

Universidad Tecnológica de La Habana “José Antonio Echeverría”

Facultad de Ingeniería Informática



Aplicación web de código abierto para la gestión y autenticación de usuarios basada en Directorio Activo.

Autor: Carlos Daniel Vilaseca Illnait

Tutores: Dra. C. Raisa Socorro Llanes,
Dra. C. Lisandra Bravo Ilisastigui

La Habana, Marzo 2025

Declaración de autoría

Por la presente, los firmantes declaramos ser los únicos autores de este trabajo desarrollado bajo el patrocinio de Avangenio S.R.L., en virtud del contrato entre el Centro de estudios de tecnologías avanzadas (CETA) y Avangenio S.R.L. Se establece que Avangenio S.R.L. conserva la titularidad exclusiva de todos los derechos de propiedad intelectual sobre los artefactos generados durante esta investigación, incluyendo, sin limitación, código fuente, documentación, diseños, sistemas de software y cualquier otro producto derivado, así como los derechos exclusivos de comercialización y reproducción de los mismos; mientras que a la Universidad Tecnológica de la Habana (CUJAE) se le otorga únicamente una licencia no exclusiva, no transferible y sin fines comerciales para almacenar, publicar y reproducir el presente documento con propósitos académicos y de investigación, sin que esto implique la adquisición de derechos sobre cualquier otro resultado de esta investigación. En fe de lo cual firmamos este documento a 12 de marzo de 2025.

Carlos Daniel Vilaseca Illnait

(Nombre y Apellidos del Diplomante)

Nombre Tutor 1

(Nombres y apellidos del Tutor 1)

Opinión del tutor

Tutores:

Cujae, La Habana, Cuba, Marzo 2025

Resumen

Esta tesis desarrolla una aplicación web de código abierto para la gestión de Directorio Activo a través de LDAP con Samba4 como controlador de dominio, abordando las limitaciones de soluciones existentes como ADWebmanager. La investigación establece una base teórica para la gestión centralizada de usuarios y servicios de directorio, analizando herramientas y tecnologías actuales. La solución propuesta implementa un modelo de dominio completo con arquitectura en capas, utilizando SvelteKit y Idapts para la integración con AD.

La aplicación proporciona funcionalidad robusta del cliente LDAP, interfaz de usuario responsiva y flujos completos de gestión de usuarios. Las características clave incluyen despliegue automatizado mediante CI/CD y contenedores Docker, junto con documentación técnica generada automáticamente. Las metodologías de prueba incluyeron pruebas unitarias y de integración con Vitest, y pruebas E2E con Playwright, validando todas las operaciones críticas.

Los resultados demuestran la implementación exitosa de funcionalidades principales, incluyendo creación, modificación y eliminación de recursos. La solución ofrece mejoras significativas en flexibilidad, seguridad y facilidad de uso en comparación con herramientas existentes. Este trabajo contribuye al campo de la gestión de usuarios y directorios al establecer un marco para futuras investigaciones y desarrollos.

La tesis concluye que la solución propuesta representa una alternativa valiosa para organizaciones que buscan mejorar sus sistemas de gestión de identidades, particularmente en entornos que requieren altos niveles de personalización y seguridad.

Palabras clave: gestión de usuarios, código abierto, Directorio Activo, LDAP, SvelteKit, despliegue automatizado, pruebas.

Abstract

This thesis develops an open-source web application for Active Directory management through LDAP with Samba4 as domain controller, addressing limitations in existing solutions like ADWebmanager. The research establishes a theoretical foundation for centralized user management and directory services, analyzing current tools and technologies. The proposed solution implements a complete domain model with layered architecture, using SvelteKit and Idapts for AD integration.

The application provides robust LDAP client functionality, responsive user interface, and comprehensive user management flows. Key features include automated deployment through CI/CD and Docker containers, along with automatically generated technical documentation. Testing methodologies included unit and integration tests with Vitest, and E2E tests with Playwright, validating all critical operations.

Results demonstrate successful implementation of core functionalities, including resource creation, modification, and deletion. The solution offers significant improvements in flexibility, security, and ease of use compared to existing tools. This work contributes to the field of user and directory management by establishing a framework for future research and development.

The thesis concludes that the proposed solution provides a valuable alternative for organizations seeking to enhance their identity management systems, particularly in environments requiring high levels of customization and security.

Keywords: user management, open source, Active Directory, LDAP, SvelteKit, automated deployment, testing.

Índice

Introducción.....	1
Capítulo 1 Fundamentación teórica	6
1.1 Gestión de usuarios y Directorio Activo	6
1.2 Protocolo Ligero de Acceso a Directorios (LDAP)	8
1.3 Tecnologías y herramientas existentes	10
1.3.1 Frameworks para el desarrollo web	10
1.3.2 Clientes LDAP	13
1.4 Archivos de configuración	17
Capítulo 2 Solución propuesta	23
2.1 Modelado de negocio	23
2.1.1 Modelo de dominio	23
2.1.2 Reglas del negocio por patrón	24
2.2 Requisitos de la aplicación	26
2.2.1 Casos de uso del sistema	26
2.2.2 Requisitos funcionales	27
2.2.3 Requisitos no funcionales	29
2.3 Arquitectura de la aplicación web	30
2.3.1 Stack tecnológico	31
2.3.2 Problemas frecuentes y soluciones en la arquitectura de la aplicación	31
2.4 Selección y modelado de estilos y patrones	33
2.5 Principios de diseño	35
2.6 Patrones de diseño	37
2.7 Implementación	40
2.7.1 Configuración del cliente LDAP seleccionado	40
2.7.2 Desarrollo de mecanismos de autenticación	42
2.7.3 Flujos de gestión	45

2.8 Documentación y despliegue	47
2.8.1 Documentación	47
2.8.2 Despliegue	48
Capítulo 3 Validación de la solución.	50
3.1 Metodología de pruebas	50
3.2 Pruebas	51
Conclusiones	59
Recomendaciones	60
Referencias bibliográficas	61
Siglario	65
Anexos	66

Índice de tablas

1	Algunos atributos del esquema de usuario en el AD	7
2	Comparación entre frameworks de desarrollo web	11
3	Comparación entre distintos clientes LDAP (Marzo 2025)	16
4	Comparación entre distintos estándares de archivos de configuración	21
5	Reglas del negocio	25
6	Requisitos funcionales del sistema	27
7	Requisitos no funcionales del sistema	29
8	Prueba del caso de uso eliminar usuario	51
9	Prueba del caso de uso crear usuario	52
10	Prueba del caso de uso autenticarse	56

Índice de figuras

1	Organización jerárquica de recursos en un AD	7
2	2 Ejemplo de esquema JSON con varias propiedades de distintos tipos	18
3	Ejemplo de archivo JSON aplicando el esquema	19
4	Ejemplo de archivo YAML aplicando el esquema	20
5	Ejemplo de archivo .env	20
6	Modelo de dominio del sistema	24
7	Diagrama de casos de uso del sistema	27
8	Ejemplo del estilo Llamada-Retorno en el sistema	33
9	Uso del cliente ldap para realizar una búsqueda en el directorio	34
10	Estructura simplificada del sistema mostrando n-capas	35
11	Módulos de la capa de logica de negocio	36
12	Instancias del componente Input ilustrando el principio Abierto- Cerrado mediante la composición de componentes	37
13	Declaracion de un store en Sveltekit	38
14	Reacción al cambio en la store	38
15	Uso de la API de Contexto de Sveltekit para inyectar la configuración de la página	39
16	Uso de la API de Contexto de Sveltekit para leer la configuracion de la página	40
17	Creación del cliente LDAP con Idapts	41
18	Diagrama de flujo: Función de autenticación	44
19	Diagrama de flujo: Función cerrar sesión	45
20	Uso de la operacion <i>add</i> con el cliente LDAP	46
21	Uso de la operación <i>search</i> del cliente LDAP para listar usuarios	46
22	Comparación de sintaxis entre YAML y JSON	66
23	Diagrama representando el estilo arquitectónico Llamada Retorno	66
24	Vista colapsada del módulo de encargado de la gestión de usuarios	67

25	Diagrama de sequencia mostrando un flujo de envío de formulario general	68
26	Pruebas con vitest pasaron exitosamente	69
27	Pruebas de e2e pasaron exitosamente	69
28	Manejador: autenticación y creacion del cliente LDAP	70
29	Manejador: Cierre de la conexion del cliente LDAP	70
30	Diagrama de actividades: Eliminar Unidad Organizacional . . .	71
31	Compañías que usan Directorio Activo hoy en día en el mundo (Marzo 2025, 100 000+)	72
32	Diagrama de actividades: Crear usuario	73
33	Diagrama de flujo: Actualizar grupo	75

Introducción

La gestión de usuarios y la autenticación en entornos de red son procesos esenciales en cualquier organización [1, 2, 3]. Los directorios activos desempeñan un papel clave al centralizar y asegurar el control de accesos a los diferentes servicios y recursos empresariales [3]. Actualmente, estas herramientas son ampliamente utilizadas en empresas de diversos tamaños para gestionar usuarios, grupos y unidades organizativas de manera unificada (Figura 31).

Un Directorio Activo (AD por sus siglas en inglés) es una base de datos jerárquica utilizada para almacenar y gestionar información sobre los recursos de la red, como usuarios, dispositivos y servicios. Proporciona una estructura centralizada que permite a los administradores controlar permisos y acceso a los recursos de manera segura y eficiente [4, 5, 6].

En este contexto, el protocolo LDAP (Lightweight Directory Access Protocol) se utiliza para interactuar con los directorios activos, facilitando la búsqueda, consulta y modificación de la información almacenada en ellos [4, 7, 8, 9, 10, 11].

Existen dos grandes grupos de directorios activos en el mercado: los de pago y los de código abierto. Las soluciones comerciales, como las ofrecidas por Microsoft, destacan por su facilidad de uso y alto nivel de integración, pero también generan una dependencia tecnológica significativa y pueden no ser viables para organizaciones con presupuestos limitados. En contraste, los directorios activos de código abierto que eliminan la necesidad de costosas licencias y ofrecen independencia tecnológica [1, 8, 12].

Entre las soluciones libres, Samba4 emerge como una solución open-source que permite implementar un controlador de dominio de AD en infraestructura Linux, ofreciendo compatibilidad con los protocolos de Microsoft AD.

Un ejemplo práctico de adopción de Samba4 se encuentra en la empresa

Avangenio. Sin embargo, las herramientas de administración incluidas con Samba4 están limitadas a la línea de comandos, lo que las hace menos accesibles en contextos de administración de redes empresariales [8, 12, 13].

Para superar esta limitación, se desarrollaron soluciones web como ADwebmanager [14], un fork del proyecto Samba4-manager [15] que actualiza el código fuente a Python 3. Aunque ADwebmanager representó un avance en la accesibilidad comparado con la gestión a través de la línea de comandos, presentó varias limitaciones que afectaron su escalabilidad y usabilidad:

- Interfaz no responsiva, dificultando su uso en dispositivos móviles.
- Requiere recargas completas de la página para actualizar la interfaz.
- Frontend basado en templates con elementos dinámicos difíciles de modificar.
- Vulnerabilidad a ataques de fuerza bruta en la página de autenticación.
- Inconsistencias en la interfaz de usuario que afectan la experiencia de usuario.

Estas limitaciones llevan a identificar como problema a resolver: las deficiencias de ADWebmanager en términos de seguridad y experiencia de usuario dificultan la gestión eficiente de los recursos empresariales.

Para solucionar este problema se tiene como objeto de estudio los las herramientas de administración de AD. El campo de acción se delimita a los sistemas web de SLCA para la gestión de AD, con especial atención a las soluciones web para Samba4.

Como hipótesis se plantea que desarrollar una herramienta web de código abierto para la gestión de AD basada en Samba4 que ofrezca un mayor nivel de personalización a través de archivos de configuración permitirá mejorar la adaptabilidad y facilidad de uso, sin comprometer la seguridad y estabilidad, en comparación con ADWebmanager.

Para demostrar esta hipótesis se plantea como objetivo general crear una consola de administración de código abierto para AD que demuestre una mejora en la personalización y facilidad de uso en comparación con las herramientas de gestión existentes.

A partir de este objetivo general, se derivan los siguientes objetivos específicos y tareas:

1. Analizar los requisitos de la aplicación y la personalización del sistema:
 - 1.1. Documentar los requisitos funcionales y no funcionales que debe cumplir la aplicación.
 - 1.2. Analizar diferentes casos de uso para identificar las opciones de personalización.
 - 1.3. Documentar los requisitos de personalización, incluyendo la interfaz de usuario y ajustes de seguridad.
2. Seleccionar tecnologías adecuadas:
 - 2.1. Evaluar diferentes clientes LDAP disponibles, considerando factores como compatibilidad, rendimiento y facilidad de integración.
 - 2.2. Elegir el cliente LDAP que mejor se alinee con los requisitos funcionales y no funcionales previamente definidos.
3. Implementar la arquitectura y funciones básicas de la aplicación:
 - 3.1. Establecer la arquitectura base del proyecto y configurar el ambiente de desarrollo necesario.
 - 3.2. Configurar el cliente LDAP seleccionado y las herramientas asociadas para iniciar el desarrollo.
 - 3.3. Desarrollar mecanismos de autenticación que interactúen con el AD utilizando el cliente LDAP seleccionado.
 - 3.4. Implementar funcionalidades críticas para la gestión de usuarios y

grupos utilizando el cliente LDAP, incluyendo operaciones de lectura, eliminación y actualización.

4. Realizar pruebas para asegurar el correcto funcionamiento del sistema:

4.1. Diseñar y ejecutar pruebas de integración que verifiquen el correcto funcionamiento del sistema en su conjunto, desde la autenticación hasta la gestión de recursos.

4.2. Documentar los resultados de las pruebas y realizar los ajustes necesarios basados en los hallazgos.

5. Generar documentación técnica:

5.1. Automatizar generación de documentación técnica utilizando los esquemas JSON existentes para describir la estructura y validación de los archivos de configuración.

5.2. Integrar el proceso de generación de documentación en un pipeline de CI/CD, asegurando la actualización automática con cada cambio en los esquemas.

5.3. Publicar la documentación como un sitio web estático accesible públicamente, proporcionando una referencia técnica completa y actualizada.

6. Desplegar aplicación:

6.1. Configurar entornos de desarrollo, pruebas y producción utilizando Docker, garantizando consistencia entre diferentes entornos.

6.2. Implementar un pipeline de CI/CD que automatice la construcción, pruebas y despliegue de la aplicación.

Como valor práctico con la realización de este trabajo se obtuvo un diseño de software de una herramienta para la gestión de AD que es personalizable y fácil de desplegar. La herramienta permite a las organizaciones gestionar usuarios, grupos y unidades organizativas de manera centralizada,

personalizar la configuración del AD mediante archivos JSON/YAML para adaptarse a diferentes políticas organizacionales, realizar tareas comunes de administración a través de una interfaz web intuitiva, reducir costos al eliminar la dependencia de soluciones comerciales propietarias, proporcionando así una alternativa flexible y accesible especialmente útil para organizaciones con recursos limitados o necesidades específicas.

La estructura del informe se organiza de la siguiente manera:

Capítulo 1 Fundamentación teórica. En este capítulo se presenta una explicación teórica sobre los conceptos y tecnologías fundamentales para la gestión de usuarios y AD. Se analizarán en profundidad temas como la importancia de la gestión de usuarios en entornos digitales, los principios de seguridad y autenticación, y el funcionamiento de los directorios activos y LDAP. También se abordarán las diferentes herramientas existentes para la gestión de AD, sus ventajas y limitaciones, y se establecerá el marco teórico que sustenta la propuesta de solución planteada en este trabajo.

Capítulo 2 Descripción de la propuesta de solución. Este capítulo aporta una explicación detallada sobre la propuesta de solución a los problemas identificados en la gestión de AD. Se describirá la arquitectura de la aplicación web de código abierto propuesta, sus funcionalidades principales, y cómo se abordarán los requisitos de personalización y simplicidad de despliegue. Además, se explicarán las decisiones de diseño y las tecnologías seleccionadas (SvelteKit y Idapts) para el desarrollo de la herramienta, así como los procesos de documentación y despliegue automatizado.

Capítulo 3 Validación de la propuesta de solución. Este capítulo está dedicado a la validación de la propuesta de solución mediante la realización de pruebas exhaustivas. Se diseñarán y ejecutarán pruebas de integración para verificar el correcto funcionamiento del sistema en su conjunto, desde la autenticación hasta la gestión de recursos. Los resultados de estas pruebas se documentarán y se realizarán los ajustes necesarios basados en los hallazgos.

Capítulo 1 Fundamentación teórica

Este capítulo establece los fundamentos teóricos para la gestión de usuarios y AD. Cubre: principios de gestión de usuarios, conceptos de AD y LDAP, y análisis de herramientas existentes. El objetivo es proporcionar una base para desarrollar una solución personalizable de gestión de directorios.

