api"
      - "/health"

      # Autenticación
      - "/auth/login"
      - "/auth/register"
      - "/auth/user"
      - "/auth/verify"
      - "/auth/refresh-token"
      - "/auth/resend-verify"
      - "/auth/recover-password"
      - "/auth/change-password"
      - "/auth/verify-email"
      - "/auth/me"

      # Solicitudes
```

```
      - "/appeal"
      - "/appeal/count"
      - "/appeal/{id}"
      - "/appeal/{id}/scale"

      # Horario
      - "/schedule"
      - "/schedule/count"
      - "/schedule/{id}"

      #Estudiantes
      - "/student"

      # Materias  - Administrador
      - "/subjects"
      - "/subjects/count"
      - "/subjects/{sku}"

      # Usuarios
      - "/users"
      - "/users/count"
      - "/users/{id}"

variables: # variables de entorno
   - GW_INTERNAL_SECRET: GW_INTERNAL_SECRET_SAAM
   - CLIENT_ID: CLIENT_ID_SAAM
   - CLIENT_SECRET: CLIENT_SECRET_SAAM
   - OIDC_CLIENT_ID: OIDC_CLIENT_ID_SAAM
   - OIDC_CLIENT_SECRET: OIDC_CLIENT_SECRET_SAAM
```

Lo primero es registrar SAAM en el gateway mediante su archivo de declaración YAML. Esta configuración incluye tanto el frontend como el backend del proyecto, sus dominios por ambiente, sus rutas y redes. El elemento que activa toda la maquinaria de autenticación es la bandera protegido: true, que le indica al generador que debe inyectar en el bloque Nginx de este proyecto toda la lógica Lua de verificación OIDC, gestión de sesiones y generación del Phantom Token.

Al activar la protección, el YAML requiere declarar un conjunto de variables que el gateway necesita para operar de forma segura. La primera es GateWay Internal Secret, un valor

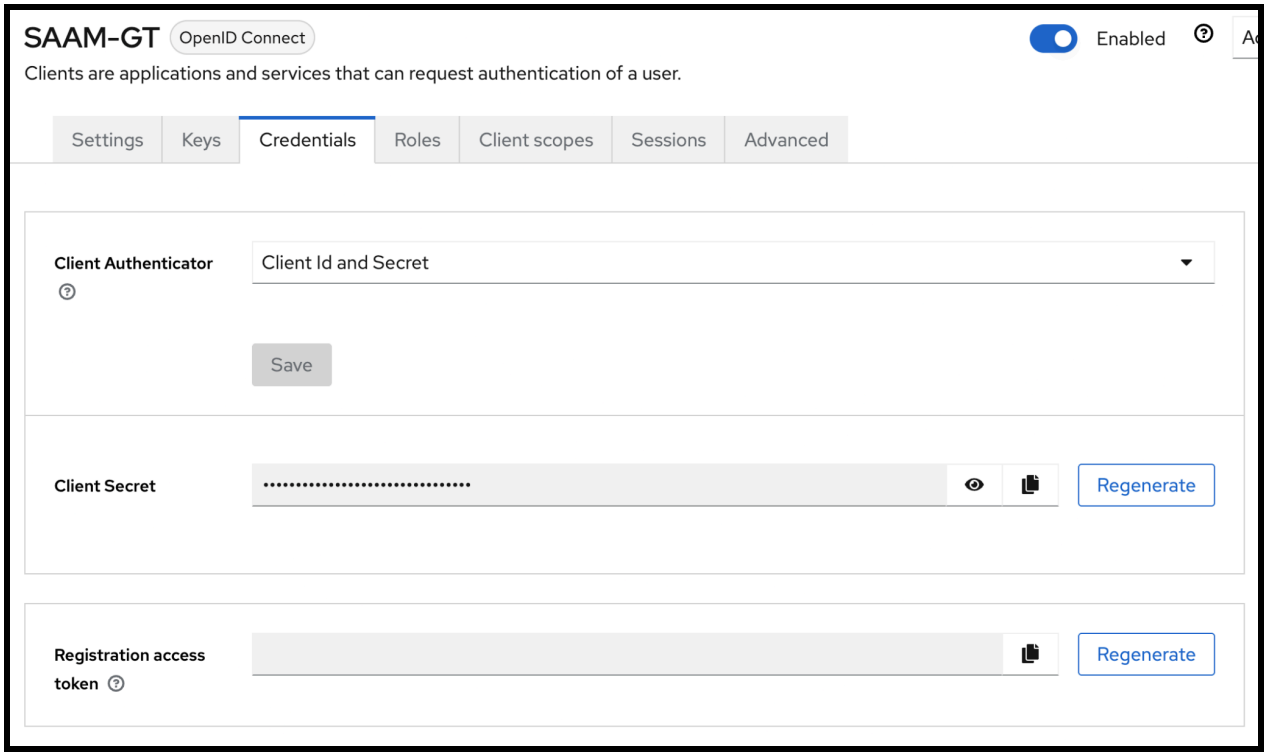
hexadecimal que el gateway usa como llave para firmar el Internal Token mediante HMAC simétrico. Esta firma es la que permite al backend de SAAM verificar que el token que recibe en cada petición fue generado legítimamente por el gateway y no fue fabricado por un tercero. Al ser una firma simétrica, el backend debe conocer esta misma llave para verificarla, por lo que debe declararse también como secreto en el contenedor del backend. El identificador de cliente y el secreto de cliente son las credenciales propias de la aplicación SAAM dentro del gateway, utilizadas cuando el backend necesita consultar información del usuario autenticado al endpoint /auth/userinfo del gateway, identificándose como un cliente legítimo antes de recibir los datos.

Las variables “OIDC_CLIENT_ID” y “OIDC_CLIENT_SECRET” son distintas a las anteriores: son las credenciales que el gateway usa para comunicarse con Keycloak en nombre de SAAM durante el flujo de autenticación OIDC. Estos valores no se inventan sino que los provee Keycloak al registrar la aplicación como cliente, por lo que antes de completar la configuración del gateway es necesario crear el cliente en Keycloak.

5.3.3.3 Creación del cliente en Keycloak

Figura 75

Cliente SAAM en Keycloak



En Keycloak, un cliente representa una aplicación que delega la autenticación de sus usuarios al sistema de identidades. Crear el cliente para SAAM genera el par "OIDC Client Id" y "OIDC Client Secret" que el gateway necesita. Durante la creación se configura el tipo de cliente como confidential, lo que habilita la generación de un secreto, y se declaran las URLs autorizadas para redirección y logout que Keycloak aceptará durante los flujos de autenticación.

Cada aplicación protegida por el gateway tiene dos endpoints reservados que no llegan a los servicios internos sino que son interceptados y procesados directamente por la lógica Lua del gateway.

El primero es https://saam.uis.edu.co/_oauth_callback. Este endpoint es el punto de retorno que Keycloak usa al finalizar la autenticación del usuario. Cuando el gateway redirige al usuario a Keycloak para que inicie sesión, le indica a Keycloak que una vez completada la

autenticación debe redirigir de vuelta a este endpoint, enviando el código de autorización y el estado de la sesión. El gateway intercepta esta redirección, intercambia el código por los tokens de acceso e identidad con Keycloak, crea la sesión en Redis y redirige al usuario a la aplicación ya autenticada. Este endpoint debe registrarse explícitamente en la configuración del cliente en Keycloak como URL de redirección autorizada, ya que Keycloak rechaza cualquier redirección hacia URLs no declaradas previamente como medida de seguridad contra ataques de redirección abierta.