1.1. Gestión de usuarios y Directorio Activo

En la administración de sistemas informáticos, la gestión de usuarios es un componente esencial para asegurar el correcto funcionamiento y la seguridad de la infraestructura tecnológica de una organización. El AD es una herramienta crucial en este ámbito, ya que permite la centralización y automatización de la gestión de usuarios, dispositivos y recursos. Este epígrafe aborda los conceptos fundamentales de la gestión de usuarios, las ventajas y características del AD, así como su implementación y administración [1, 2]. Un AD es una base de datos central que se utiliza para administrar y organizar los recursos de una red de computadoras. La gestión de usuarios es una función importante en un AD, ya que permite crear, eliminar y editar los perfiles de los usuarios en un dominio. Además, también implica la asignación de permisos y roles, que determinan el nivel de acceso y las acciones que cada usuario puede realizar en la red [8, 12, 16].

En la Figura 1 a continuación se muestra como se organizan jerárquicamente los recursos dentro de un AD.

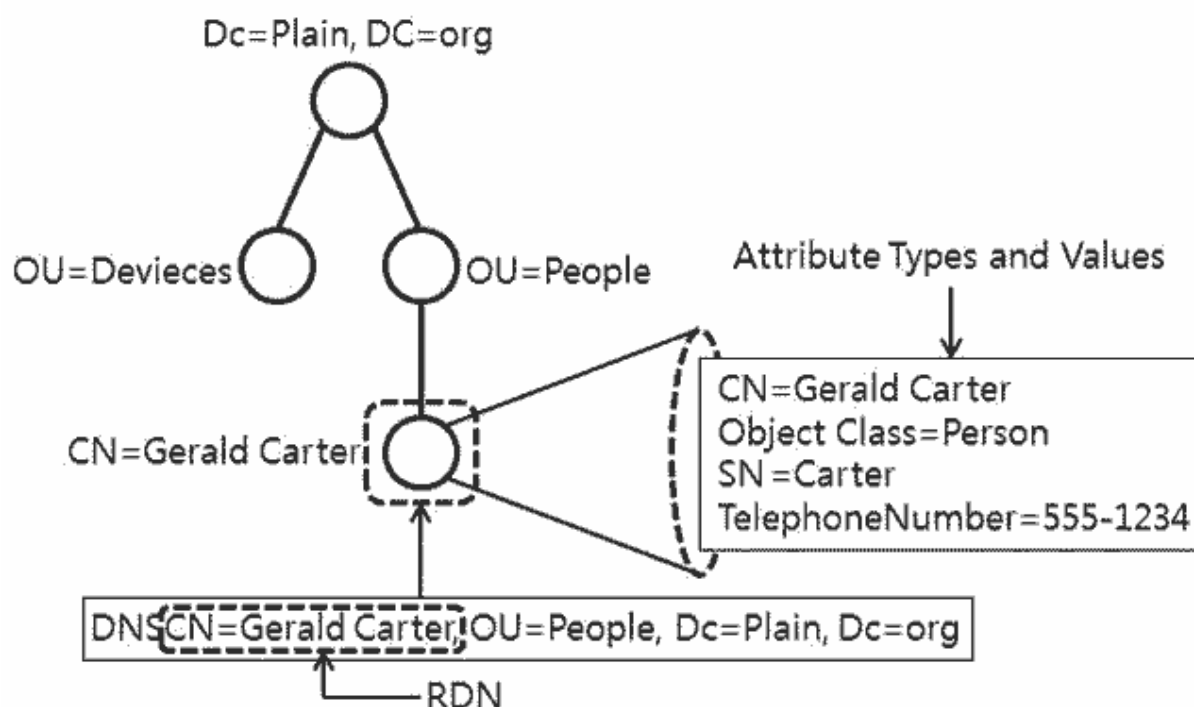


Figura 1: Organización jerárquica de recursos en un AD

El esquema de AD contiene definiciones formales de cada clase de objeto que se puede crear en un directorio. Por ejemplo, en el caso de los usuarios, el esquema define atributos como el nombre, apellido, nombre para mostrar, nombre de inicio de sesión, dirección de correo electrónico, número de teléfono [4, 9, 17]. En la Tabla 1 se muestran algunos atributos del esquema de usuario de un AD, así como ejemplos de valores que pueden tomar estos atributos [18].

Tabla 1: Algunos atributos del esquema de usuario en el AD

Atributo	Descripción	Ejemplo
displayName	Nombre para mostrar del usuario, puede ser diferente al nombre real	Juan Pérez

distinguishedName	Nombre único del objeto en el directorio, incluye la ruta completa	CN=jperez, OU=Dirección, DC=ejemplo, DC=com
givenName	Nombre del usuario	Juan
mail	Dirección de correo electrónico del usuario	juan.perez@ejemplo.com
objectSid	Identificador de seguridad (SID) del usuario	S-1-5-21-1234567890-1234567890-1234567890-1234
sAMAccountName	Nombre de inicio de sesión compatible con versiones anteriores de Windows	jperez
sn	Apellido del usuario	Pérez
telephoneNumber	Número de teléfono del usuario	+34 123 456 789
userPrincipalName	Nombre de inicio de sesión del usuario	jperez@ejemplo.com

1.2. Protocolo Ligero de Acceso a Directorios (LDAP)

LDAP es un estándar de protocolo utilizado para acceder y gestionar información almacenada en directorios de manera eficiente y segura. En el contexto de la administración de sistemas informáticos, LDAP desempeña un papel fundamental al facilitar la búsqueda, autenticación y gestión de usuarios, dispositivos y otros recursos [6, 7].

Este epígrafe explora los principios fundamentales de LDAP, incluyendo su arquitectura, funcionamiento y principales características. Se examinará cómo LDAP permite la estructuración jerárquica de la información mediante la utilización de entradas y atributos, lo cual facilita la organización y el acceso

a datos.

Arquitectura de LDAP

LDAP se basa en una arquitectura cliente-servidor, donde el cliente LDAP envía solicitudes al servidor LDAP para realizar diversas operaciones, como búsquedas, actualizaciones y autenticaciones de información almacenada en el directorio. Esta arquitectura facilita la gestión centralizada y eficiente de los datos [4, 6, 7].

- **Cliente LDAP:** Es el software que realiza peticiones de búsqueda, modificación o consulta de información almacenada en el servidor LDAP.
- **Servidor LDAP:** Es el software que almacena la base de datos de directorio y responde a las peticiones de los clientes LDAP. El servidor LDAP gestiona y organiza la información en forma de entradas almacenadas en un árbol de directorio.
- **Protocolo de Comunicación:** LDAP define cómo se comunican el cliente y el servidor a través de un protocolo eficiente y ligero, diseñado principalmente para la lectura, búsqueda y modificación de información en directorios.
- **Eficiencia:** permite búsquedas rápidas y eficientes de información en grandes volúmenes de datos.
- **Seguridad:** soporta protocolos de seguridad como TLS (Transport Layer Security) para proteger la integridad y confidencialidad de los datos transmitidos.
- **Escalabilidad:** capacidad para manejar grandes cantidades de datos y usuarios dentro de un directorio, adaptándose a las necesidades de crecimiento de una organización.
- **Interoperabilidad:** estándar abierto compatible con una amplia gama de plataformas y sistemas de directorio.

Aplicaciones de LDAP

LDAP se utiliza ampliamente en la autenticación de usuarios, control de acceso y gestión de identidades en sistemas operativos, aplicaciones web y servicios de correo electrónico. Su flexibilidad y robustez lo convierten en una herramienta fundamental para la integración y administración de infraestructuras de TI empresariales [6, 7, 10].

1.3. Tecnologías y herramientas existentes

En el ámbito de la gestión de AD, existen diversas tecnologías y herramientas diseñadas para facilitar esta tarea esencial en la administración de sistemas informáticos. Estas herramientas no solo permiten una gestión más eficiente de los recursos y usuarios dentro de una organización, sino que también contribuyen a mejorar la seguridad y el control de acceso.

Este epígrafe se centrará en revisar las tecnologías clave para la implementación de soluciones de gestión de AD, incluyendo frameworks de desarrollo web y clientes LDAP. Se analizarán las ventajas y limitaciones de estas soluciones.

1.3.1. Frameworks para el desarrollo web

En el ámbito del desarrollo web, los frameworks juegan un papel crucial al proporcionar estructuras y herramientas que simplifican y aceleran la creación de aplicaciones. Estos frameworks ofrecen una base sólida para la implementación de funcionalidades complejas, facilitando la interacción entre el diseño front-end y la lógica de negocio back-end. Esta sección explora diversos frameworks destacados en el panorama actual, analizando sus características, ventajas y aplicaciones específicas en el desarrollo web moderno.

La Tabla 2 presenta una comparación detallada de tres frameworks populares en el desarrollo web: SvelteKit, Next.js y Nuxt.js. Esta comparación se enfoca en varias características clave que influyen en la elección de un framework para proyectos web, tales como eficiencia, flexibilidad, escalabilidad, curva de aprendizaje y comunidad. Al analizar estos aspectos, se puede obtener una visión más clara de las fortalezas y debilidades de cada framework, ayudando a los desarrolladores a tomar decisiones informadas al seleccionar la herramienta más adecuada para sus necesidades específicas.

Tabla 2: Comparación entre frameworks de desarrollo web

Característica	Sveltekit	Next.js	Nuxt.js
Eficiencia	Alto rendimiento con compilación previa y sitios estáticos	Buena eficiencia con renderizado del lado del servidor y en el cliente	Eficiencia en el desarrollo de aplicaciones web, con facilidades para la creación de aplicaciones universal y estáticamente generadas
Flexibilidad	Gran flexibilidad en personalización y configuración	Flexible para la creación de diferentes tipos de aplicaciones web	Flexible y adaptable a diferentes tipos de proyectos, con un enfoque en la simplicidad y facilidad de uso
Escalabilidad	Altamente escalable para proyectos de diferentes tamaños	Buena capacidad de escalar y manejar proyectos de gran envergadura	Puede escalar adecuadamente para manejar proyectos de diversos tamaños y complejidades

Curva de aprendizaje	Muy baja, con sintaxis simple y familiar	Moderada, requiere familiarizarse con sus conceptos y funcionalidades	Moderada, especialmente para aquellos que están familiarizados con Vue.js
Comunidad	En crecimiento, con soporte activo y recursos disponibles	Amplia comunidad de desarrolladores, con gran cantidad de recursos y soporte en línea	Comunidad activa y en crecimiento, con soporte y recursos disponibles

Es importante destacar que realizar una comparación exhaustiva de todos los frameworks disponibles es complicado por varias razones:

- Abundancia de opciones: Actualmente, existen cientos de frameworks de desarrollo web, cada uno con características y ventajas únicas.
- Complejidad inherente: Los frameworks son complejos y ofrecen una amplia gama de características, lo que dificulta una comparación exhaustiva y detallada.
- Variabilidad en los requisitos: Las aplicaciones web tienen requisitos diversos, lo que significa que un framework ideal para una aplicación puede no serlo para otra.

Por lo tanto, es difícil determinar que un framework sea superior a otro de manera generalizada. La elección del mejor framework para una aplicación específica depende en gran medida de los requisitos particulares de cada aplicación y de las preferencias del equipo de desarrollo. En última instancia, la selección de un framework adecuado se basa en su capacidad para resolver los problemas específicos de desarrollo que se presentan en el contexto de la aplicación deseada.

1.3.2. Clientes LDAP

LDAP se ha consolidado como un estándar esencial en la administración de AD. Su capacidad para gestionar y acceder a información jerárquica de manera eficiente ha hecho que múltiples aplicaciones y servicios adopten clientes LDAP para interactuar con los directorios.

En este epígrafe, se proporciona una definición de cliente LDAP basada en la literatura, se exponen sus características principales, se explica la importancia del uso de estos clientes y se exploran diversos clientes LDAP disponibles, analizando sus características, ventajas y limitaciones.

¿Qué es un cliente LDAP?

Un cliente LDAP es una aplicación o herramienta que permite a los usuarios y sistemas interactuar con un servidor de AD. Su función principal es facilitar la comunicación con el directorio, permitiendo que se realicen operaciones como búsquedas, modificaciones, adiciones y eliminaciones de entradas en la base de datos del directorio. Estos clientes actúan como intermediarios que traducen las solicitudes de los usuarios o aplicaciones a un formato comprensible para el directorio, simplificando la interacción y mejorando la eficiencia en la administración de datos.

Capacidades de los clientes LDAP

- **Búsquedas:** Los clientes LDAP pueden realizar búsquedas en el directorio para encontrar información específica basada en varios criterios. Esto es crucial para aplicaciones que necesitan recuperar datos de manera rápida y eficiente.
- **Modificaciones:** Permiten actualizar la información existente en el directorio. Las modificaciones pueden incluir cambios en atributos de una entrada o actualizaciones de múltiples entradas simultáneamente.
- **Adiciones:** Los clientes LDAP facilitan la adición de nuevas entradas en el directorio. Esto es útil para la incorporación de nuevos usuarios,

dispositivos o cualquier otra entidad que necesite ser gestionada dentro del directorio.

- **Eliminaciones:** También soportan la eliminación de entradas del directorio, ayudando a mantener la información actualizada y eliminando datos obsoletos o incorrectos.

Abstracción de la lógica

La abstracción de la lógica en la interacción con el directorio mediante un cliente LDAP es fundamental por varias razones:

- **Simplicidad y Eficiencia:** Al utilizar un cliente LDAP, los desarrolladores y administradores no necesitan conocer los detalles específicos del protocolo LDAP. Esto simplifica el desarrollo y la administración, permitiendo centrarse en la lógica de negocio en lugar de en los detalles técnicos.
- **Interoperabilidad:** Los clientes LDAP son compatibles con múltiples sistemas y aplicaciones, lo que facilita la integración de diversas soluciones en una infraestructura común. Esto es crucial para la interoperabilidad entre sistemas heterogéneos.
- **Seguridad:** Al centralizar las peticiones a través de un cliente LDAP, es posible implementar políticas de seguridad consistentes, como autenticación y autorización, garantizando que solo usuarios y aplicaciones autorizadas puedan acceder y modificar la información del directorio.

Comparación entre clientes LDAP existentes

La elección del cliente LDAP adecuado es crucial para gestionar eficientemente un AD. Con la creciente variedad de opciones disponibles, es fundamental entender las diferencias y capacidades de cada cliente LDAP. En la Tabla 3 se comparan varios clientes LDAP, evaluando aspectos clave que pueden influir en la elección de una solución:

- **Dependencia de Idapjs:** Indica si el cliente LDAP tiene dependencia de la biblioteca Idapjs. Esto es relevante debido a la discontinuación de Idapjs, que hasta el 14 de Mayo de 2024 era ampliamente utilizada. Esta evaluación asegura la selección de soluciones que ofrecen una base estable y sostenible, minimizando riesgos asociados con la obsolescencia y garantizando la compatibilidad a largo plazo con el ecosistema LDAP.
- **Soporte TypeScript:** La compatibilidad con TypeScript no solo permite el desarrollo más seguro y estructurado de aplicaciones modernas, sino que también mejora significativamente la detección de errores en tiempo de desarrollo.
- **Calidad de la documentación:** La disponibilidad y calidad de la documentación afecta directamente la rapidez con la que los desarrolladores pueden familiarizarse con el cliente LDAP y resolver problemas.
- **Comunidad:** Una comunidad activa y un soporte sólido aseguran que los problemas se resuelvan rápidamente y que el cliente LDAP se mantenga actualizado con las mejores prácticas. El nivel de actividad de la comunidad y la frecuencia de las actualizaciones pueden variar, impactando la estabilidad y la confianza en el uso a largo plazo del cliente.
- **Facilidad de uso:** Evalúa la simplicidad y la curva de aprendizaje del cliente LDAP. Una alta facilidad de uso reduce el tiempo de integración y minimiza errores durante la implementación. Esto incluye si el cliente provee abstracciones y/o funciones de alto nivel, lo cual puede facilitar significativamente su uso.

Tabla 3: Comparación entre distintos clientes
LDAP (Marzo 2025)

Cliente LDAP	Dependencia de Idapjs	Soporta TypeScript	Calidad de la documentación	Comunidad y soporte (en julio 2024)	Facilidad de uso
Idapts	No	Sí	Excelente, documentación completa y fácil de entender, con ejemplos detallados.	Activa y sólida, última publicación en Enero 2025 y más de 90 mil descargas semanales.	Alta, fácil de usar y aprender, con una curva de aprendizaje baja. Provee abstracciones para la composición de filtros
Idap-client	No	No	Buena, documentación adecuada, pero básica. Cubre la mayoría de los casos de uso comunes, aunque carece de ejemplos avanzados.	Baja, última publicación 2016, con 20 descargas semanales.	Media, requiere algún tiempo de aprendizaje, pero es manejable. No provee abstracciones para la composición de filtros
activedirectory	Sí	No	Buena, documentación adecuada, con suficiente información para la configuración y uso básico, aunque podría mejorar en detalle y ejemplos.	Moderada, última publicación 2016, con 15 mil descargas semanales	Media, interfaz familiar, provee funciones de más alto nivel específicas para la búsqueda de usuarios y grupos.
Idap-ts-client	Sí	Sí	Escasa, documentación con poca información disponible y pocos ejemplos.	Baja, última publicación 2022, con 51 descargas semanales	Baja, debido a la falta de documentacion.