El segundo es https://saam.uis.edu.co/_oidc_logout. Este endpoint implementa el Single Logout, que es la capacidad de cerrar sesión en todas las aplicaciones simultáneamente cuando el usuario cierra sesión en cualquiera de ellas. Cuando el usuario inicia el cierre de sesión, Keycloak notifica a todos los clientes registrados llamando a sus respectivos endpoints de logout con el token de sesión correspondiente. El gateway intercepta esta llamada, destruye la sesión local en Redis y confirma el cierre a Keycloak. Esto garantiza que no queden sesiones huérfanas en ninguna aplicación después de que el usuario cierre sesión, un requisito fundamental en un sistema de identidades institucional donde un usuario podría tener sesiones abiertas en múltiples aplicaciones de la escuela simultáneamente.

Figura 76

Configuración del cliente SAAM en Keycloak

Access settings

Root URL

Home URL

Valid redirect URIs
https://saam.eisi.online/_oauth_callback
Add valid redirect URIs

Valid post logout redirect URIs
https://saam.eisi.online/
Add valid post logout redirect URIs

Web origins
https://saam.eisi.online
Add web origins

Admin URL

Logout settings

Front channel logout
On

Front-channel logout URL
https://saam.eisi.online/_oidc_logout

Backchannel logout URL
https://saam.eisi.online/_oidc_logout

Backchannel logout session required
On

Backchannel logout revoke offline sessions
Off

5.3.3.4 Pruebas de integración con SAAM. Con toda la infraestructura configurada, el siguiente paso es validar que el ciclo completo de autenticación funciona de extremo a extremo utilizando la aplicación SAAM como caso de prueba real. Para entender las pruebas es importante recordar cómo funciona el flujo completo desde la perspectiva del usuario y del sistema.

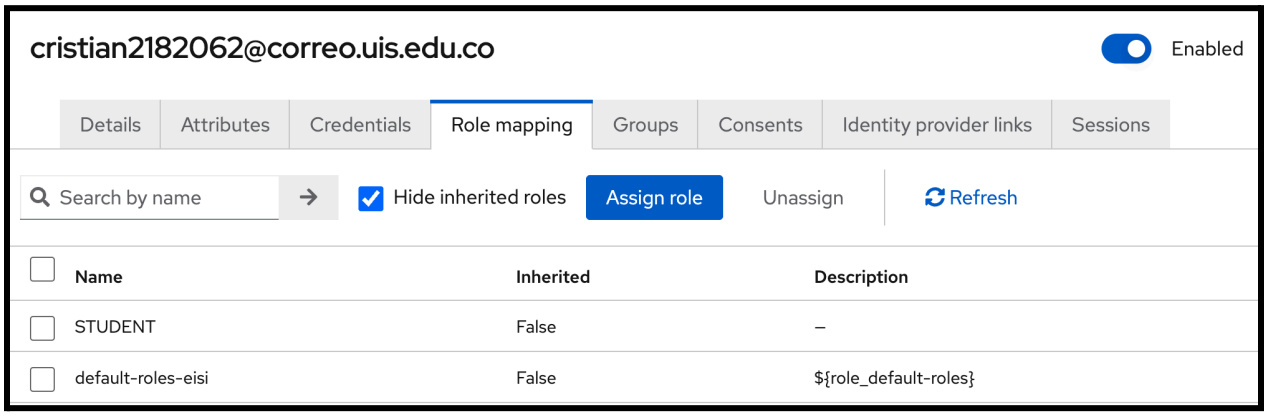
Cuando un usuario intenta acceder a saam.uis.edu.co, la petición llega al API Gateway, que identifica que no existe ni cookie ni sesión activa en Redis para ese navegador. El gateway redirige al usuario a Keycloak, donde se presenta la pantalla de autenticación institucional. Una vez que el usuario se autentica exitosamente, Keycloak redirige de vuelta a https://saam.uis.edu.co/_oauth_callback incluyendo en la URL el código de autorización y el estado de la sesión. El gateway intercepta esta redirección, intercambia el código por los tokens de acceso e identidad con Keycloak, abre una sesión en Redis relacionando esos tokens con una cookie, y devuelve al navegador los recursos del frontend. A partir de ese momento, cada vez que el frontend hace una petición hacia /api, el gateway verifica la sesión asociada a la cookie, construye el Internal Token firmado con la clave simétrica compartida y lo inyecta en el header X-Internal-Token antes de reenviar la petición al backend, que nunca recibe credenciales de Keycloak directamente.

Para soportar este modelo, el backend de SAAM fue ajustado con un interceptor que extrae el header X-Internal-Token de cada petición entrante, verifica su firma HMAC usando la misma clave simétrica configurada en el gateway, valida el campo aud para confirmar que el token fue emitido para SAAM, y extrae los roles del usuario para aplicar control de acceso. Se definieron dos roles: Estudiante, que permite crear solicitudes de cambio de horario, y Raíz, asignado al director o coordinador, que permite visualizar y aprobar las solicitudes recibidas.

5.3.3.5 Asignación de roles y flujo de primer acceso. Página de roles en Keycloak:

Figura 77

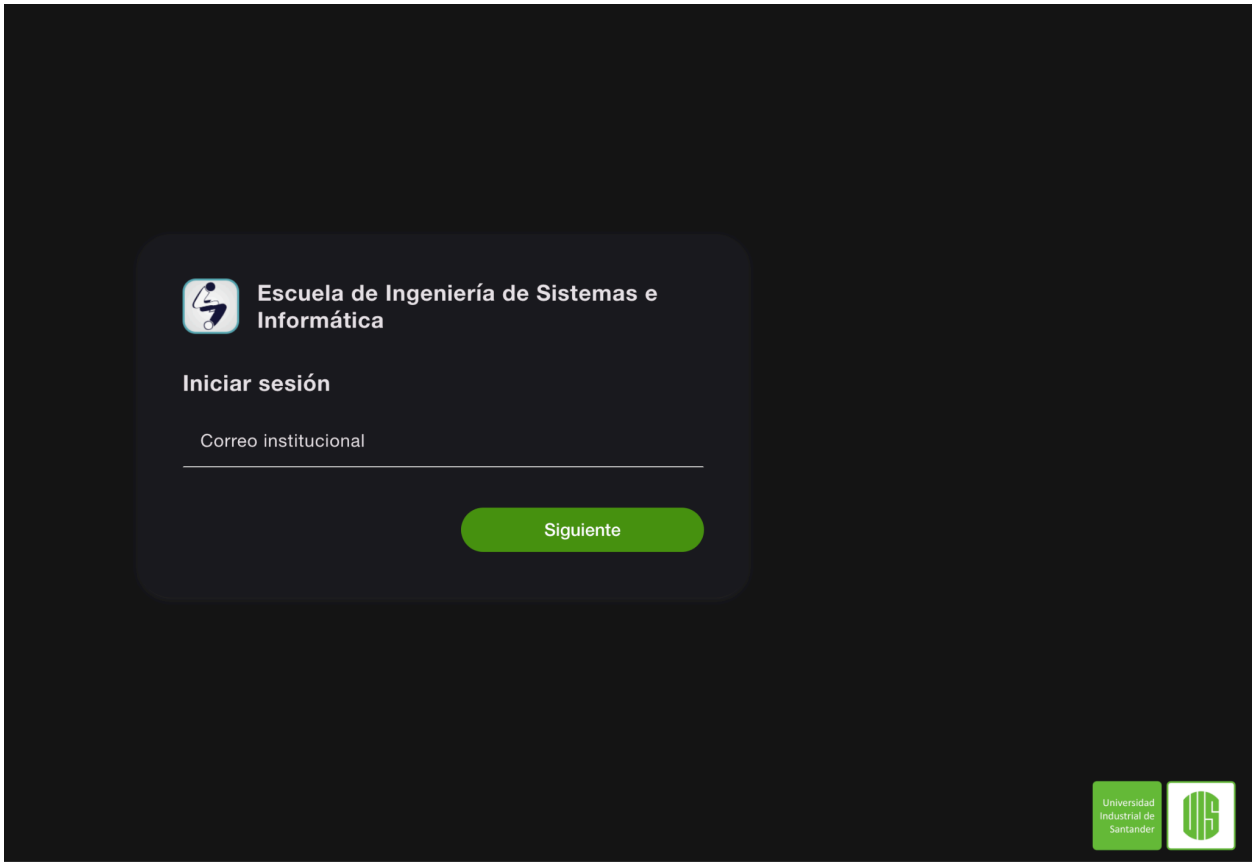
Asignación de roles de un usuario en Keycloak



Para las pruebas se utiliza el usuario cristian2182062@correo.uis.edu.co, al que se le asigna el rol Estudiante en la consola de administración de Keycloak. Al intentar acceder a SAAM por primera vez, el sistema presenta el flujo de autenticación Identity First que configuramos.

Figura 78

Identity First Log-in

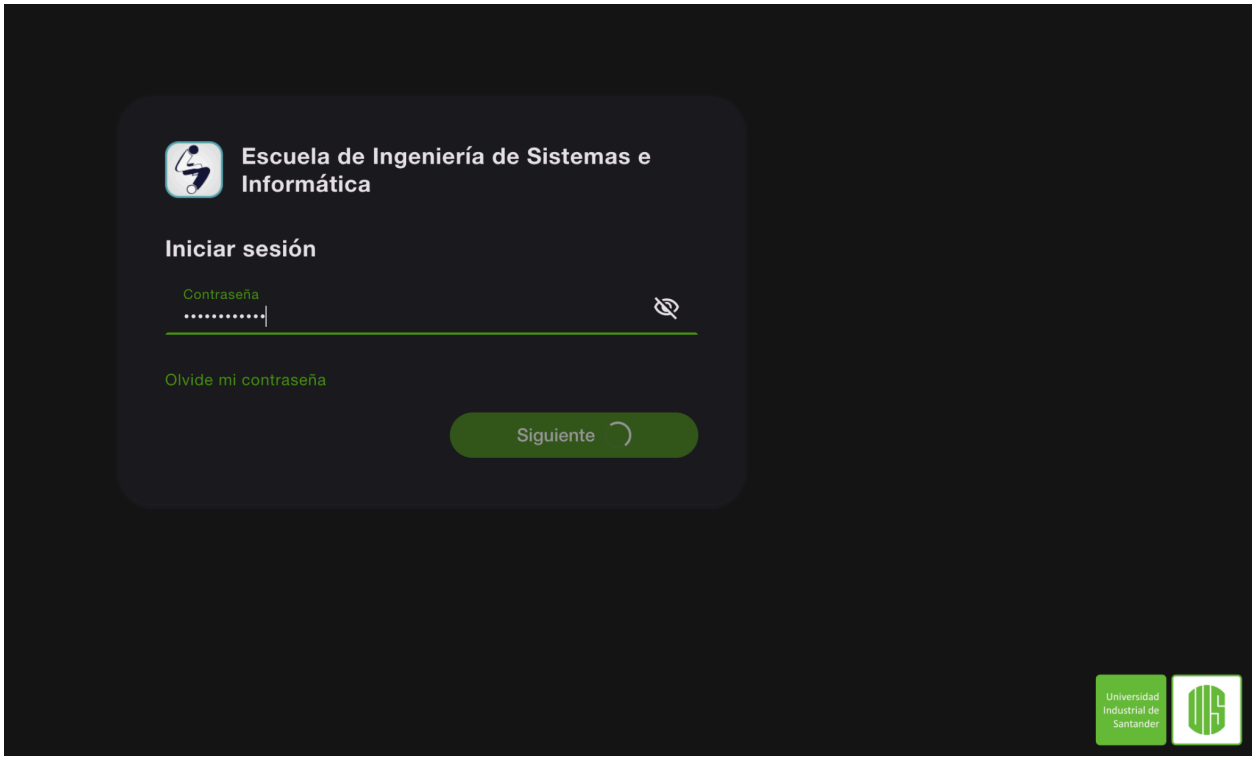


El primer paso solicita únicamente el correo institucional. Al ingresar, el sistema identifica al usuario y evalúa su estado.

Como la cuenta fue registrada sin verificación de correo previa, el sistema activa la Required Action de verificación OTP, enviando un código numérico al correo institucional del usuario.

Figura 79

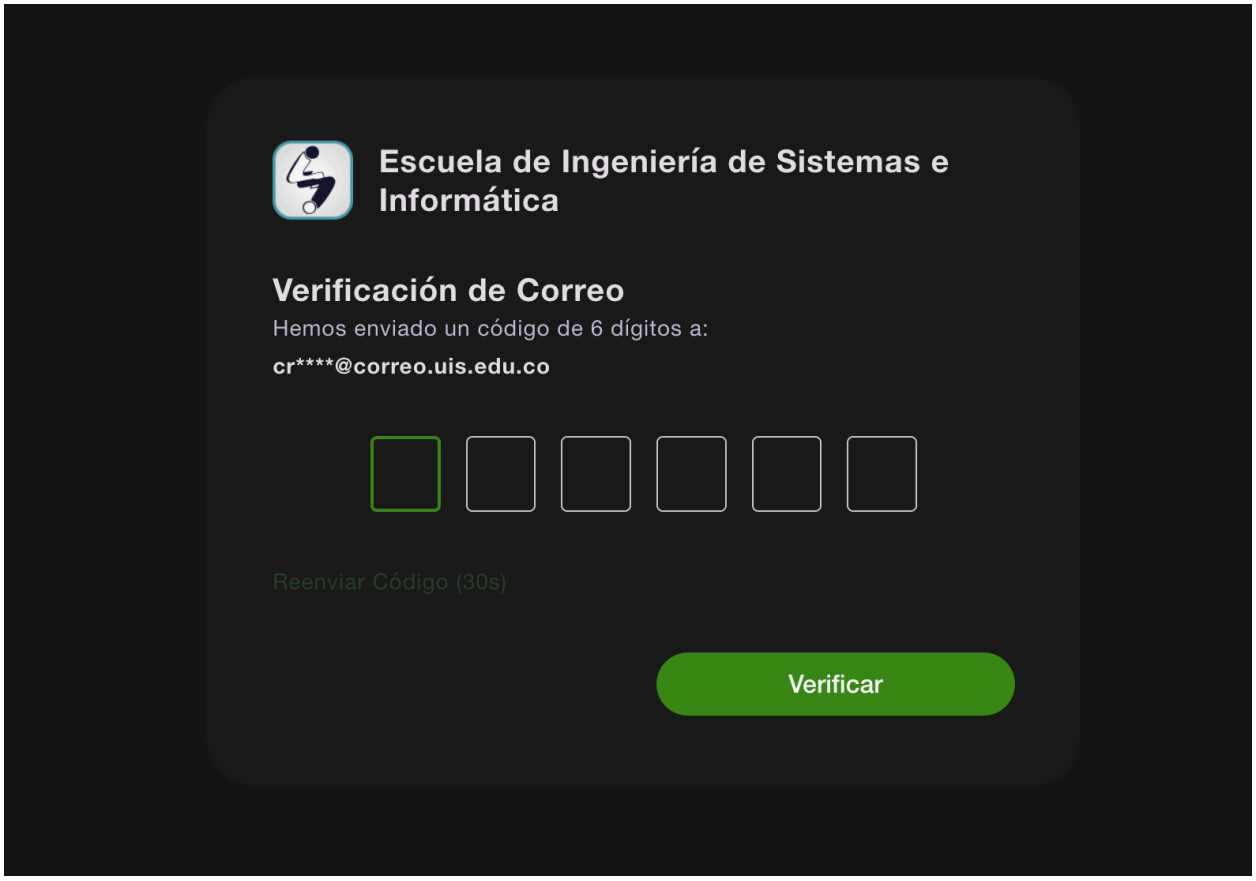
Paso 2 del log-in: contraseña



Una vez verificado el correo, el sistema solicita la contraseña. En los siguientes inicios de sesión el usuario sólo necesitará su correo y contraseña sin pasar nuevamente por la verificación.