1.4. Archivos de configuración

Los archivos de configuración desempeñan un papel fundamental en el desarrollo y la operación de sistemas informáticos modernos al proporcionar una forma estructurada de definir variables y ajustes clave que modifican y parametrizan el comportamiento de los sistemas. Además, facilitan la modificación del sistema sin necesidad de acceder y modificar directamente el código fuente, lo que promueve la flexibilidad y la mantenibilidad.

Este epígrafe explora diversas técnicas y formatos utilizados para la configuración de aplicaciones, destacando su importancia en la gestión eficiente de la infraestructura y la personalización de comportamientos. Se analiza el uso de archivos de variables de entorno (.env) [19], así como de JSON [20, 21] y YAML [22, 23] en combinación con JSON Schema [24, 25] para la validación y estructuración de configuraciones. Aunque existen otros formatos como TOML [20], estos no tienen soporte para JSON Schema, lo cual carece de la capa adicional de seguridad y estructuración que es fundamental en muchos entornos de desarrollo. Este epígrafe proporciona una visión comprehensiva para comprender cómo estos archivos facilitan la configuración flexible y robusta.

JSON Schema

JSON Schema o Esquema JSON es un estándar para la definición y validación de la estructura de documentos en distintos formatos. Proporciona un marco para especificar las propiedades requeridas, los tipos de datos, las restricciones de valores y otras reglas que los datos deben cumplir. JSON Schema no es un archivo de configuración por sí mismo, sino más bien una interfaz para asegurar que los archivos de configuración cumplan con las especificaciones esperadas [24, 25].

El uso de JSON Schema mejora la seguridad y la robustez de los sistemas al validar automáticamente la configuración antes de su aplicación, evitando errores y asegurando la conformidad con los requisitos del sistema [24, 25].

En la Figura 2 se muestra un ejemplo de JSON Schema donde se definen 4 propiedades simples de distintos tipos y distintas restricciones. Por ejemplo prop1 se define como una cadena de caracteres, prop2 como un entero con valor máximo 100 y valor mínimo 0, prop3 un booleano, y prop4 como un arreglo que debe tener elementos únicos y un máximo de 3 elementos. A cada una de estas propiedades se les puede proveer de una descripción lo cual ayuda en la comprensión del propósito de la propiedad.

```
{
  "$id": "test.schema.json",
  "description": "Example json schema",
  "title": "ExampleConfig",
  "type": "object",
  "properties": {
    "prop1": {
      "type": "string",
      "default": "default value for prop1",
      "description": "This is the first property for ExampleConfig @default \"default value for prop1\""
    },
    "prop2": {
      "type": "integer",
      "default": 10,
      "minimum": 0,
      "maximum": 100,
      "description": "This is the second property for ExampleConfig @default 10"
    },
    "prop3": {
      "type": "boolean",
      "default": true,
      "description": "This is the third property for ExampleConfig @default true"
    },
    "prop4": {
      "type": "array",
      "maxItems": 3,
      "uniqueItems": true,
      "default": [1, 2, 3],
      "description": "This is the fourth property for ExampleConfig @default [1,2,3]"
    }
  }
}
```

Figura 2: 2 Ejemplo de esquema JSON con varias propiedades de distintos tipos

JSON + JSON Schema

Los archivos JSON son ampliamente utilizados para la configuración de aplicaciones debido a su simplicidad y compatibilidad con muchas herramientas y lenguajes de programación [20]. Cuando se combinan con JSON Schema, estos archivos pueden ser validados para asegurar que cumplen con las especificaciones esperadas. Esto permite definir

configuraciones de manera clara y estructurada, garantizando que los datos sean consistentes y conformes a los requisitos del sistema [21, 24].

En la Figura 3 se muestra un ejemplo de archivo JSON aplicando el esquema definido en la Figura 2. Se puede ver como establecer el esquema dota al archivo JSON de autocompletado en las propiedades, además de validaciones según las restricciones del esquema.

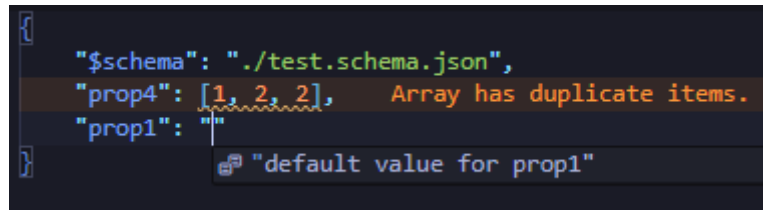
A screenshot of a code editor showing a JSON object. The first line is {"\$schema": "./test.schema.json",. The second line is "prop4": [1, 2, 2], with a red squiggly line under the second '2' and a tooltip that says "Array has duplicate items.". The third line is "prop1": "", with a tooltip that says "default value for prop1". The editor has a dark background and syntax highlighting.

Figura 3: Ejemplo de archivo JSON aplicando el esquema

YAML + JSON Schema

YAML es un formato de serialización de datos más legible que JSON (Figura 22) y ampliamente utilizado en la configuración de aplicaciones. Su sintaxis limpia y sencilla lo hace ideal para archivos de configuración [22, 23]. Al igual que JSON, los archivos YAML pueden ser validados utilizando JSON Schema, proporcionando una capa adicional de seguridad y estructuración. Esto combina la legibilidad de YAML con la robustez de la validación de JSON Schema, haciendo que las configuraciones sean tanto claras como seguras [24].

En la Figura 4 se muestra un ejemplo de archivo YAML aplicando el esquema definido en la Figura 2. Se puede ver como establecer el esquema dota al archivo YAML de autocompletado en las propiedades, además de validaciones dependiendo de las restricciones del esquema.

```

ExampleConfig - Example json schema (test.schema.json)
# yaml-language-server: $schema=./test.schema.json
prop3: true
prop1: default value for prop1
prop4:
  - 1      Array has duplicate items.
  - 2
  - 2

```

☐ faf
☐ rcon
☒ prop2

Figura 4: Ejemplo de archivo YAML aplicando el esquema

Variables de entorno (.env)

Las variables de entorno son una técnica comúnmente utilizada para configurar aplicaciones. Estos archivos, generalmente con la extensión .env, permiten definir variables clave en un formato sencillo de clave-valor (Figura 5), facilitando la gestión de configuraciones sensibles y específicas del entorno, como credenciales de acceso, URLs de servicios externos, y configuraciones de depuración [19].

```

1  ORIGIN="http://localhost:5173"
2  PUBLIC_BASE_DN="DC=local,DC=com"
3  PUBLIC_LDAP_DOMAIN="local.com"
4  SAMBA_DC_REALM=${PUBLIC_LDAP_DOMAIN}
5  # openssl rand --hex 32
6  SECRET_KEY=8e558d3d1138af3425058cb690bdc8916bda42ff283db39500cfb0cb452b4ba
7  # A path to the configuration file. If a URL is provided, it must use the file: protocol.
8  CONFIG_PATH="app.config.dev.json"

```

Figura 5: Ejemplo de archivo .env

En la Tabla 4 se examinan aspectos como la facilidad de implementación, el soporte de tipos de datos, la capacidad de validación, la popularidad en diversos contextos de desarrollo, la complejidad de la sintaxis y el soporte nativo en el lenguaje Javascript. Esta información permitirá una comprensión detallada de cuándo y cómo utilizar cada formato para mejorar la configuración y la gestión de aplicaciones [19, 26, 27].

Tabla 4: Comparación entre distintos estándares de archivos de configuración

Característica	.env	JSON + JSON schema	YAML + JSON schema
Facilidad de uso	Alta: Requiere mínimo conocimiento técnico.	Media: Estructurado pero con una sintaxis muy estricta.	Alta: Fácil de usar con herramientas y bibliotecas bien soportadas.
Soporte de tipos	Ninguno, todo son cadenas de texto	Completo: Admite tipos como cadenas, números, booleanos, arrays, objetos, y más	Completo: Soporta varios tipos de datos, incluidos mapas y listas.
Validación de datos	No: No tiene capacidad de validación	Sí: Puede validar datos contra un esquema definido	No: No tiene capacidad de validación intrínseca
Complejidad de sintaxis	Muy Baja: Sintaxis muy simple y directa	Media: Sintaxis detallada y estructurada	Baja: Sintaxis simple y directa
Uso en aplicaciones	Variables de entorno: Ideal para configuraciones sensibles al entorno	Definición y validación de datos: Ideal para garantizar que los datos sean consistentes y conformes a las especificaciones	Configuración, automatización: Utilizado para definir configuraciones complejas y scripts de automatización
Popularidad	Alta: Ampliamente utilizado en aplicaciones web y de software	Alta: Ampliamente utilizado para validación de datos y definición de esquemas	Alta: Popular en DevOps y administración de sistemas
Soporte nativo	Sí	Sí	No, se necesitan dependencias extra para parsearlo primero.

Conclusiones parciales

Este capítulo ha establecido los fundamentos teóricos esenciales para la gestión de usuarios y directorios activos. Se ha analizado la importancia de la gestión centralizada de usuarios mediante AD, destacando su estructura jerárquica y esquema de atributos. El protocolo LDAP se ha presentado como un estándar clave para la interacción eficiente y segura con directorios. Se han evaluado diversas herramientas de gestión de AD y clientes LDAP, considerando factores como personalización, facilidad de uso y soporte técnico. Finalmente, se han explorado diferentes enfoques para la configuración de sistemas, destacando la utilidad de formatos como JSON, YAML y variables de entorno, complementados con JSON Schema para validación. Estos conceptos proporcionan la base teórica necesaria para el desarrollo de la solución de gestión de usuarios y directorios activos.

Capítulo 2 Solución propuesta

Este capítulo presenta la solución propuesta para la gestión de AD, cubriendo: modelado del negocio, requisitos, arquitectura, patrones de diseño, implementación, despliegue y documentación. El objetivo es desarrollar una aplicación web flexible para la administración de directorios activos, con procesos automatizados de despliegue.

2.1. Modelado de negocio

En este apartado se presentan tanto el modelo de dominio como las reglas del negocio organizadas por patrones. El modelo de dominio define las principales entidades y relaciones dentro del sistema, tales como usuarios, grupos, unidades organizacionales, proporcionando una representación clara del entorno de AD. Por su parte, las reglas del negocio describen los comportamientos y restricciones que deben cumplirse durante la interacción con el sistema, agrupadas según patrones que facilitan su comprensión y posterior implementación.

2.1.1. Modelo de dominio

El modelo de dominio representado en la Figura 6 ilustra las principales entidades y relaciones involucradas en la gestión de un Directorio Activo, que es la base del sistema propuesto. Este modelo permite comprender las interacciones y dependencias entre los diferentes actores y objetos del sistema.

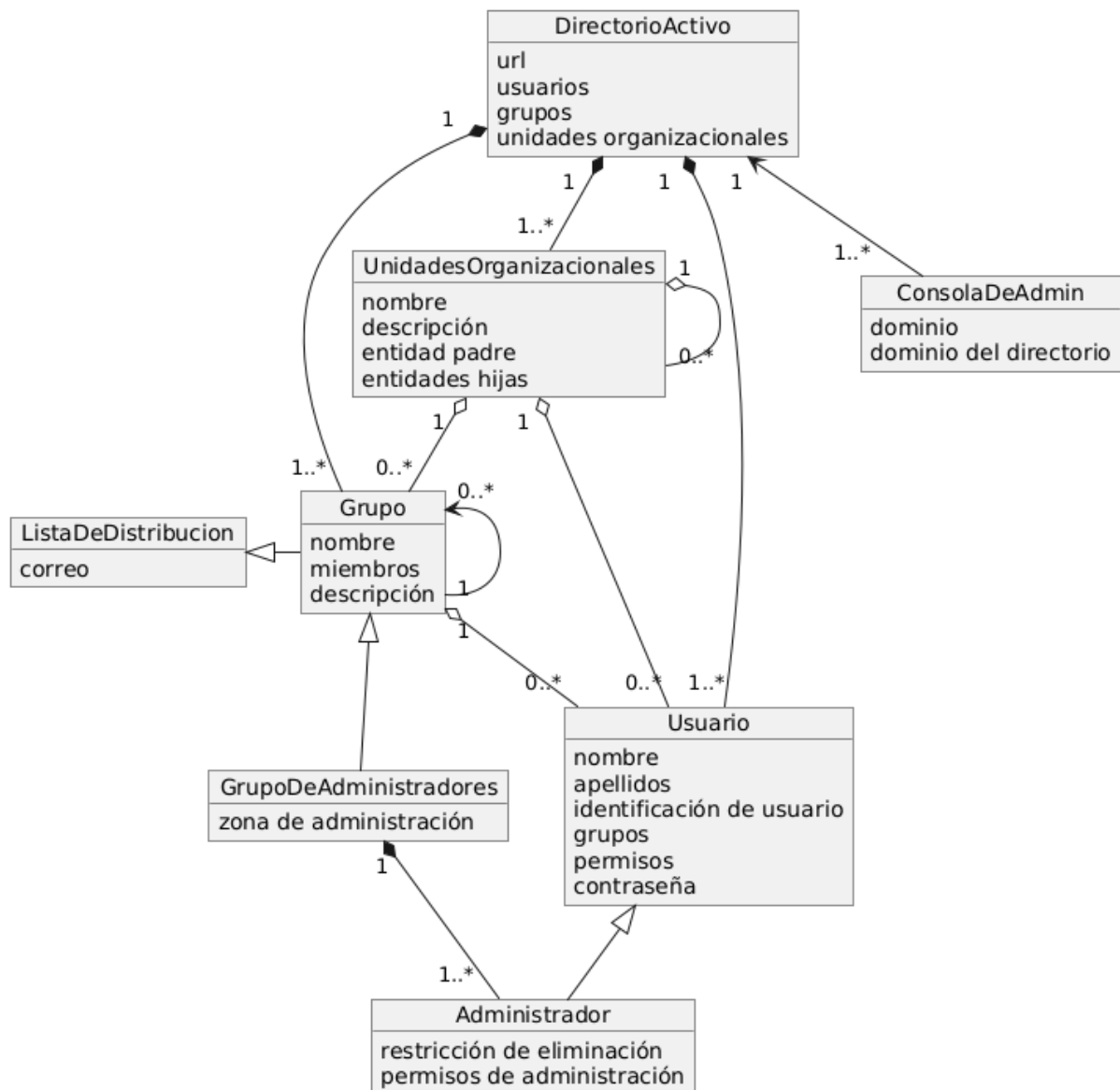


Figura 6: Modelo de dominio del sistema

2.1.2. Reglas del negocio por patrón

En el desarrollo de sistemas de información, las reglas de negocio son cruciales, ya que establecen las políticas, restricciones y condiciones que guían el funcionamiento de una organización. Estas reglas resultan fundamentales para asegurar la integridad, consistencia y eficiencia de los procesos en el negocio.

La Tabla 5 muestra un resumen de las reglas de negocio identificadas hasta ahora, junto con el patrón que las representa. Esta clasificación facilita la implementación eficiente de dichas reglas en el sistema, garantizando el cumplimiento de los requisitos y expectativas del negocio.

Tabla 5: Reglas del negocio

Regla	Patrón
Solo si el usuario no existe se puede crear.	Precondición
Solo si el usuario existe se puede eliminar.	Precondición
Solo si el usuario existe se puede editar.	Precondición
Solo si el grupo no existe se puede crear.	Precondición
Solo si el grupo existe se puede eliminar.	Precondición
Solo si el grupo existe se puede editar.	Precondición
Un grupo de tipo lista de distribución tiene: miembros, nombre, descripción y correo electrónico.	Estructura
Un grupo tiene: miembros, nombre, descripción.	Estructura
Una unidad organizacional tiene: nombre y descripción	Estructura
El administrador es responsable de la creación, edición, y eliminación de los usuarios, grupos y unidades organizacionales.	Responsabilidad
El usuario debe estar autenticado para acceder a las funciones del sistema.	Precondición
Solo si no se ha alcanzado el límite de usuarios, el administrador puede crear más usuarios.	Precondicion
Solo si no se ha alcanzado el límite de grupos, el administrador puede crear más grupos.	Precondición
Solo si no se ha alcanzado el límite de unidades organizacionales, el administrador puede crear más unidades organizacionales.	Precondición

Un usuario tiene nombre, apellidos, nombre de usuario, correo empresarial, correos personales, dirección, carnet de identidad, contraseña, grupos a los que pertenece, permisos, teléfono celular, telefono fijo, telefono de oficina, rol que desempeña dentro de la empresa, quién es su manager y foto de perfil.	Estructura
Si el usuario es un objeto crítico, no puede ser eliminado	Precondición
Si el grupo es un objeto crítico, no puede ser eliminado	Precondición

2.2. Requisitos de la aplicación

Este apartado describe los requisitos esenciales que deben cumplirse para el correcto funcionamiento de la aplicación web propuesta. Los requisitos funcionales especifican las principales características que permitirán la gestión de entidades en el Directorio, mientras que los requisitos no funcionales establecen criterios relacionados con el rendimiento, la seguridad y la usabilidad del sistema. Además, se incluye un diagrama de caso de uso para ilustrar la interacción de los usuarios con las funcionalidades clave del sistema.