Figura 80

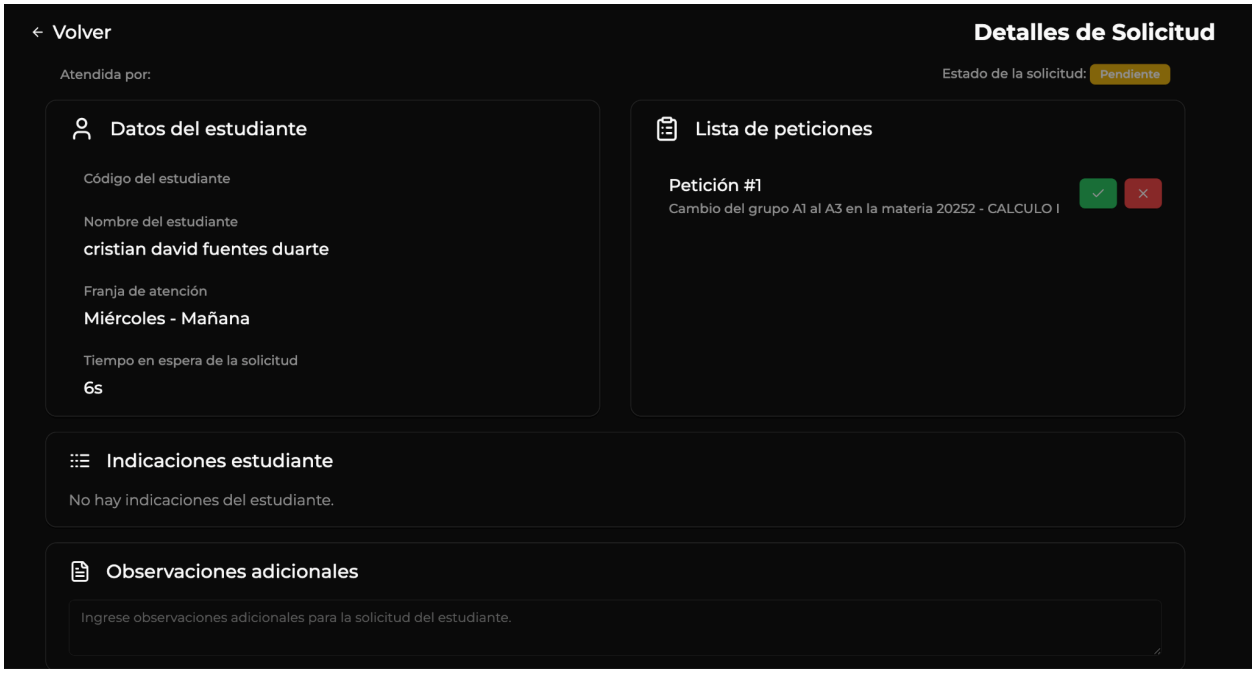
Paso 3 del log-in: Código OTP



Dentro de SAAM se puede crear una solicitud de cambio de horario. El sistema muestra el nombre completo del estudiante, lo cual requiere una aclaración técnica importante: el Internal Token contiene el campo sub, que es el identificador único del usuario en Keycloak, un valor alfanumérico que no comunica información legible como el nombre. Para resolver esto, el backend de SAAM implementa un interceptor adicional que toma ese sub y lo consulta contra el Proxy IAM del gateway en el endpoint /auth/userinfo, autenticandose con sus credenciales de Client Id y Client Secret. El gateway verifica las credenciales, valida el token y devuelve los atributos del usuario incluyendo su nombre completo. De esta forma el backend obtiene información de identidad sin almacenarla localmente ni depender de Keycloak directamente.

Figura 81

Sesión Iniciada en SAAM



Cambiando el rol del usuario a Raíz, la interfaz de SAAM muestra las solicitudes recibidas con el nombre del estudiante que las generó, validando que el control de acceso basado en roles funciona correctamente a través del Internal Token. La solicitud creada por el estudiante es visible y puede ser aprobada por el coordinador, demostrando el ciclo completo de la aplicación integrada con el sistema de identidades.

La prueba de Single Logout confirma que al cerrar sesión en SAAM, Keycloak notifica el cierre a todos los clientes registrados. Si el usuario accede posteriormente a otra aplicación como UISrooms, el gateway detecta que no hay sesión activa y reinicia el flujo de autenticación, pero como Keycloak recuerda la sesión del navegador, el proceso de code flow se completa sin solicitar credenciales nuevamente, emitiendo nuevas cookies y tokens de forma transparente para el usuario.

Figura 82

Solicitud de Ajuste de Matrícula en SAAM

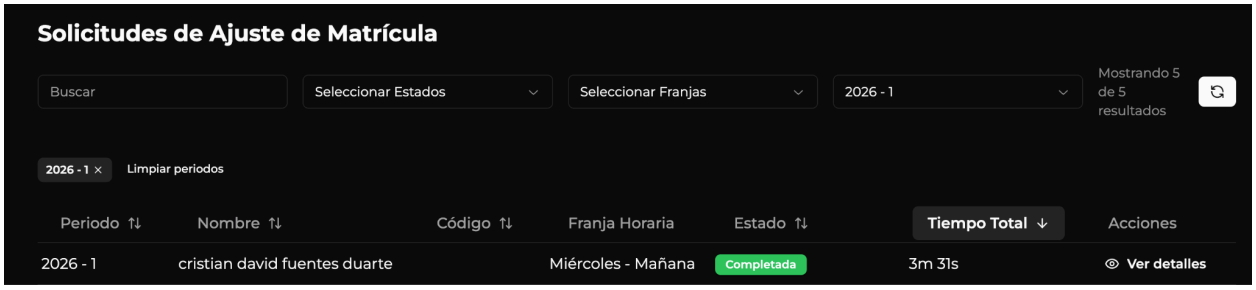
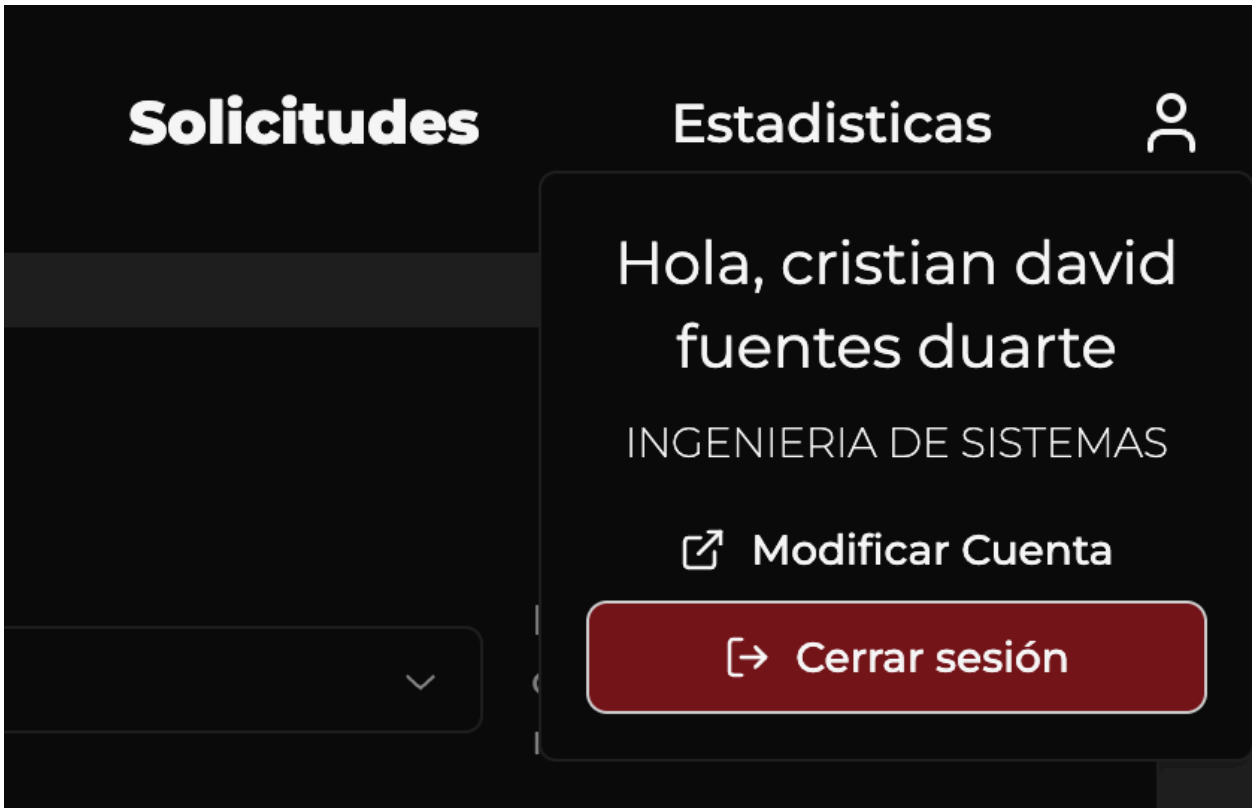


Figura 83

Cierre de sesión en SAAM



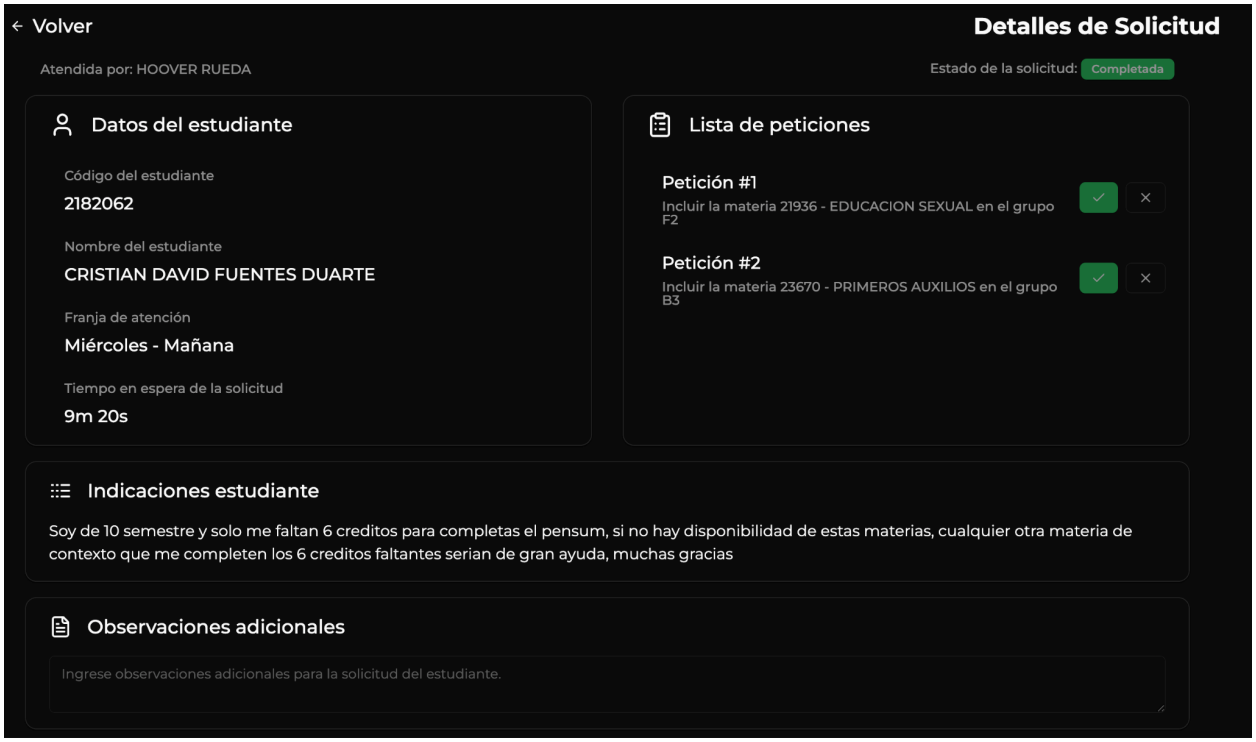
5.4 Fase de transición

5.4.1 *Sistema en producción y validación funcional*

5.4.1.1 Despliegue en producción y migración masiva. Una vez validado el sistema en el ambiente de desarrollo, se procedió al despliegue en producción. Este proceso incluyó la configuración de los dominios productivos y una migración masiva de usuarios al Identity Provider, cargando para cada estudiante sus datos institucionales como programa académico, correo institucional y código estudiantil. La migración funcionó correctamente con el flujo Identity First: los estudiantes migrados que accedían por primera vez eran reconocidos por el sistema, verificaban su correo mediante OTP y establecen su contraseña sin necesidad de que el equipo enviará correos masivos con credenciales preestablecidas, eliminando un vector de vulnerabilidad común en procesos de migración tradicionales. El sistema fue puesto a prueba con casos reales: el autor de este trabajo realizó una solicitud de cambio de horario con su usuario institucional, que fue atendida y aprobada por el coordinador de la escuela, validando la integración de extremo a extremo en ambiente productivo.