2.2.1. Casos de uso del sistema

Los casos de uso del sistema representan las interacciones entre los actores y las funcionalidades clave de la aplicación. A través de estos, se detallan los escenarios en los que los usuarios, administradores y otros actores relevantes interactúan con el sistema para cumplir con los objetivos propuestos. A continuación en la Figura 7, se presenta un diagrama de caso de uso que

proporciona una vista general de las acciones que los usuarios pueden realizar, contribuyendo a una mejor comprensión de los requisitos funcionales definidos.

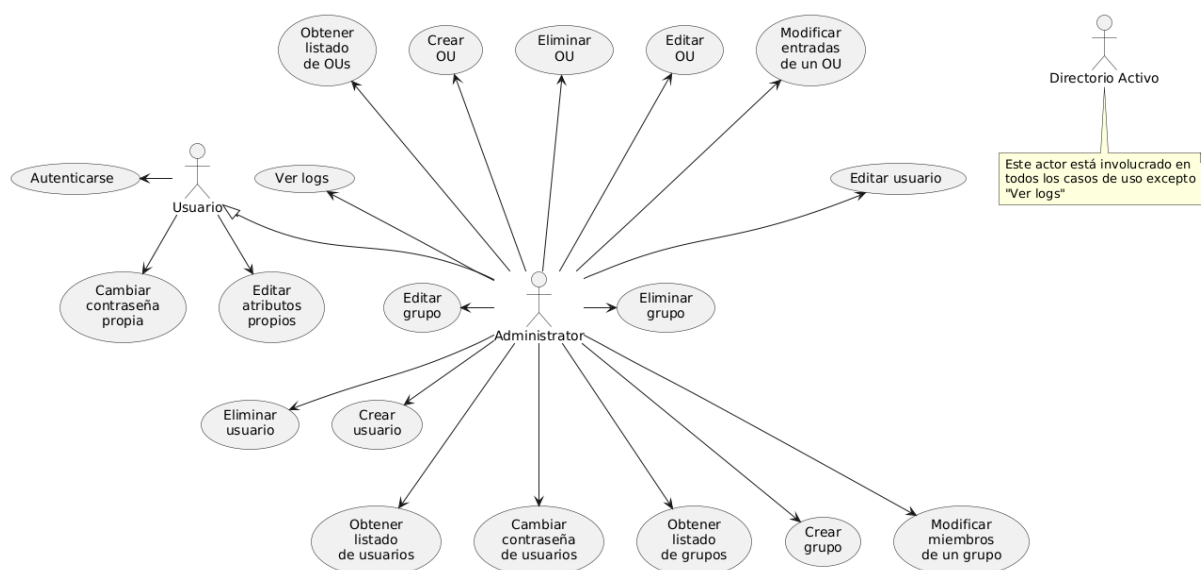


Figura 7: Diagrama de casos de uso del sistema

2.2.2. Requisitos funcionales

Los requisitos funcionales definen las funcionalidades y comportamientos que la aplicación debe ofrecer para cumplir con las necesidades del usuario y los objetivos del proyecto. En la Tabla 6 a continuación se desglosan los requisitos funcionales del sistema.

Tabla 6: Requisitos funcionales del sistema

Código	Requisito	Descripción
RF1	Gestión de Usuarios: Creación de usuarios	La aplicación debe permitir la creación de nuevas cuentas de usuario en el AD.
RF2	Gestión de Usuarios: Modificación de usuarios	Los administradores deben poder actualizar la información de los usuarios, como nombres, correos electrónicos, roles, etc.

RF3	Gestión de Usuarios: Eliminación de usuarios	La aplicación debe permitir la eliminación segura de cuentas de usuario.
RF4	Gestión de Usuarios: Asignación de roles y permisos	Debe ser posible asignar y modificar roles y permisos a los usuarios para definir su nivel de acceso a los recursos.
RF5	Gestión de Grupos: Creación de grupos	Permitir la creación de nuevos grupos en el AD.
RF6	Gestión de Grupos: Modificación de grupos	Posibilidad de añadir o remover usuarios de grupos existentes.
RF7	Gestión de Grupos: Eliminación de grupos	Permitir la eliminación de grupos.
RF8	Gestión de Unidades Organizacionales (OU): Creación de OU	La aplicación debe permitir la creación de nuevas Unidades Organizacionales dentro del AD.
RF9	Gestión de Unidades Organizacionales (OU): Modificación de OU	Los administradores deben poder renombrar y mover OUs dentro de la jerarquía del directorio.
RF10	Gestión de Unidades Organizacionales (OU): Eliminación de OU	Debe ser posible eliminar OUs.
RF11	Gestión de Unidades Organizacionales (OU): Asignación de usuarios y grupos a OU	La aplicación debe facilitar la asignación de usuarios y grupos a las OUs para una mejor organización dentro del AD.
RF12	Autenticación y Autorización: Integración con LDAP	La aplicación debe autenticarse a través de LDAP con el AD para validar usuarios y permisos.
RF13	Autenticación y Autorización: Gestión de sesiones	Manejo de sesiones de usuario con opciones de inicio y cierre de sesión seguro.

RF14	Auditoría y Registro de Actividades: Registro de eventos	La aplicación debe registrar todas las operaciones realizadas sobre los usuarios, grupos y OUs, incluyendo quién y cuándo.
RF15	Interfaz de Usuario (UI): Consola de administración	Proveer una interfaz amigable y accesible para la gestión de usuarios, grupos, OUs y revisión del registro de eventos.
RF16	Interfaz de Usuario (UI): Notificaciones	Mostrar notificaciones en la UI para confirmar la ejecución exitosa de acciones o para advertir sobre errores.
RF17	Configuración de la Aplicación: Facilidad de configuración	La aplicación debe ser fácilmente configurable a través de archivos .json o .yaml, validados mediante JSON Schema.

2.2.3. Requisitos no funcionales

Los requisitos no funcionales describen las cualidades que debe tener la aplicación para garantizar un desempeño óptimo y satisfacer las expectativas de los usuarios. En la Tabla 7 se enumeran los requisitos funcionales que debe presentar el sistema.

Tabla 7: Requisitos no funcionales del sistema

Código	Requisito	Descripción
RNF1	Calidad	La aplicación debe ser capaz de manejar un creciente número de usuarios y grupos sin degradación significativa en el rendimiento.

RNF2	Restricción	La aplicación debe implementar mecanismos de seguridad como cifrado de datos, autenticación segura, y control de acceso basado en roles (RBAC).
RNF3	Calidad	Las operaciones críticas como la autenticación y la gestión de usuarios deben completarse en menos de 2 segundos en condiciones normales de carga.
RNF4	Calidad	La interfaz debe ser intuitiva y fácil de usar, con una curva de aprendizaje mínima para administradores y usuarios avanzados.
RNF5	Restricción	La aplicación debe ser compatible con múltiples navegadores web modernos y adaptarse a diferentes tamaños de pantalla (responsive design).
RNF6	Calidad	El código debe ser modular y bien documentado, facilitando futuras actualizaciones y correcciones de errores.
RNF7	Calidad	La configuración de la aplicación debe estar documentada de manera clara y accesible, disponible en línea en el sitio web del proyecto.
RNF8	Restricción	La aplicación debe ser web
RNF9	Restricción	La aplicación debe ser de código abierto

2.3. Arquitectura de la aplicación web

En esta sección se aborda la arquitectura de la aplicación web propuesta, describiendo su estructura general, los patrones arquitectónicos adoptados y el stack tecnológico utilizado. La arquitectura de la aplicación se diseña con el objetivo de proporcionar una solución escalable, mantenible y eficiente para la gestión de AD. Se detalla la elección de tecnologías así como los problemas frecuentes y su solución.

2.3.1. Stack tecnológico

El stack tecnológico elegido para el desarrollo de la aplicación fue fundamental para garantizar tanto el rendimiento como la mantenibilidad del sistema. A continuación, se describen los componentes principales del stack:

Framework de desarrollo: SvelteKit

SvelteKit se ha seleccionado como el framework principal debido a su capacidad para crear aplicaciones rápidas con una experiencia de desarrollo eficiente. SvelteKit ofrece ventajas como el renderizado en el lado del servidor (SSR), la generación de sitios estáticos (SSG), y una integración sencilla con herramientas modernas como Vite. Además, su enfoque en la eliminación del tiempo de ejecución permite que las aplicaciones sean ligeras y rápidas, lo cual es esencial para la experiencia del usuario.

Cliente LDAP: Idapts

Para la comunicación con el servidor LDAP, se ha optado por Idapts, un cliente LDAP para Node.js que ofrece una API asíncrona basada en promesas y con soporte para typescript. Esta herramienta es ideal para interactuar con AD, permitiendo operaciones como búsqueda, adición, modificación y eliminación de entradas en el directorio. Su uso simplifica la integración de LDAP en la aplicación, garantizando la seguridad y eficiencia necesarias para la gestión de usuarios.

2.3.2. Problemas frecuentes y soluciones en la arquitectura de la aplicación

Este epígrafe aborda los problemas comunes que pueden surgir en el desarrollo de aplicaciones web y presenta soluciones basadas en la arquitectura adoptada. Se exploran aspectos clave como la seguridad, el rendimiento, la escalabilidad y la integración de servicios externos,

destacando cómo la elección adecuada de tecnologías puede ayudar a mitigar estos desafíos.

Integración de APIs y servicios externos: las aplicaciones web a menudo requieren la integración con servicios externos. En este caso, la única integración externa es con AD, que se realiza a través del cliente LDAP proporcionado por la librería `ldapts`. Esta librería permite una conexión eficiente con el directorio, aprovechando su naturaleza asíncrona para mejorar el rendimiento y los tiempos de carga.

Rendimiento y tiempos de carga: problemas de rendimiento y tiempos de carga pueden afectar la experiencia del usuario. `SvelteKit`, con su enfoque de generación de sitios estáticos y pre-renderización, ayuda a mejorar la velocidad de carga. La integración con `ldapts`, una librería asíncrona para la comunicación con AD, mejora el rendimiento al manejar operaciones de directorio de manera eficiente sin bloquear el hilo principal. Esto asegura que las consultas LDAP no impacten negativamente en el rendimiento global de la aplicación.

Escalabilidad y mantenibilidad: con el crecimiento de la aplicación, la escalabilidad y mantenibilidad son esenciales. La arquitectura en capas implementada con `SvelteKit` facilita una separación clara de responsabilidades, mejorando la organización y la evolución del código. La capa de presentación, la capa de negocio y la capa de datos están claramente definidas, permitiendo un desarrollo modular y la fácil adaptación a nuevas funcionalidades. Esta estructura en capas asegura que la aplicación pueda escalar de manera efectiva y mantenerse fácilmente a medida que se amplían los requisitos.

Seguridad de la información y acceso: La gestión de identidades y el control de acceso son esenciales para asegurar la aplicación web. La integración con AD a través del cliente LDAP de `ldapts` permite una autenticación centralizada y segura para los usuarios. Para reforzar la seguridad, se utiliza LDAPS (LDAP sobre SSL/TLS), que cifra la

comunicación entre la aplicación y el servidor LDAP, protegiendo los datos de autenticación y los permisos de acceso contra interceptaciones y ataques. Este enfoque garantiza un manejo riguroso de credenciales y permisos, manteniendo la integridad y confidencialidad de la información.

2.4. Selección y modelado de estilos y patrones

Los estilos arquitectónicos guían la composición y relación de los componentes en un sistema de software. A continuación, se presentan brevemente los estilos seleccionados para la implementación de la solución.

Estilo Llamada-Retorno

El sistema sigue un enfoque de "Llamada-Retorno", donde las capas del sistema se comunican a través de llamadas y retornos de resultados. Esto se aplica a la interacción entre la capa de presentación, la capa de lógica de negocio y la capa de acceso a datos (Figura 23), asegurando una separación clara de responsabilidades y facilitando la escalabilidad y mantenibilidad del software.

En la Figura 8 se muestra un ejemplo de puesta en práctica de este estilo en el sistema, donde la capa de lógica de negocio llama a la función **getDirectChildren** de la capa de acceso a datos, la cual retorna los hijos directos de una entrada del directorio.

```
34  if (config.app.views.ousPage.details.member.show) {  
35      members = getDirectChildren<EntryWithObjectClass>(ldap, ou.distinguishedName, {  
36          attributes: ['dn', 'distinguishedName', 'name', 'objectClass']  
37      });  
38  }
```

Figura 8: Ejemplo del estilo Llamada-Retorno en el sistema

Patrón Cliente-Servidor

El sistema implementa el patrón Cliente-Servidor, donde el cliente (la interfaz de usuario) se comunica con el servidor para solicitar servicios o datos.

Este patrón facilita la separación de responsabilidades y la escalabilidad del sistema. Esta comunicación se realiza en su mayoría mediante el envío de formularios.

Patrón de Acceso a Datos

Para la gestión de los datos en el sistema, se implementa el Patrón de Acceso a Datos como un intermediario entre los servicios de la aplicación y el AD. Específicamente, este patrón facilita las siguientes operaciones:

- **Abstracción del AD:** Se usa la librería *Idapts*, que abstrae las interacciones con el AD, simplificando el código y reduciendo errores.
- **Centralización de acceso:** Todas las solicitudes de datos pasan por esta capa, asegurando un flujo controlado y consistente.
- **Seguridad centralizada:** La capa implementa políticas de seguridad uniformes para todas las operaciones en el AD.

En la Figura 9 muestra la declaración de la función **getDirectChildren** mencionada anteriormente, donde se utiliza el cliente *Idap* (de la librería *Idapts*) para realizar una búsqueda en el directorio. El uso del cliente *Idap* abstrae la interacción con el directorio además de centralizar el acceso.

```
234 export const getDirectChildren = <T extends Entry>(
235   ldap: Client,
236   base: string,
237   opts?: SearchOptions
238 ) => ldap.search(base, { scope: 'one', ...opts }).then(({ searchEntries }) => searchEntries as T[]);
239
```

Figura 9: Uso del cliente *Idap* para realizar una búsqueda en el directorio

Patrón n-capas La arquitectura del sistema está organizada en un patrón de n-capas (Figura 10), con una capa de presentación que interactúa con la capa de lógica de negocio, la cual a su vez se comunica con la capa de acceso a datos. Este enfoque modulariza las responsabilidades y mejora la escalabilidad y mantenibilidad del sistema.

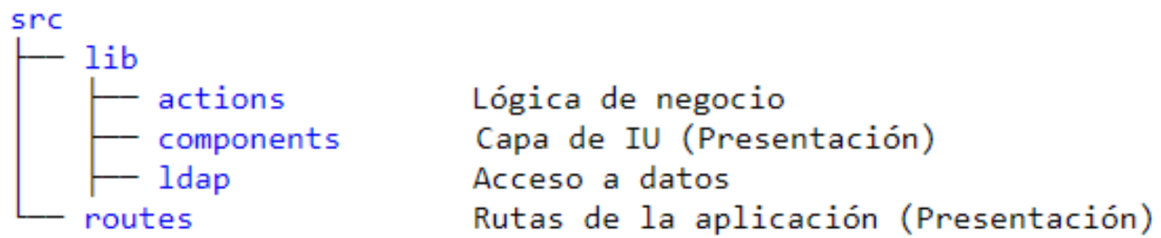


Figura 10: Estructura simplificada del sistema mostrando n-capas

2.5. Principios de diseño

Principio de responsabilidad única

El Principio de Responsabilidad Única (SRP, por sus siglas en inglés) es uno de los principios SOLID de diseño de software. Según SRP, una clase o módulo debería tener una única razón para cambiar, es decir, debería tener una única responsabilidad o propósito.

En el sistema se tienen varios módulos que funcionan de manera independiente y tienen un único propósito, en la Figura 11 se muestran algunos de estos módulos cuyas responsabilidades son la autenticación, la gestión de grupos, la gestión de unidades organizacionales, el manejo de las operaciones directas sobre el árbol del directorio, y la gestión de usuarios respectivamente (Figura 24).