Figura 84

Usuario en SAAM

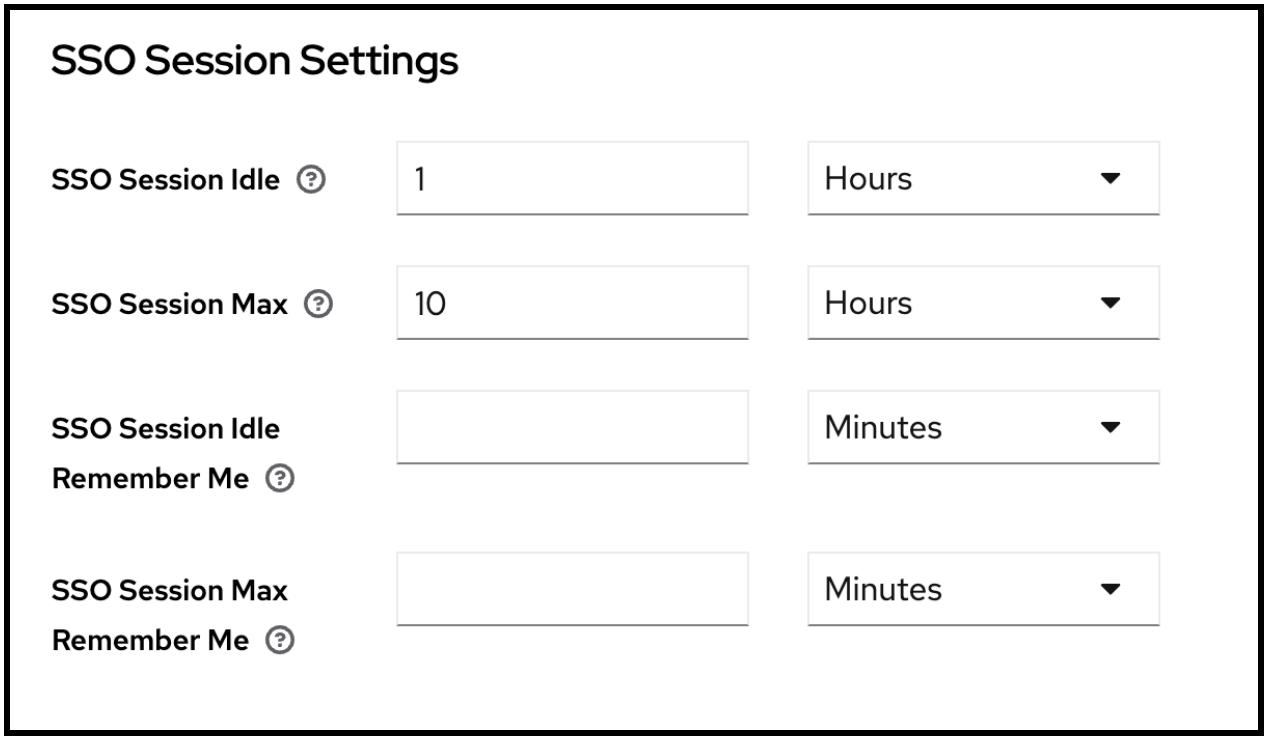


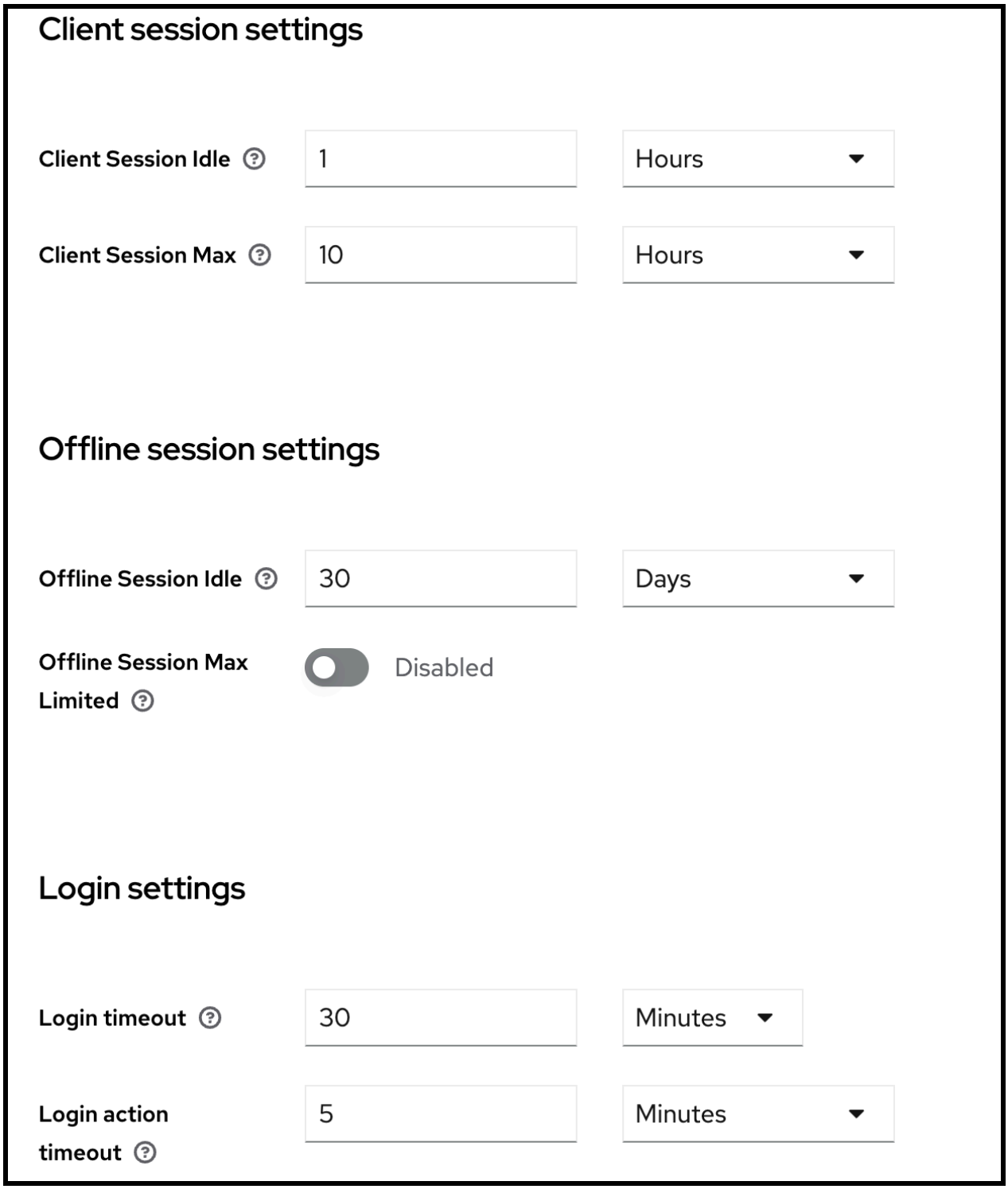
5.4.1.2 Ajustes post-despliegue. El despliegue masivo expuso dos problemas que requirieron corrección en caliente. El primero fue reportado por directores y coordinadores: sus sesiones expiraban con demasiada frecuencia, interrumpiendo su trabajo. El problema radica en que el intervalo de refresco del token en el gateway era demasiado corto. Se ajustó refresh_session_interval a 2700 segundos, es decir 45 minutos, que es el tiempo máximo que puede transcurrir entre peticiones antes de que el gateway renueve el token de acceso automáticamente mientras el usuario esté activo.

Figura 85

Configuración de sesiones en Keycloak

SISTEMA PROVEEDOR DE IDENTIDAD





Complementariamente se configuraron dos parámetros en Keycloak. El Client Session Idle define cuánto tiempo puede estar una sesión sin actividad antes de expirar: si el usuario no realiza ninguna acción durante ese período la sesión se cierra y debe autenticarse nuevamente. Se

configuró en una hora. El Client Session Max define el tiempo máximo absoluto de una sesión independientemente de la actividad del usuario: aunque el usuario esté activo, al cumplirse este tiempo la sesión expira y debe renovarse. Se configuró en diez horas, cubriendo una jornada laboral completa. La combinación de estos tres valores garantiza que un usuario activo nunca vea su sesión interrumpida durante su jornada, mientras que las sesiones abandonadas se limpian en un tiempo razonable.

El segundo problema afectaba a algunos usuarios que veían una pantalla de error al ser redirigidos de Keycloak de vuelta a SAAM tras autenticarse. La causa era que el Internal Token, que contiene los atributos del usuario incluyendo roles, nombre completo y datos institucionales, generaba un header HTTP tan grande que supera el tamaño del buffer de Nginx, causando un error de proxy antes de llegar al backend. La solución fue aumentar los valores de buffer en la configuración de Nginx: proxy_buffer_size, proxy_buffers, proxy_busy_buffers_size y large_client_header_buffers se incrementaron para acomodar los headers de mayor tamaño. Adicionalmente se aumentaron los timeouts de conexión, envío y lectura del proxy a 300 segundos para evitar interrupciones en operaciones que requieren más tiempo de procesamiento. Estos ajustes resolvieron completamente el problema sin necesidad de reducir la información transportada en el token.

5.4.1.3 Retroalimentación de usuarios. La recepción del sistema por parte de estudiantes, directores y coordinadores fue positiva. El flujo de autenticación resultó intuitivo: ingresar el correo, verificar la identidad mediante un código numérico y establecer la contraseña fue un proceso que los usuarios completaron sin necesidad de instrucciones adicionales. Esto validó la decisión de usar OTP numérico en lugar de magic links, que además de ser compatible con los filtros del correo institucional resultó más familiar para los usuarios. La posibilidad de

que los estudiantes no migrados se registraran directamente con su correo institucional eliminó la necesidad de gestionar y distribuir credenciales iniciales, reduciendo tanto la carga operativa del equipo como los riesgos de seguridad asociados al envío masivo de contraseñas.

5.4.2 *Documentación técnica*

La documentación técnica del proyecto fue centralizada en una plataforma tipo wiki, accesible a través de la dirección https://wiki.eisi.internal. Esta solución fue adoptada con el objetivo de garantizar la organización, actualización continua y disponibilidad de la información relevante para el uso, mantenimiento y evolución del sistema desarrollado.

El uso de una wiki como herramienta de documentación permite mantener un espacio vivo de conocimiento, donde la información puede ser actualizada conforme evoluciona el sistema. Este enfoque contribuye a la sostenibilidad del proyecto a largo plazo, reduciendo la dependencia del conocimiento tácito y promoviendo buenas prácticas de documentación dentro de la Escuela.

Finalmente, la documentación técnica cumple un papel clave en la transferencia de conocimiento, ya que permite que futuros equipos puedan comprender, utilizar y extender la solución propuesta sin necesidad de intervención directa del autor, garantizando así la continuidad del sistema en el tiempo.

6. Conclusiones

El presente proyecto logró diseñar e implementar una solución integral que aborda de raíz los dos problemas estructurales identificados en el análisis del estado actual de la Escuela: la ausencia de una infraestructura de desarrollo organizada y la fragmentación en la gestión de identidades de los usuarios.

La implementación del laboratorio de desarrollo estableció por primera vez un entorno profesional y estandarizado para todos los equipos de la Escuela, con ambientes diferenciados, acceso controlado mediante VPN, herramientas centralizadas de versionamiento, calidad de código, monitoreo y gestión de secretos, y procesos automáticos de aprovisionamiento e integración continua. Este entorno demostró ser el prerrequisito indispensable para el desarrollo del Sistema Proveedor de Identidad: sin él, la coordinación entre equipos y la validación de integraciones hubiera sido prácticamente inviable.

El Sistema Proveedor de Identidad fue integrado exitosamente con SAAM, el primer sistema de la Escuela en adoptarlo, validando en un caso de uso real todos los flujos diseñados: autenticación centralizada, Single Sign-On, cierre de sesión global, inyección del token interno hacia el backend y consultas al directorio de usuarios a través del proxy admin. La integración demostró ser sencilla para el equipo de desarrollo: siguiendo los lineamientos definidos, pudieron autenticar usuarios y consumir la identidad centralizada sin implementar ninguna lógica de autenticación propia, que era precisamente el objetivo del sistema.

Desde el punto de vista académico, el proyecto permitió aplicar en un contexto real un conjunto amplio de conceptos y prácticas vigentes en la industria tecnológica: arquitectura de microservicios, seguridad en profundidad, GitOps, infraestructura como código, protocolos

estándar de autenticación como OIDC y OAuth2, y automatización del ciclo de vida del software. Estos conocimientos, adquiridos e implementados en el marco del trabajo de grado, representan un valor formativo significativo tanto para el autor como para los equipos de estudiantes que en adelante desarrollarán sus proyectos sobre esta infraestructura.

Finalmente, el impacto del proyecto trasciende el entregable técnico: la Escuela cuenta ahora con una plataforma que le permite escalar su ecosistema digital de forma sostenida, con estándares definidos, herramientas compartidas y una base de identidad unificada sobre la cual todos los sistemas futuros pueden construirse con consistencia, seguridad y calidad desde el primer día.

7. Recomendaciones

7.1 Trabajo Futuro

El sistema desarrollado cumple con los objetivos planteados y constituye una base sólida para el ecosistema digital de la Escuela. Sin embargo, a medida que el número de aplicaciones y equipos de desarrollo crezca, surgirán nuevos retos de escalabilidad y operación que vale la pena anticipar.

El primer aspecto a considerar es la evolución de la plataforma de contenedores. Actualmente el laboratorio opera sobre Docker con Docker Compose, lo cual es adecuado para el volumen actual de aplicaciones. Sin embargo, a medida que la cantidad de proyectos crezca, la gestión manual de contenedores en múltiples máquinas se volverá compleja. La adopción de Kubernetes como plataforma de orquestación permitiría gestionar el ciclo de vida de los contenedores de forma automatizada, con capacidades nativas de escalado horizontal, recuperación ante fallos, balanceo de carga y despliegues sin tiempo de inactividad, llevando el

laboratorio a un nivel de madurez operativa comparable al de la industria.