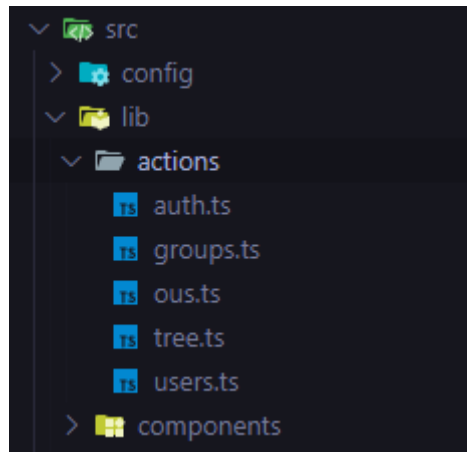


Figura 11: Módulos de la capa de logica de negocio

Principio Hollywood (Principio de Inversión de Control)

Los módulos de alto nivel no deben depender de módulos de bajo nivel. Los módulos de bajo nivel deben depender de módulos de alto nivel. Los módulos de bajo nivel deben ser "llamados" por módulos de alto nivel y no al revés.

El diagrama de secuencia en la Figura 25 ilustra cómo se lleva a cabo el procesamiento de una solicitud de envío de formulario en la arquitectura de n-capas del sistema, destacando la implementación del Principio de Inversión de Control (IoC).

La lógica de negocio delega la interacción con el AD al servicio de acceso a datos, siguiendo el Principio de Inversión de Control. Esto asegura que los módulos de alto nivel no dependan directamente de los módulos de bajo nivel, mejorando la flexibilidad y mantenibilidad del sistema.

Principio Abierto-Cerrado

El Principio Abierto-Cerrado establece que un módulo debe estar abierto para su extensión pero cerrado para su modificación. Este principio, aunque tradicionalmente asociado a la programación orientada a objetos, también es aplicable en la arquitectura de componentes en Svelte.

En lugar de modificar directamente el comportamiento de un componente, se sugiere su extensión mediante la creación de nuevos componentes o

la utilización de la composición. Esto permite la introducción de nuevas funcionalidades sin alterar el código fuente original, preservando la estabilidad y confiabilidad del sistema.

Este principio se aplica en la manera en que los componentes de Svelte están diseñados para ser reutilizables y componibles. Un ejemplo claro se presenta en la Figura 12, muestra cómo un mismo componente Input en Svelte puede adaptarse a través del uso de diferentes propiedades (props) y ranuras (slots), permitiendo su reutilización en distintos contextos sin necesidad de modificar su implementación base.

```
<div class="grid gap-4">
  <Input name="sAMAccountName" inputProps={{ required: true }} {methods}>
    <svelte:fragment slot="label">sAMAccountName</svelte:fragment>
  </Input>

  <Input name="description" {methods}>
    <svelte:fragment slot="label">description</svelte:fragment>
    <svelte:fragment slot="adornment-left">
      <Captions />
    </svelte:fragment>
  </Input>
</div>
```

Figura 12: Instancias del componente Input ilustrando el principio Abierto-Cerrado mediante la composición de componentes

2.6. Patrones de diseño

En el desarrollo de software, los patrones de diseño son soluciones probadas y reutilizables para problemas comunes que los desarrolladores enfrentan al diseñar aplicaciones.

Patrón de Convención sobre Configuración (CoC)

El sistema adopta el Patrón CoC, que permite gestionar configuraciones a través de archivos externos. Esta práctica facilita la personalización y la flexibilidad, permitiendo ajustes sin necesidad de modificar el código fuente.

Esto es crucial para adaptar la aplicación a diferentes entornos y necesidades organizacionales, mejorando la mantenibilidad y escalabilidad del sistema.

En la implementación del Patrón de CoC, se establecieron valores predeterminados para la configuración LDAP y otros aspectos. Se implementó soporte para archivos de configuración en formato JSON y YAML, validados mediante JSON Schema, para permitir ajustes específicos. Esta aproximación no solo facilita la configuración inicial de la aplicación, sino que también ofrece flexibilidad para personalizar la configuración según los requisitos del usuario final.

Patrón de Observador El Patrón Observador es una técnica de diseño fundamental en el desarrollo de software para establecer relaciones de dependencia. En este patrón, cuando un objeto cambia su estado, todos los objetos dependientes son notificados y actualizados automáticamente.

En el contexto de SvelteKit, el patrón Observador es crucial para la gestión de la reactividad a través de los stores. En la Figura 14 se muestra un componente que solo es visible si el valor de la store **loading** de la Figura 13 tiene valor **true**. Por lo tanto cualquier componente que se modifique el valor de esta store provoca una mutación en el estado del componente de la Figura 14.

```
export const globalLoader = writable(false);
```

Figura 13: Declaracion de un store en Sveltekit

```
45 {#if $globalLoader}
46   <div
47     class="..."
48   >
49     <Loader size="xl" />
50   </div>
51 {/if}
52
```

Figura 14: Reacción al cambio en la store

Patrón inyección de dependencias

El patrón de Inyección de Dependencias es un patrón de diseño que busca mejorar la modularidad y la reutilización del código al desacoplar la creación de objetos de su uso. En lugar de que una clase o módulo cree directamente sus dependencias, estas son proporcionadas desde el exterior.

La *Context API* (API de contexto) de Svelte es una herramienta que facilita la comunicación entre componentes sin necesidad de pasar propiedades a través de cada nivel del árbol de componentes. Este mecanismo se utiliza para compartir datos y servicios globales dentro de una aplicación, lo que permite una gestión más eficiente de las dependencias y contribuye a un diseño más limpio y modular.

En el contexto del sistema, la API de contexto se utiliza para inyectar la configuración de la aplicación relacionada con la página actual (Figura 15). Esto permite que los componentes accedan a la configuración específica de la página sin necesidad de recibirla a través de props (Figura 16).

```
import { setContext } from 'svelte';
const { details: config } = $page.data.config.app.views.usersPage;
setContext('config', config);
```

Figura 15: Uso de la API de Contexto de Sveltekit para inyectar la configuración de la página


```

src > lib > components > info-value > info-value.svelte > span.info-value > slot
You, 3 weeks ago | 1 author (You)
1  <script lang="ts">
2      import { getContext } from 'svelte';
3
4      export let key: keyof typeof config;
5      export let entry: Record<string, unknown>;
6      const config = getContext<Record<string, { show: boolean; label: string }>>('config');
7  </script>
8
9  {#if config?.[key].show && entry[key]}
10     <span class="justify-self-end">{config[key].label}</span>
11     <span data-test={key} class="info-value">
12         <slot>
13             {entry[key]}
14         </slot>
15     </span>
16 {/if}
17

```

Figura 16: Uso de la API de Contexto de Sveltekit para leer la configuración de la página

2.7. Implementación

En esta sección se abordarán las funciones fundamentales implementadas en la aplicación web para la gestión y autenticación de usuarios basada en Directorio Activo. Estas funciones son esenciales para garantizar una operación eficiente y segura del sistema.

2.7.1. Configuración del cliente LDAP seleccionado

La configuración del cliente LDAP es crucial para la comunicación efectiva entre la aplicación web y el AD. En esta sección, se detallará el proceso de configuración del cliente LDAP seleccionado, ldaps.

Creación de la instancia del cliente

La creación de la instancia del cliente LDAP (Figura 17) es un proceso modular que permite incorporar opciones de configuración personalizadas. La función **getLDAPClient**, la toma las opciones por defecto de la configuración del sistema, definida en el objeto **config**, y permite la sobrescritura de

parámetros mediante el uso de la función **deepmerge**. De este modo, se asegura flexibilidad en la configuración sin comprometer las configuraciones base.

```
src > lib > ldap > client.ts > ...
1  import config from '$config';
2  import deepmerge from 'deepmerge';
3  import { Client, type ClientOptions } from 'ldaps';
4
5  const { ldapURL: url, strictDN, tlsOptions } = config.directory.ldap;
6
7  export const getLDAPClient = (options?: ClientOptions) => {
8    const opts = deepmerge({ url, strictDN, tlsOptions }, options || {});
9    return new Client({ ...opts });
10 };
11
```

Figura 17: Creación del cliente LDAP con ldaps

En el fragmento de código de la Figura 17, se extraen las configuraciones necesarias para la conexión con el AD (como url, strictDN y tlsOptions) y se combinan con las opciones adicionales que se pueden pasar al crear la instancia del cliente LDAP.

Manejadores de autenticación

Para asegurar una operación segura y eficiente del cliente LDAP durante el ciclo de vida de las solicitudes HTTP en la aplicación, se implementaron dos manejadores (handlers) clave en el archivo *src/hooks.server.ts*:

- **authenticationSetHandler**: Este manejador se encarga de inyectar una función de autenticación en las variables locales del evento. Esta función, llamada auth, valida la presencia de un token de acceso y un token de sesión, y si ambos son válidos, realiza la operación bind con el cliente LDAP, utilizando las credenciales del usuario autenticado (Figura 28).
- **ldapUnbindHandler**: Este manejador asegura que el cliente LDAP cierre correctamente la conexión después de procesar cada solicitud,

mediante la operación **unbind**, liberando así los recursos de forma segura y evitando conexiones abiertas innecesarias (Figura 29).

Este enfoque garantiza que el cliente LDAP esté disponible solo cuando se necesita y que se libere de manera adecuada al final de cada solicitud, lo que mejora tanto la seguridad como el rendimiento general del sistema. Al centralizar la desconexión en un solo lugar, se garantiza que el cliente siempre se cierre adecuadamente después de cada solicitud, evitando conexiones abiertas que podrían saturar el sistema o generar problemas de seguridad. Esta práctica contribuye a una gestión más eficiente y segura del ciclo de vida del cliente LDAP.

2.7.2. Desarrollo de mecanismos de autenticación

La autenticación es una de las funciones más críticas en el sistema, ya que asegura que los usuarios solo puedan acceder a la aplicación con credenciales válidas en el AD. A continuación, se describen los mecanismos de autenticación desarrollados, destacando las interacciones con el AD y el uso de tokens para la gestión de sesiones.

Manejadores de autenticación

Como se explicó en secciones anteriores, los manejadores (handlers) `authenticationSetHandler` (Figura 28) y `ldapUnbindHandler` (Figura 29) juegan un papel crucial en la gestión de la autenticación. Estos aseguran que las conexiones al AD se establezcan y se liberen correctamente tras cada solicitud, manteniendo el sistema seguro y eficiente.

Proceso de inicio de sesión

El proceso de autenticación se basa en la operación **bind** proporcionada por el cliente LDAP. Este proceso se gestiona en el módulo de autenticación, método **signIn**, encargado de validar las credenciales del usuario y establecer las cookies de sesión. En la Figura 18 se muestra el diagrama de flujo

detallando los pasos seguidos por el método:

1. **Validación del formulario y captcha:** El sistema recibe los datos del formulario de inicio de sesión y valida tanto las credenciales del usuario como el token de captcha para prevenir intentos de acceso automatizados.
2. **Autenticación en el AD:** Una vez que el captcha y las credenciales del usuario se validan correctamente, el sistema intenta autenticar al usuario en el AD a través del cliente LDAP. Si la autenticación falla, se maneja de acuerdo con los diferentes códigos de error que proporciona el AD, permitiendo mensajes de error detallados como “Credenciales inválidas”, “Cuenta deshabilitada”, o “Contraseña expirada”.
3. **Generación de tokens y cookies:** Si la autenticación es exitosa, el sistema recupera la entrada del usuario en el AD y genera los tokens de sesión y acceso. Estos tokens se almacenan en cookies para futuras solicitudes.
4. **Redirección del usuario:** Una vez completado el proceso de inicio de sesión, el sistema redirige al usuario a su página de perfil.

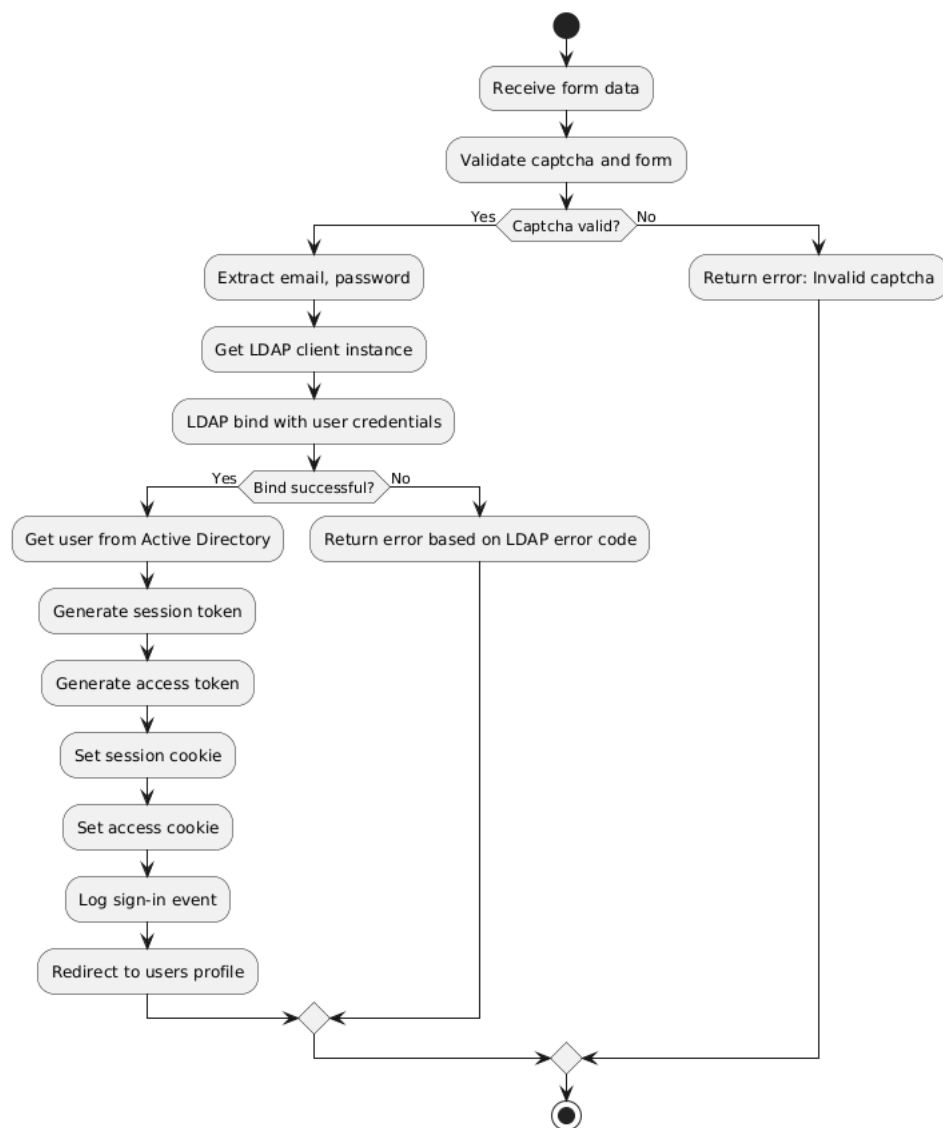


Figura 18: Diagrama de flujo: Función de autenticación

Proceso de cierre de sesión

Este proceso se gestiona en el módulo de autenticación, método signOut, que elimina las cookies de sesión y acceso, invalidando así la sesión del usuario. Este mecanismo asegura que, tras cerrar la sesión, el usuario ya no podrá acceder a la aplicación hasta que vuelva a autenticarse correctamente (Figura 19).

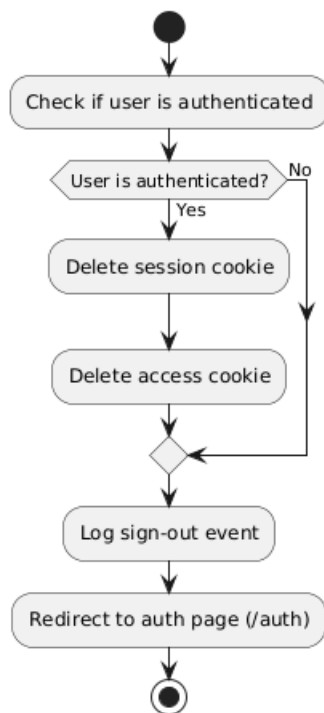


Figura 19: Diagrama de flujo: Función cerrar sesión

2.7.3. Flujos de gestión

Los flujos de gestión definen el ciclo de vida de los objetos del Directorio en la aplicación, asegurando que las operaciones realizadas por los administradores se sincronicen correctamente con el AD. Estas operaciones incluyen la creación, lectura, actualización y eliminación de objetos.

Creación de objetos

Cuando un nuevo objeto, como un usuario o grupo, es creado desde la aplicación, este es registrado en el AD mediante una operación add (Figura 20). El cliente LDAP, ldaps, se utiliza para establecer una conexión segura y confiable con el servidor del AD, donde se envían los atributos requeridos para el nuevo objeto. A lo largo del proceso, se validan los datos, y cualquier error encontrado es notificado al usuario tal y como se muestra en la Figura 32.

```

120 //create the user
121 try {
122     await ldap.add(dn, attributes);
123 } catch (e) {
124     if (e instanceof AlreadyExistsError) {
125         return setError(form, 'sAMAccountName', 'sAMAccountName already in use!');
126     } else if (e instanceof InsufficientAccessException) {
127         appLog(
128             `(InsufficientAccessException) User ${session.sAMAccountName} tried creating a user without having enough access`,
129             'Error'
130         );
131         throw error(403, "You don't have permission to create users!");
132     }
133     const message = 'Something unexpected happened while creating the user';
134     const errorId = errorLog(e, { message });
135     throw error(500, { message, errorId });
136 }
137 appLog(`${session.sAMAccountName} created user: ${dn}`);

```

Figura 20: Uso de la operacion *add* con el cliente LDAP

Lectura de objetos

La lectura de objetos permite a los administradores consultar los detalles de los usuarios, grupos y otros elementos en el AD. Esta operación se realiza mediante el cliente LDAP utilizando la función de búsqueda, filtrando los resultados con base en los parámetros establecidos. Los administradores pueden visualizar los detalles de los objetos, incluidas sus propiedades y relaciones con otros objetos. En la Figura 21 se muestra la función `getAllUsers` donde se utiliza la operación `search` aplicando determinados filtros para obtener las entradas del directorio que sean del esquema usuario.

```

144 export const getAllUsers = (ldap: Client, extraFilters: Filter[] = []): Promise<User[]> =>
145     ldap
146         .search(PUBLIC_BASE_DN, {
147             filter: new AndFilter({
148                 filters: [
149                     new NotFilter({
150                         filter: new EqualityFilter({ attribute: 'objectClass', value: 'computer' })
151                     }),
152                     ...['top', 'person', 'organizationalPerson', 'user'].map(
153                         (value) => new EqualityFilter({ attribute: 'objectClass', value })
154                     ),
155                     ...extraFilters
156                 ]
157             })
158         })
159         .then(({ searchEntries }) => searchEntries as User[]);

```

Figura 21: Uso de la operación *search* del cliente LDAP para listar usuarios

Actualización de objetos

La actualización de objetos en el AD se gestiona a través de operaciones **modify** o **replace**. La aplicación permite a los administradores modificar los atributos de los objetos, como los datos personales de un usuario o la membresía de un grupo. Una vez validados los cambios, estos se aplican directamente al AD, asegurando que la información sea precisa y esté actualizada. En la Figura 33 se detalla el flujo de actualización de un grupo.

Eliminación de objetos

Finalmente, la eliminación de objetos se maneja mediante una operación delete. Los administradores pueden eliminar usuarios, grupos o unidades organizativas, y el cambio se refleja de inmediato en el AD. Antes de la eliminación, se realizan verificaciones adicionales para evitar la pérdida de información importante o la eliminación accidental de objetos clave. En la Figura 30 se muestra el diagrama de actividades para el proceso de eliminar una Unidad Organizacional.

2.8. Documentación y despliegue

Esta sección describe los procesos de despliegue y documentación implementados para garantizar la disponibilidad y mantenibilidad del sistema.

2.8.1. Documentación

La documentación técnica fue generada automáticamente utilizando herramientas especializadas. Para documentar las configuraciones del sistema, se tomó provecho de los esquemas de JSON ya existentes que permiten validar y describir la estructura de los archivos de configuración.

El proceso de generación de documentación está integrado en un pipeline de CI/CD, que incluye:

- Validación de los archivos de JSON Schema.

- Manejo de referencias entre esquemas.
- Generación de documentación web a partir de los esquemas.
- Publicación automática del sitio estático.

La documentación resultante se publica automáticamente en <https://carlos-err406.github.io/opendirectory-config-schema>, proporcionando una referencia técnica completa y siempre actualizada.

2.8.2. Despliegue

El proceso de despliegue de la aplicación fue diseñado para ser simple y reproducible. Se configuraron los entornos de producción, aseguramiento de la calidad y desarrollo, utilizando Docker para empaquetar la aplicación junto con todas sus dependencias, garantizando consistencia entre diferentes entornos.

Se implementó un pipeline de CI/CD utilizando GitLab Pipelines que automatiza el proceso de construcción, pruebas y despliegue. Este pipeline incluye etapas para:

- Ejecución de pruebas unitarias y de integración
- Construcción de imágenes Docker
- Despliegue en el entorno de producción

Se adoptaron estrategias de despliegue continuo que permiten actualizaciones frecuentes con mínimo tiempo de inactividad, asegurando la disponibilidad del sistema mientras se implementan nuevas características y correcciones.

Conclusiones parciales

Este capítulo ha presentado la solución propuesta para la gestión de AD, detallando su modelo de negocio, requisitos, arquitectura y patrones de

diseño. El sistema implementa un modelo de dominio claro con reglas de negocio bien definidas, cubriendo todos los aspectos esenciales de la gestión de usuarios, grupos y unidades organizativas. La arquitectura en capas, combinada con patrones como Llamada-Retorno, Cliente-Servidor y n-Capas, asegura una estructura modular y mantenible. Los principios de diseño SOLID y patrones como Observador e Inyección de Dependencias mejoran la calidad del código. La implementación utiliza SvelteKit como framework principal y Idapts para la integración con AD. Los flujos de gestión implementados cubren todas las operaciones CRUD necesarias, garantizando una sincronización adecuada con el AD. El despliegue se automatizó mediante GitLab Pipelines utilizando contenedores Docker, y la documentación técnica se generó automáticamente a partir de los mismos esquemas que se utilizan para validar la configuración, publicándose automáticamente en un sitio web estático que asegura que siempre se encuentre accesible y actualizada.

Capítulo 3 Validación de la solución

Este capítulo presenta la validación de la solución de gestión de AD, mediante pruebas de lógica del cliente LDAP y flujos de usuario en la interfaz web. El objetivo es verificar que el sistema cumple con los requisitos funcionales y no funcionales.

3.1. Metodología de pruebas

Las pruebas en esta solución se llevaron a cabo utilizando las herramientas Vitest y Playwright, recomendadas por defecto en el framework SvelteKit. Estas herramientas proporcionan un entorno robusto para garantizar que el sistema funcione correctamente bajo distintas condiciones, identificando y corrigiendo posibles errores.

La metodología de pruebas utilizada se basa principalmente en el enfoque de **pruebas de caja negra**, donde se evaluó el comportamiento del sistema sin necesidad de conocer su implementación interna. El objetivo fue garantizar que las entradas proporcionadas generaran las salidas esperadas, según los requisitos funcionales definidos.

Vitest fue la herramienta utilizada para implementar pruebas de **integración** enfocadas en el cliente LDAP, implementado con la librería Idapts. Estas pruebas validaron el comportamiento de las operaciones principales, como la búsqueda, adición, modificación y eliminación de entradas en el AD.

Por otro lado, Playwright se utilizó para realizar pruebas de **extremo a extremo** (*E2E* por sus siglas en inglés) en los flujos de la interfaz de usuario. Estas pruebas evaluaron la interacción del usuario con la aplicación, simulando acciones como la creación, modificación y eliminación de usuarios. Se verificó que las notificaciones de éxito y error se mostraran correctamente, y que el sistema gestionara las sesiones de usuario de manera eficiente.

Tipos de Pruebas

Pruebas unitarias: se centraron en funciones individuales del cliente LDAP para asegurar que las operaciones básicas, como búsquedas y modificaciones, se comporten correctamente en distintos escenarios.

Pruebas E2E: a través de Playwright, se simularon interacciones completas de los usuarios con la interfaz, desde el inicio de sesión hasta la gestión de usuarios, garantizando que la aplicación responda adecuadamente a las solicitudes del usuario final.

3.2. Pruebas

El proceso de validación de la solución propuesta se llevó a cabo mediante un conjunto de pruebas que aseguran el correcto funcionamiento de las funcionalidades más relevantes del sistema. Dado el número considerable de casos de uso (19 en total), se ha priorizado la creación de tablas de casos de prueba para los casos de uso esenciales: creación y eliminación de usuarios, así como la autenticación. Estas tablas permiten organizar y documentar las diferentes condiciones y resultados esperados de cada operación, garantizando una cobertura adecuada de los escenarios más críticos para el sistema.

Tabla 8: Prueba del caso de uso eliminar usuario

Caso de uso a probar	Eliminar usuario
Desarrollador	Carlos Daniel Vilaseca Illnait
Probador	Carlos Daniel Vilaseca Illnait
Fecha	5 de julio 2024
Version	1

Objetivo de la Prueba	Comprobar que el usuario se elimina del directorio correctamente				
Descripción de la prueba	El caso de uso de prueba inicia cuando se acepta la eliminación del usuario en la ventana de confirmación y culmina cuando el usuario queda eliminado del directorio.				
Condiciones	El usuario a eliminar existe en el directorio. El usuario a eliminar no es un objeto crítico del sistema.				
Combinaciones de valores de entrada				Resultado esperado	Resultado obtenido
CP	Escenario	Nombre de la variable de entrada	Valor		
1	El usuario no existe en el directorio	-	-	Error, el usuario no existe	Error, el usuario no existe
2	El usuario es un objeto crítico del sistema	-	-	Error, el usuario no puede ser eliminado	Error, el usuario no puede ser eliminado
3	El usuario existe y no es un objeto crítico del sistema	-	-	Éxito, usuario eliminado	Éxito, usuario eliminado
Observaciones:					

Tabla 9: Prueba del caso de uso crear usuario

Caso de uso a probar	Crear usuario				
Desarrollador	Carlos Daniel Vilaseca Illnait				
Probador	Carlos Daniel Vilaseca Illnait				
Fecha	5 de julio 2024				
Version	1				
Objetivo de la Prueba	Comprobar que el usuario se crea en el directorio correctamente				
Descripción de la prueba	El caso de uso de prueba inicia cuando se accede al formulario de creación de usuario y culmina cuando el nuevo usuario queda creado en el directorio.				
Condiciones	El usuario a crear no existe en el directorio. No se ha alcanzado el límite de usuarios.				
Combinaciones de valores de entrada				Resultado esperado	Resultado obtenido
CP	Escenario	Nombre de la variable de entrada	Valor		
1	No se introduce el nombre	Nombre	None	Error, el campo es requerido	Error, el campo es requerido
2	No se introduce la contraseña	Contraseña	None	Error. el campo es requerido	Error. el campo es requerido
3	No se introduce el nombre de usuario	Nombre de usuario	None	Error. el campo es requerido	Error. el campo es requerido

4	No se introduce la confirmación de contraseña	Confirmación de contraseña	None	Error. Las contraseñas deben coincidir	Error. Las contraseñas deben coincidir
5	No se introduce el correo	Correo	None	Error. el campo es requerido	Error. el campo es requerido
6	La contraseña no tiene dígitos	Contraseña	\$inNumero\$	La contraseña debe incluir al menos un número	La contraseña debe incluir al menos un número
7	La contraseña no llega a los 8 caracteres	Contraseña	C0rt@	La contraseña debe tener al menos 8 caracteres	La contraseña debe tener al menos 8 caracteres
8	La contraseña no tiene caracteres especiales	Contraseña	SinC4r4cter esEsp	La contraseña debe incluir al menos un caracter especial	La contraseña debe incluir al menos un caracter especial
9	La contraseña no tiene mayúsculas	Contraseña	s1n_mayus culas	La contraseña debe incluir al menos una mayúscula	La contraseña debe incluir al menos una mayúscula

10	La contraseña no tiene minúsculas	Contraseña	S1N_MINUSCULAS	La contraseña debe incluir al menos una minúscula	La contraseña debe incluir al menos una minúscula
11	Usuario ya existe	Nombre de usuario	Administrador	Error: El usuario ya existe	Error: El usuario ya existe
12	El correo no es válido	Correo	user@domain	Correo inválido	Correo inválido
			user@domain..com		
			user@.domain.com		
			user@domain@.com		
			user@domain!com		
13	Las contraseñas no coinciden	Contraseña	Def password(1)	Las contraseñas no coinciden	Las contraseñas no coinciden
		Conf. de contraseña	Def password()		
14	Datos requeridos correctos	Nombre	Carlos Daniel	Usuario creado exitosamente	Usuario creado exitosamente
		Nombre de usuario	charlie.01		
		Correo	carlosd@comp.cu		

		Contraseña	dsc!QK33		
		Conf. de contraseña	dsc!QK33		
Observaciones:					

Tabla 10: Prueba del caso de uso autenticarse

Caso de uso a probar	Eliminar usuario				
Desarrollador	Carlos Daniel Vilaseca Illnait				
Probador	Carlos Daniel Vilaseca Illnait				
Fecha	5 de julio 2024				
Version	1				
Objetivo de la Prueba	Comprobar que los usuarios pueden autenticarse en el sistema				
Descripción de la prueba	El caso de uso de prueba inicia cuando se accede a la aplicación y se muestra el formulario de autenticación y culmina cuando el usuario queda autenticado en el sistema y se encuentra en la página de su perfil.				
Condiciones	Los credenciales y el captcha deben ser correctos.				
Combinaciones de valores de entrada				Resultado esperado	Resultado obtenido
CP	Escenario	Nombre de la variable de entrada	Valor		

1	Se resuelve el captcha incorrectamente	captcha	***	Error, Captcha incorrecto	Error, Captcha incorrecto
2	Las credenciales son inválidos	correo	some.email@mail.com	Error, credenciales inválidas	Error, credenciales inválidas
		contraseña	123		
3	Las credenciales son válidas, pero la contraseña expiró	correo	some.email@mail.com	Error, contraseña expirada	Error, contraseña expirada
		contraseña	cOrrEctP@assw0rd		
4	Las credenciales son válidas, pero la cuenta está deshabilitada	correo	some.email@mail.com	Error, su cuenta se encuentra deshabilitada	Error, su cuenta se encuentra deshabilitada
		contraseña	cOrrEctP@assw0rd		
5	Las credenciales son válidas	correo	some.email@mail.com	Éxito, sesión iniciada	Éxito, sesión iniciada
		contraseña	cOrrEctP@assw0rd		
Observaciones:					

En la Figura 26 y Figura 27 se puede apreciar cómo las pruebas de integración realizadas con Vitest y las pruebas E2E ejecutadas con Playwright se llevaron a cabo con éxito. Durante el proceso de prueba, se detectaron varios errores en la implementación del manejo de errores en el cliente LDAP y en la validación de datos de entrada en los formularios. Estos errores, que inicialmente no eran evidentes en pruebas manuales, fueron identificados y corregidos gracias a la robustez de las pruebas implementadas.

La corrección de estos problemas permitió mejorar significativamente la confiabilidad del sistema. En particular, se mejoró la respuesta del cliente LDAP ante entradas no válidas y se reforzó la validación de datos en tiempo real, reduciendo así la posibilidad de fallos en la interacción con el AD. Estas mejoras contribuyen a una experiencia de usuario más estable y eficiente, cumpliendo con los requisitos de robustez y fiabilidad establecidos para la aplicación.

Conclusiones parciales

El proceso de validación demostró que la solución cumple satisfactoriamente con los requisitos funcionales y no funcionales establecidos. Las pruebas unitarias y de integración con Vitest validaron el correcto funcionamiento del cliente LDAP en operaciones críticas como creación, modificación y eliminación de usuarios. Las pruebas E2E con Playwright verificaron que los flujos de usuario en la interfaz web se comportan según lo esperado, incluyendo la autenticación y gestión de usuarios. Durante las pruebas se identificaron y corrigieron errores en el manejo de excepciones y validación de datos, mejorando la robustez del sistema. Los resultados obtenidos confirman que la aplicación es capaz de gestionar el AD a través de LDAP, cumpliendo con los criterios de calidad establecidos en términos de funcionalidad, usabilidad y estabilidad.

Conclusiones

Este trabajo de investigación desarrolló una solución para la gestión de Directorio Activo mediante una aplicación web usando el protocolo LDAP. Las principales conclusiones son:

- El análisis teórico confirmó la importancia de la gestión centralizada mediante Directorio Activo y el protocolo LDAP como estándar para la interacción con directorios.
- La solución propuesta implementa un modelo de dominio completo y una arquitectura en capas que asegura modularidad y mantenibilidad, utilizando SvelteKit y Idapts para la integración con AD.
- El proceso de validación, mediante pruebas unitarias, de integración y E2E, confirmó que la solución cumple con los requisitos funcionales y no funcionales establecidos.

En conclusión, esta investigación contribuye al campo de la gestión de usuarios y directorios con una solución que combina flexibilidad y facilidad de uso, estableciendo un marco de referencia para futuras investigaciones en este ámbito.