En la misma línea, el ambiente de producción actualmente opera sobre una única máquina virtual, lo que representa un punto único de fallo y un potencial cuello de botella cuando múltiples aplicaciones compiten por los mismos recursos. El enfoque recomendado a futuro es una arquitectura multi-máquina donde cada aplicación o grupo de aplicaciones relacionadas tenga su propia máquina virtual dentro de la red interna de la universidad, con el proxy enrutador desplegado en una máquina dedicada con réplicas para garantizar alta disponibilidad. Este direccionamiento podría gestionarse directamente sobre la red interna universitaria, aprovechando la infraestructura de red existente.

Como mejora específica al Sistema Proveedor de Identidad, se identifica la oportunidad de desarrollar una interfaz de administración de usuarios personalizada que reemplace la consola de administración que trae Keycloak por defecto. Si bien la consola nativa es funcional, su curva de aprendizaje es elevada y no está adaptada al flujo de trabajo específico de la Escuela. Keycloak utiliza internamente React para esta interfaz, lo que implica que su personalización requiere conocimiento del framework y de la arquitectura interna de Keycloak, representando un trabajo de mayor complejidad técnica que se deja como línea de desarrollo futura.

Referencias Bibliográficas

- Arulmozhi, K. (14 de marzo de 2025). Federated Identity vs SSO: What 's the Difference?. Infisign.
- Baquero Giraldo, H. A. (2019). Ley 1581 de 2012: Protección de datos personales en Colombia. Repositorio Institucional Universidad Piloto de Colombia.
- Bhargava, R. (7 de noviembre de 2024). Federated Authentication vs. SSO: What 's the Difference?. Descope.
- Congreso de la República de Colombia. (17 de octubre de 2012). Ley 1581 de 2012. Por la cual se dictan disposiciones generales para la protección de datos personales.
- Congreso de la República de Colombia. (17 de octubre de 2012). Ley Estatutaria 1581 de 2012. Por la cual se dictan disposiciones generales para la protección de datos personales. Departamento Administrativo de la Función Pública.
- Congreso de la República de Colombia. (17 de octubre de 2012). Ley 1581 de 2012. Por la cual se dictan disposiciones generales para la protección de datos personales. Diario Oficial No. 48.587.
- Digital Citizens Alliance. (2017) University Credentials for Sale on Dark Web Study. Digital Citizens Alliance Report.
- EDUCAUSE. (2024). 2024 EDUCAUSE Top 10 IT Issues.
- Fantenberg, J. (19 de septiembre de 2024). Single Sign-on vs. Federated Identity Management: The Complete Guide. Ping Identity.
- FBI Cyber Division. (2022). Private Industry Notifitacion : U.S. Academic Credentials Being Sold on Public and Dark Web Forums. Federal Bureau of Investigation.
- Forrester Research. (2024). The Total Economic Impact of Password Reset Solutions.

Forrester Consulting.

- Fortinet. (s.f.). What is Federated Identity?. Fortinet Cyber Glossary.
- Guidepoint Security. (2024). How Fragmented Application Access is Sabotaging Your Security. F5 Networks Report.
- Fox, P., & Kozyra, J. (2015). eScience and Informatics for international science programs.
- Hardt, D., Parecki, A., & Lodderstedt, T. (2023). OAuth 2.1 Authorization Framework. Internet-Draft draft-ietf-oauth-v2-1-13. Internet Engineering Task Force.
- Hypr & Beyond Identity. (2024). Password Fatigue Survey: Impact on User Behavior and Security
- ISO/IEC. (2022). ISO/IEC 27001:2022 - Information security, cybersecurity and privacy protection — Information security management systems — Requirements [ISO/IEC 27001:2022 specifies the requirements for establishing, implementing, maintaining, and continually improving an information security management system (ISMS).].
- International Organization for Standardization.
- Kleppner, D. (2009). Ensuring the integrity, accessibility, and stewardship of research data in the digital age (1st ed.). National Academies Press.
- Meyer, L., et al. (2023). How effective is multifactor authentication at deterring cyberattacks? arXiv preprint arXiv:2305.00945.
- Ministerio de Educación Nacional (MEN). (30 de noviembre de 2020). Protección de Datos Personales. Modelo Integrado de Planeación y Gestión (MIPG).
- National Institute of Standards and Technologies (NIST). (2020). NIST Special Publication 800-207: Zero Trust Architecture.

- National Institute of Standards and Technologies (NIST). (2024). The NIST Cybersecurity Framework (CSF) 2.0 NIST CSWP 29.
- National Institute of Standards and Technology (NIST). (2025). Digital Identity Guidelines. NIST Special Publication 800-63-4.
- National Institute of Standards and Technologies (NIST). (2025). NIST Special Publication 800-63B-4: Digital Identity Guidelines - Authentication and Authenticator Management.
- Nisenoff, A., et al. (2023). A two-decade retrospective analysis of a university's vulnerability to attacks of exploiting reused passwords. En: 32nd USENIX Security Symposium.
- OpenID Foundation (2023). OpenID Connect Core 1.0 incorporating errata set 2. Open ID Foundation Specification.
- Ordoñez, D. (s.f.). Ley 1581 de 2012. Diana Ordoñez Abogada.
- OWASP Foundation. (2024). Authentication Cheat Sheet.
- Piyoungkorn, K., Chaisawat, S., & Vorakulpipat, C. (2022). Trusted Electronic Contract for Enabling Peer-to-Peer HPC Resource Sharing. Applied sciences, 12(10), 5153-.
- Poma Ramírez, A. D., & Zavaleta Reyes, A. (2023). Implementación de una arquitectura empresarial para mejorar la gestión de identidades y acceso en una empresa de seguros en Lima [Tesis, Universidad Tecnológica del Perú]. Repositorio Institucional UTP.
- Rescorla, E. (2023). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446bis. Internet Engineering Task Force.
- Rodríguez Romero, J. D. (2021). Servicio de autenticación unificado para personas en el marco de los servicios ciudadanos digitales [Tesis, Universidad de los Andes].

Repositorio Institucional Universidad de los Andes

- Sánchez Castillo, V. (2 de diciembre de 2025). Modernización de la Ley 1581 de 2012: aspectos claves del proyecto de ley y desafíos para la protección de datos personales en Colombia. Blog de Derecho de los Negocios, Universidad Externado de Colombia.
- Serrano Mora, M. A. (2023). Diseño y desarrollo de una aplicación web que permita conocer el nivel de madurez en ciberseguridad para entidades pertenecientes al sector educativo [Trabajo de grado, Universidad Industrial de Santander]. Repositorio Institucional.
- Universidad Industrial de Santander. (2025). Escuela de Ingeniería de Sistemas e Informática. UIS. Misión Escuela Ing. de Sistemas e Informática.
- Universidad Nacional Abierta y a Distancia (UNAD). (27 de enero de 2025). Lo que debes saber sobre el tratamiento y protección de datos personales. Noticias UNAD.
- UK Deparment for science, innovation and technology. (2024). Cyber Security Breaches Survey 2024: Educational Sector Report.
- TI Rescue. (7 de julio de 2025). Ley 1581 de 2012. TI Rescue.
- Verizon. (2024). 2024 Data Breach Investigations Report. Verizon Enterprise.
- Vertocci, B. (2021). JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens. RFC 9068. Internet Engineering Task Forceh