Recomendaciones

- Extender aun más las opciones de configuración.
- Extender las pruebas realizadas para hacerlas exhaustivas.
- Incluir mas atributos del esquema de usuarios en los formularios.
- Incluir vistas para otros tipos de objetos del directorio como las computadoras.

Referencias bibliográficas

- [1] M. A. Thakur y R. Gaikwad, «User identity and Access Management trends in IT infrastructure- an overview», en *2015 International Conference on Pervasive Computing (ICPC)*, Pune, India: IEEE, ene. de 2015, págs. 1-4, ISBN: 978-1-4799-6272-3. DOI: 10.1109/PERVASIVE.2015.7086972. visitado 11 de jun. de 2024. dirección: <http://ieeexplore.ieee.org/document/7086972/>.
- [2] A. Josang et al., «Local user-centric identity management», *Journal of Trust Management*, vol. 2, n.º 1, pág. 1, dic. de 2015, ISSN: 2196-064X. DOI: 10.1186/s40493-014-0009-6. visitado 11 de jun. de 2024. dirección: <http://www.journaloftrustmanagement.com/content/2/1/1>.
- [3] J. M. Kizza, «Access control and authorization», en *Guide to Computer Network Security*. Cham: Springer International Publishing, 2024, págs. 195-214, Series Title: Texts in Computer Science, ISBN: 978-3-031-47548-1 978-3-031-47549-8. DOI: 10.1007/978-3-031-47549-8_9. visitado 11 de jun. de 2024. dirección: https://link.springer.com/10.1007/978-3-031-47549-8_9.
- [4] R. Harrison, *Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms*, jun. de 2006. visitado 15 de mayo de 2024. dirección: <https://datatracker.ietf.org/doc/html/rfc4513>.
- [5] M. A. Thakur y R. Gaikwad, «User identity and lifecycle management using LDAP directory server on distributed network», en *2015 International Conference on Pervasive Computing (ICPC)*, Pune, India: IEEE, ene. de 2015, págs. 1-3, ISBN: 978-1-4799-6272-3. DOI: 10.1109/PERVASIVE.2015.7086970. visitado 11 de jun. de 2024. dirección: <http://ieeexplore.ieee.org/document/7086970/>.
- [6] G. Carter, *LDAP system administration*, 1st ed. Beijing ; Sebastopol, CA: O'Reilly, 2003, 294 págs., OCLC: ocm52331373, ISBN: 978-1-56592-491-8.

- [7] J. Sermersheim, «Lightweight Directory Access Protocol (LDAP): The Protocol», *www.rfc-editor.org*, jun. de 2006. DOI: 10 . 17487 / RFC4511. dirección: <https://www.rfc-editor.org/rfc/rfc4511>.
- [8] A. Bartlett, *Samba 4 -Active Directory*, 2005. dirección: https://www.samba.org/samba/news/articles/abartlet_thesis.pdf.
- [9] R. E. Voglmaier, *The ABCs of LDAP*. CRC Press, nov. de 2003.
- [10] RedHat, *What is LDAP authentication?*, 3 de jun. de 2022. dirección: <https://www.redhat.com/en/topics/security/what-is-ldap-authentication>.
- [11] R. Janice. «LDAP authentication with microsoft entra ID - microsoft entra», visitado 28 de jun. de 2024. dirección: <https://learn.microsoft.com/en-us/entra/architecture/auth-ldap>.
- [12] A. Imanudin, *Active Directory Berbasis Linux Samba 4*. Excellent Publishing, oct. de 2019.
- [13] SAMBA, *What Is Samba?*, 2019. dirección: https://www.samba.org/samba/what_is_samba.html.
- [14] V. S. G. Jerez, *VicentGJ/AD-webmanager*, original-date: 2020-09-30T20:15:22Z, 13 de jun. de 2024. visitado 28 de jun. de 2024. dirección: <https://github.com/VicentGJ/AD-webmanager>.
- [15] S. Graber, *stgraber/samba4-manager*, original-date: 2015-09-20T19:41:21Z, 28 de jun. de 2024. visitado 28 de jun. de 2024. dirección: <https://github.com/stgraber/samba4-manager>.
- [16] Dansimp. «Active directory accounts», visitado 30 de jun. de 2024. dirección: <https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-default-user-accounts>.
- [17] T. Howes y M. Smith, *LDAP: programming directory-enabled applications with lightweight directory access protocol* (Macmillan technology series). Indianapolis, Ind: Macmillan Technical Publishing, 1997, 462 págs., ISBN: 978-1-57870-000-4.

- [18] K. Derdus. «User profile attributes in azure active directory b2c», visitado 3 de jul. de 2024. dirección: <https://learn.microsoft.com/en-us/azure/active-directory-b2c/user-profile-attributes>.
- [19] N. Pandey. «A Guide to Node Environment Variables [Process.env Node]», visitado 2 de jul. de 2024. dirección: <https://www.knowledgehut.com/blog/web-development/node-environment-variables>.
- [20] J. Erickson. «What is JSON?», What Is JSON? Oracle, visitado 2 de jul. de 2024. dirección: <https://www.oracle.com/database/what-is-json/>.
- [21] T. Bray, «The JavaScript object notation (JSON) data interchange format», RFC Editor, RFC7158, mar. de 2014, RFC7158. DOI: 10.17487/rfc7158. visitado 2 de jul. de 2024. dirección: <https://www.rfc-editor.org/info/rfc7158>.
- [22] O. Ben-Kiki, C. Evans e I. döt Net, «YAML Ain't Markup Language (YAML™) revision 1.2.2», 1 de oct. de 2021. visitado 3 de jul. de 2024. dirección: <https://yaml.org/spec/1.2.2/>.
- [23] RedHat. «What is YAML?», visitado 2 de jul. de 2024. dirección: <https://www.redhat.com/en/topics/automation/what-is-yaml>.
- [24] L. Attouche, M.-A. Baazizi, D. Colazzo, G. Ghelli, C. Sartiani y S. Scherzinger, *Witness generation for JSON schema*, sep. de 2022. DOI: 10.14778/3565838.3565852. visitado 2 de jul. de 2024. dirección: <https://dl.acm.org/doi/10.14778/3565838.3565852>.
- [25] J. Schema, *JSON Schema - What is a schema?*, ago. de 2024. visitado 10 de nov. de 2024. dirección: <https://json-schema.org/understanding-json-schema/about>.
- [26] AWS. «YAML vs JSON - Difference Between Data Serialization Formats - AWS», visitado 3 de jul. de 2024. dirección: <https://aws.amazon.com/compare/the-difference-between-yaml-and-json/>.
- [27] M. Eriksson y V. Hallberg, «Comparison between JSON and YAML for data serialization», Tesis doct., School of Computer Science y Engineering Royal Institute of Technology, 2011. visitado 3 de jul. de 2024.

2024. dirección: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=636a2b04d98c0af8e9d6f59148352dd63af4f0c1>.

Siglarío

AD: Directorio Activo (Active Directory)

API: Interfaz de Programación de Aplicaciones (Application Programming Interface)

e2e: Pruebas de extremo a extremo (End-to-End)

JSON: Notación de Objetos JavaScript (JavaScript Object Notation)

LDAP: Protocolo Ligero de Acceso a Directorios (Lightweight Directory Access Protocol)

LDAPS: Protocolo Ligero de Acceso a Directorios Seguro (Lightweight Directory Access Protocol Secure)

SLCA: Software Libre y Código Abierto (Free and Open-Source Software)

Sveltekit: Framework de desarrollo web

YAML: Lenguaje de Marcado de Datos (YAML Ain't Markup Language)

Anexos

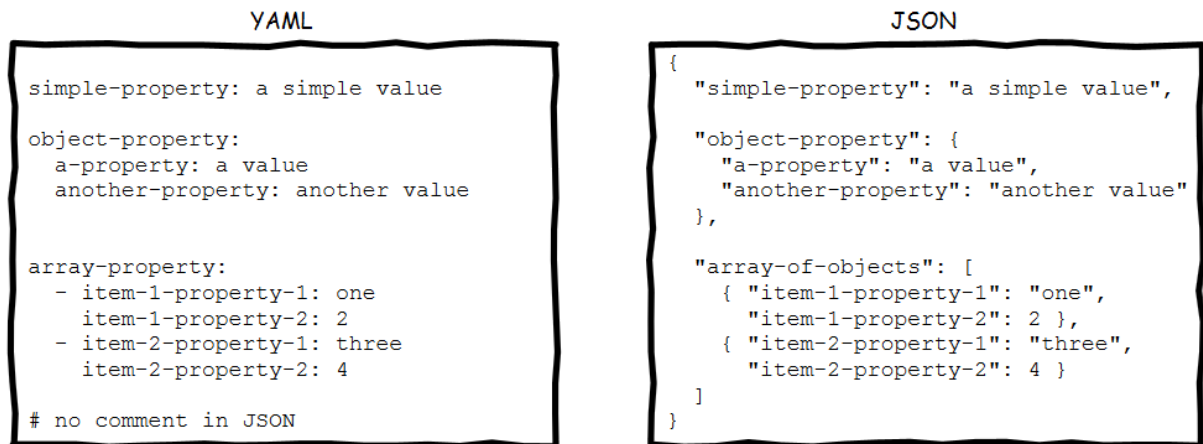


Figura 22: Comparación de sintaxis entre YAML y JSON

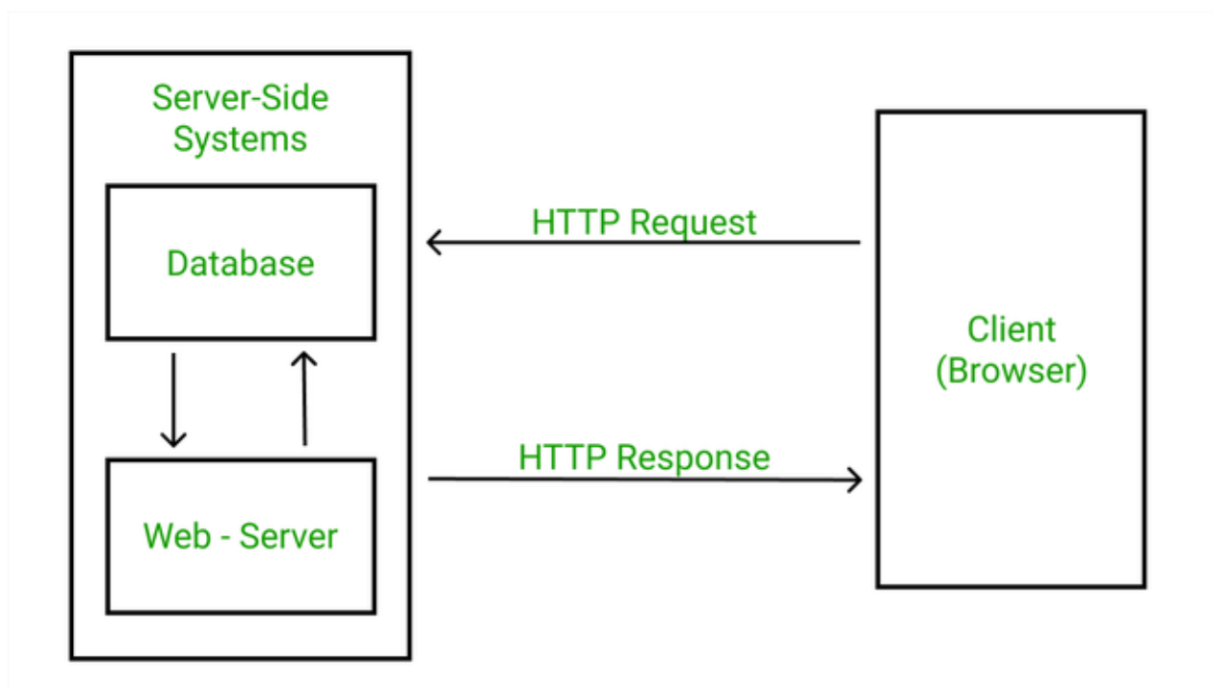


Figura 23: Diagrama representando el estilo arquitectónico Llamada Retorno

```
src > lib > actions > users.ts > ...  
You, last month | 1 author (You)  
1 > import config from '$config'; ...  
47  
48 > export const createUser: Action = async (event) => { ...  
156 };  
157  
158 > export const deleteUser: Action = async (event) => { ...  
193 };  
194  
195 > export const deleteManyUsers: Action = async (event) => { ...  
231 };  
232  
233 > export const changeUserPassword: Action = async (event) => { ...  
298 };  
299  
300 > export const updateUser: Action = async (event) => { ...  
416 };  
417  
418 > export const updateMembership: Action = async (event) => { ...  
463 };  
464
```

Figura 24: Vista colapsada del módulo de encargado de la gestión de usuarios

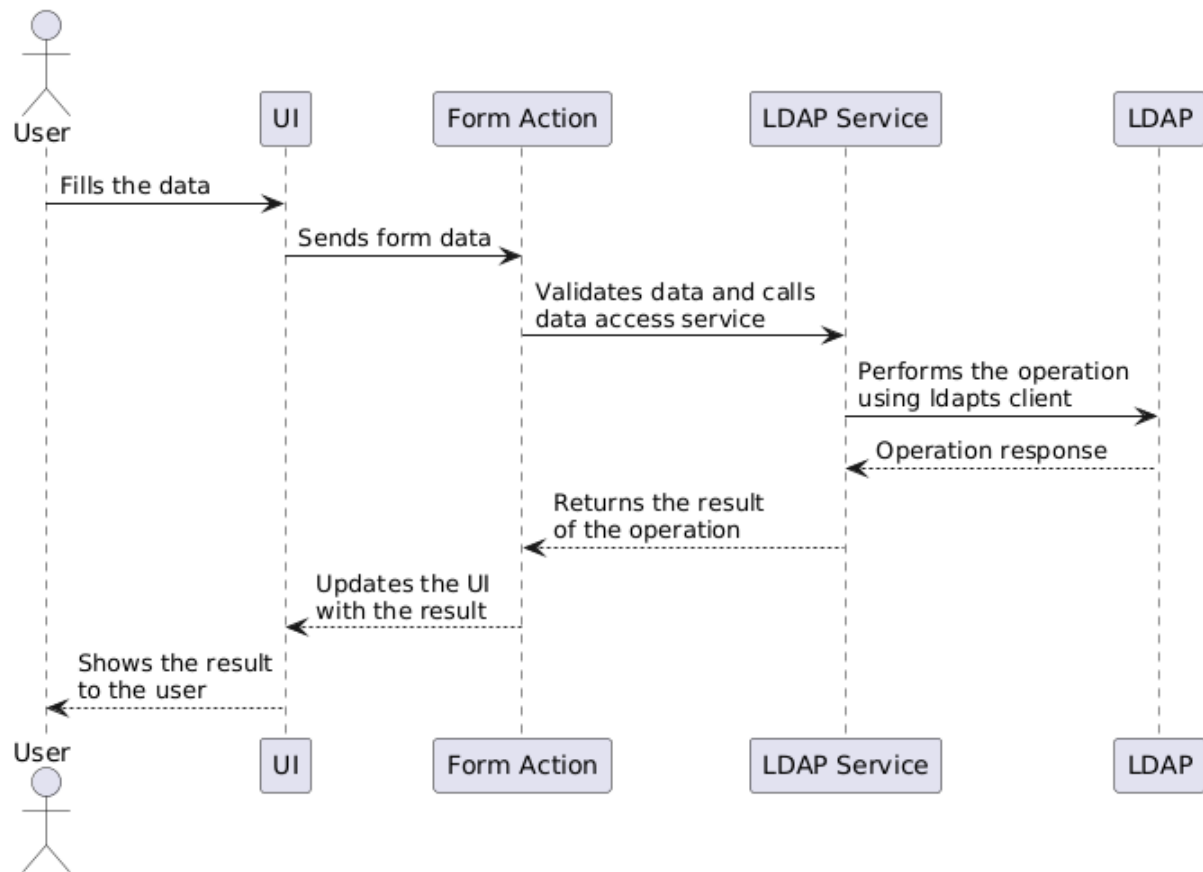


Figura 25: Diagrama de sequencia mostrando un flujo de envío de formulario general

```
RERUN rerun all tests

stdout | src\config\index.ts:18:10
Test environment detected. Overriding CONFIG_PATH configuration for 'app.config.test.yaml'

✓ tests/unit/index.test.ts (1)
✓ tests/unit/ldap-client.test.ts (11) 2785ms
✓ tests/unit/pagination.test.ts (2)

Test Files 3 passed (3)
  Tests 14 passed (14)
  Start at 21:10:27
  Duration 4.16s

PASS Waiting for file changes...
press h to show help, press q to quit
```

Figura 26: Pruebas con vitest pasaron exitosamente

```
[WebServer] → Local: http://localhost:5173/
[WebServer] → Network: use --host to expose
Running global setup
adding test user
test user added!
[WebServer] Test environment detected. Overriding CONFIG_PATH configuration for 'app.config.test.yaml'
[WebServer] Logging is disabled

Running 34 tests using 8 workers

✓ 1 auth\load.test.ts:9:1 › should load the sign-in form (2.4s)
✓ 2 non-admin\routes.test.ts:10:1 › Loads logged user's page (16.6s)
✓ 3 admin\routes.test.ts:6:1 › Loads users page (19.6s)
✓ 4 protected\rredirects.test.ts:7:2 › Accessing to protected pages without being signed in redirects to sign in page › Accessing to /users/me (1.7s)
✓ 5 admin\me-page.test.ts:12:1 › Loads administrator avatar (12.5s)
✓ 6 non-admin\me-page.test.ts:12:1 › Loads user avatar (12.7s)
✓ 7 auth\auth-page.test.ts:12:2 › user actions › should accept user input (2.8s)
✓ 8 auth\sign-in.test.ts:12:1 › should submit the form and redirect to the users profile page (23.2s)
✓ 9 protected\rredirects.test.ts:12:2 › Accessing to protected pages without being signed in redirects to sign in page › Accessing to /users (4.1s)
✓ 10 auth\load.test.ts:13:1 › should display all required inputs and captcha (5.9s)
✓ 11 auth\auth-page.test.ts:22:2 › user actions › should refresh captcha (5.8s)
✓ 12 protected\rredirects.test.ts:17:2 › Accessing to protected pages without being signed in redirects to sign in page › Accessing to /groups (2.0s)
✓ 13 protected\rredirects.test.ts:22:2 › Accessing to protected pages without being signed in redirects to sign in page › Accessing to /logs (5.1s)
✓ 14 auth\auth-page.test.ts:30:2 › user actions › should validate user input (5.6s)
✓ 15 protected\rredirects.test.ts:27:2 › Accessing to protected pages without being signed in redirects to sign in page › Accessing to /ous (4.2s)
✓ 16 non-admin\me-page.test.ts:16:1 › Loads user name (7.6s)
✓ 17 admin\me-page.test.ts:16:1 › Loads administrator name (7.7s)
✓ 18 non-admin\routes.test.ts:15:1 › Non admin users cant access groups page (10.7s)
✓ 19 protected\rredirects.test.ts:32:2 › Accessing to protected pages without being signed in redirects to sign in page › Accessing to /tree (4.3s)
✓ 20 admin\routes.test.ts:11:1 › Loads groups page (10.1s)
✓ 21 non-admin\me-page.test.ts:21:1 › Loads user data (4.2s)
✓ 22 admin\me-page.test.ts:21:1 › Loads administrator data (4.1s)
✓ 23 protected\rredirects.test.ts:37:2 › Accessing to protected pages without being signed in redirects to sign in page › Accessing to /settings (3.2s)
✓ 24 admin\me-page.test.ts:60:1 › Loads edit user and change password buttons (4.7s)
✓ 25 non-admin\me-page.test.ts:59:1 › Loads edit user and change password buttons (5.9s)
✓ 26 non-admin\routes.test.ts:24:1 › Non admin users cant access group details page (7.0s)
✓ 27 admin\routes.test.ts:16:1 › Loads groups details page (3.6s)
✓ 28 non-admin\me-page.test.ts:64:1 › Shows the disabled self update message according to config (3.6s)
✓ 29 admin\routes.test.ts:23:1 › Loads logs page (3.9s)
✓ 30 non-admin\routes.test.ts:35:1 › Non admin users cant access the logs page (3.9s)
✓ 31 admin\routes.test.ts:28:1 › Loads tree page (4.0s)
✓ 32 non-admin\routes.test.ts:44:1 › Non admin users cant access the tree page (4.0s)
✓ 33 admin\routes.test.ts:33:1 › Loads ous page (2.5s)
✓ 34 non-admin\routes.test.ts:53:1 › Non admin users cant access the ous page (3.8s)

Running global teardown...
deleting test user
test user deleted!

34 passed (1.4m)
```

Figura 27: Pruebas de e2e pasaron exitosamente

```

25  /**
26   * sets the auth function in the locals, which returns the session and a binded ldap client instance
27   */
28  const authenticationSetHandler: Handle = async ({ event, resolve }) => {
29    const auth = async ({ cookies }: typeof event) => {
30      const access = getAccessToken(cookies);
31      if (!access) return null;
32      const session = getSessionToken(cookies);
33      if (!session) return null;
34      try {
35        const ldap = getLDAPClient();
36        const { email, password } = verifyAccessToken(access);
37        const [sAMAccountName] = email.split('@');
38        await ldap.bind(`${sAMAccountName}@${PUBLIC_LDAP_DOMAIN}`, password);
39        return {
40          ldap,
41          session: verifySessionToken(session)
42        };
43      } catch (e) {
44        errorLog(e);
45        return null;
46      }
47    };
48
49    event.locals.auth ??= () => auth(event);
50
51    return resolve(event);
52  };

```

Figura 28: Manejador: autenticación y creacion del cliente LDAP

```

54  /**
55   * Runs ldap.unbind() after resolve
56   */
57  const ldapUnbindHandler: Handle = async ({ event, resolve }) => {
58    const response = await resolve(event);
59    const { locals } = event;
60    const auth = await locals.auth();
61    if (!auth) return response;
62    const { ldap } = auth;
63    try {
64      await ldap.unbind();
65    } catch (e) {
66      errorLog(e, { message: 'error at ldapUnbindHandler hook' });
67    }
68    return response;
69  };
70

```

Figura 29: Manejador: Cierre de la conexion del cliente LDAP

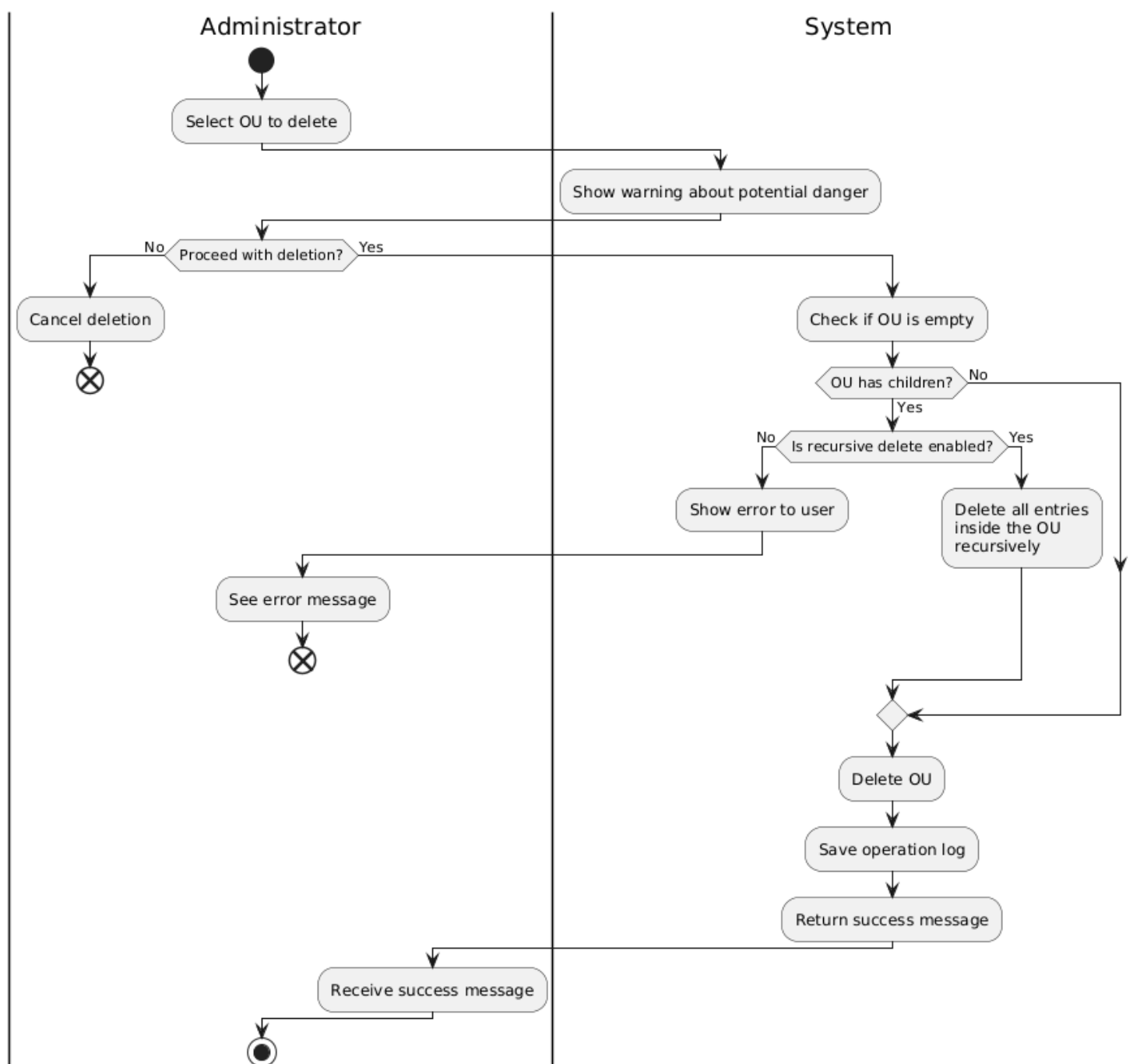

























































Figura 30: Diagrama de actividades: Eliminar Unidad Organizacional

List of companies using Active Directory

Technology is any of Active Directory+ Add Filter

{ } APIExport

Company	Country	Industry	Employees	Revenue	Technologies
 Axiom Technologies  	 Australia	It Services And It Consulting	501		 Active Directory 
 Gallagher  	 United States	Insurance	40K		 Active Directory 
 Computer World Services  		It Services And It Consulting	201		 Active Directory 
 Sybex Support Services  	 Canada	It Services And It Consulting	200		 Active Directory 
 L'Assurance Maladie  		Insurance	10K		 Active Directory 
 CompassMSP  	 United States	It Services And It Consulting	160	\$23M	 Active Directory 
 TUI Group  		Travel Arrangements	10K		 Active Directory 
 Hudson for Sharp  		Computer Hardware Manufacturing			 Active Directory 
 Xcelerate Solutions  	 United States	Business Consulting And Services	344	\$36M	 Active Directory 
 バリューアークコンサルティング株式会社  					 Active Directory 

Showing 10 of 103,755 resultsPage 1 of 10376<K<>>

Figura 31: Compañías que usan Directorio Activo hoy en día en el mundo (Marzo 2025, 100 000+)

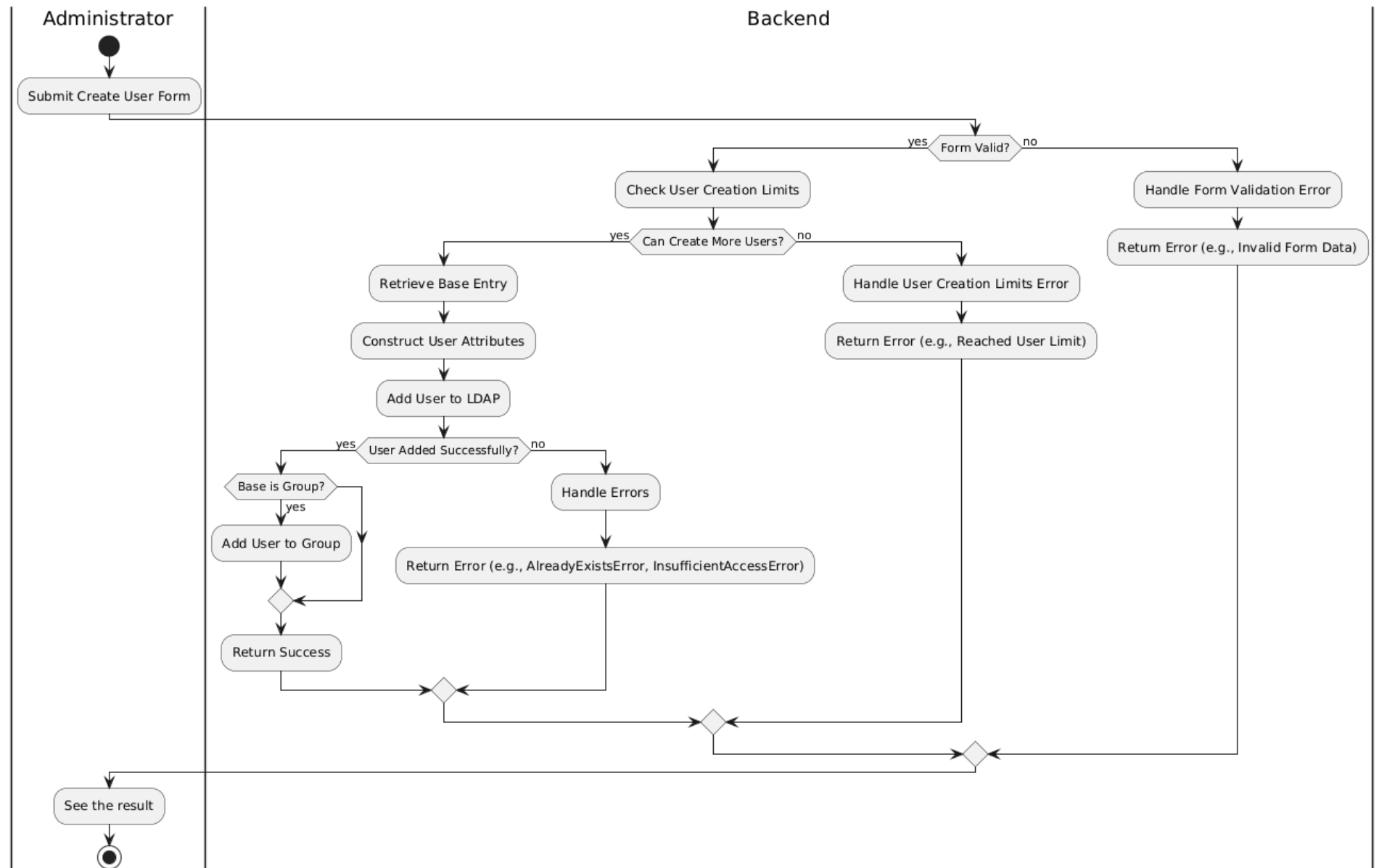


Figura 32: Diagrama de actividades: Crear usuario

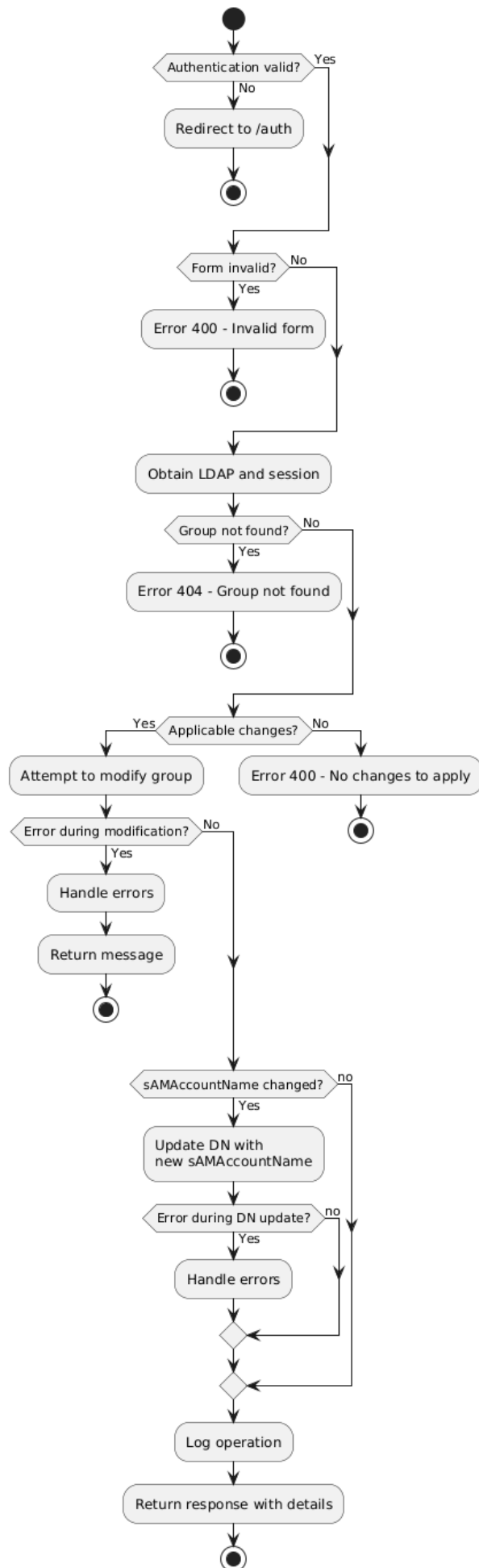


Figura 33: Diagrama de flujo: Actualizar grupo