Universidad Politécnica
de Madrid
**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

Plataforma para la Gestión de Usuarios en Servicios de un Grupo de Investigación

Autor: Jaime Martín-Borregón Musso
Tutor: Alejandro Rodríguez González
Co-tutora: Lucía Prieto Santamaría

Madrid, Enero de 2026

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Plataforma para la Gestión de Usuarios en Servicios de un Grupo de Investigación

Enero, 2026

Autor: Jaime Martín-Borregón Musso

Tutor: Alejandro Rodríguez González
alejandro.rg@upm.es
ETSI Informáticos
Universidad Politécnica de Madrid
Lenguajes y Sistemas Informáticos e Ingeniería del Software

Co-tutora: Lucía Prieto Santamaría
Lenguajes y Sistemas Informáticos e Ingeniería del Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

La gestión de usuarios y accesos a servicios en grupos de investigación conlleva una complejidad creciente a medida que aumenta el número de miembros, servicios y recursos compartidos. En muchos casos, estas tareas se realizan de forma manual o mediante herramientas heterogéneas, lo que dificulta el control del ciclo de vida de los usuarios y la trazabilidad de las operaciones realizadas.

Este Trabajo Fin de Grado aborda el análisis, diseño e implementación de una plataforma web orientada a la gestión centralizada de usuarios en los servicios de un grupo de investigación. El objetivo principal del trabajo es definir una solución que permita administrar usuarios, afiliaciones temporales y accesos a distintos servicios de forma estructurada, proporcionando un marco común para la gestión de estas operaciones.

El trabajo se inicia con un análisis de requisitos que identifica las funcionalidades necesarias del sistema, diferenciando entre requisitos funcionales y no funcionales. A partir de este análisis se plantea un diseño de la arquitectura de la plataforma basado en un modelo web en capas, así como un diseño del modelo de datos que representa los principales elementos del dominio, como usuarios, afiliaciones, servicios, documentos y tareas internas.

Sobre la base de este diseño, se desarrolla una implementación funcional de la plataforma, centrada en la gestión de usuarios, afiliaciones y accesos a servicios mediante un enfoque demostrativo. Las operaciones relacionadas con la provisión de servicios se modelan a través de tareas internas que permiten simular la ejecución diferida de acciones sobre sistemas externos, aportando trazabilidad y control sin comprometer infraestructuras reales. Este enfoque resulta especialmente adecuado en un contexto académico, donde el acceso directo a determinados servicios no siempre es posible.

La plataforma incluye asimismo una interfaz web que permite a los administradores consultar y gestionar la información asociada a cada usuario desde un entorno unificado. A través de esta interfaz es posible visualizar el estado de las afiliaciones, los accesos a servicios, la documentación asociada y el registro de tareas realizadas o programadas.

Finalmente, se describen las pruebas realizadas sobre el sistema con el fin de validar su correcto funcionamiento y verificar el cumplimiento de los requisitos definidos. Los resultados obtenidos permiten concluir que la solución propuesta es viable desde el punto de vista técnico y constituye una base sólida para una posible evolución futura hacia un sistema plenamente operativo, en el que las integraciones demostrativas puedan sustituirse por integraciones reales con los servicios del grupo de investigación.

Abstract

User and access management in research groups becomes increasingly complex as the number of members, services, and shared resources grows. In many cases, these tasks are performed manually or using heterogeneous tools, which makes it difficult to control the user lifecycle and to ensure proper traceability of the operations carried out.

This Final Degree Project addresses the analysis, design, and implementation of a web-based platform aimed at the centralized management of users in the services of a research group. The main objective of the work is to define a solution that allows the administration of users, temporary affiliations, and access to different services in a structured manner, providing a unified framework for managing these operations.

The project begins with a requirements analysis that identifies the necessary functionalities of the system, distinguishing between functional and non-functional requirements. Based on this analysis, a layered web architecture is designed, along with a data model that represents the main domain elements, such as users, affiliations, services, documents, and internal tasks.

Building upon this design, a functional implementation of the platform is developed, focusing on user management, affiliations, and service access through a demonstrative approach. Operations related to service provisioning are modeled using internal tasks that simulate deferred execution of actions on external systems, providing control and traceability without affecting real infrastructures. This approach is particularly suitable in an academic context, where direct access to certain services is often restricted.

The platform also includes a web-based user interface that allows administrators to consult and manage user-related information from a unified environment. Through this interface, it is possible to visualize the status of affiliations, service accesses, associated documentation, and the log of executed or scheduled tasks.

Finally, the tests carried out on the system are described in order to validate its correct behavior and verify compliance with the defined requirements. The results show that the proposed solution is technically feasible and provides a solid foundation for future evolution towards a fully operational system, where demonstrative integrations could be replaced by real integrations with the research group services.

Contenido

Índice de figuras

Índice de listings

Índice de tablas

1 Introducción

La gestión de usuarios y de sus permisos de acceso a servicios informáticos es una tarea fundamental en los grupos de investigación universitarios. Estos entornos suelen disponer de una infraestructura tecnológica heterogénea formada por servidores, sistemas de almacenamiento, plataformas de desarrollo colaborativo y otros servicios necesarios para la actividad investigadora diaria. El acceso a dichos recursos está condicionado por la pertenencia temporal de los usuarios al grupo, así como por su rol y afiliación institucional.

En muchos grupos de investigación, la administración de usuarios se realiza de forma manual o mediante procedimientos poco centralizados. Este enfoque puede dar lugar a problemas como errores en la concesión o revocación de accesos, falta de coherencia entre servicios, ausencia de trazabilidad de las acciones realizadas y dificultades para gestionar la caducidad de permisos cuando un usuario deja de formar parte del grupo.

La rotación frecuente de estudiantes, investigadores y colaboradores externos incrementa aún más la complejidad de esta gestión. En este contexto, resulta necesario disponer de herramientas que permitan centralizar la información de los usuarios y facilitar el control de accesos de forma clara, estructurada y auditable.

Este Trabajo Fin de Grado propone el diseño y desarrollo de una plataforma web para la gestión de usuarios en los servicios de un grupo de investigación. La plataforma permite registrar usuarios, asociarlos a afiliaciones con fechas de inicio y fin, gestionar los servicios a los que tienen acceso y visualizar el estado de dichas asignaciones desde una única interfaz. La solución está concebida con un enfoque modular y extensible, de modo que pueda adaptarse a distintos servicios y evolucionar hacia una integración real con infraestructuras existentes.

Objetivos del trabajo

El objetivo principal de este Trabajo Fin de Grado es diseñar e implementar una plataforma web que permita la gestión centralizada de usuarios y de sus accesos a los distintos servicios de un grupo de investigación, mejorando el control, la trazabilidad y la organización de los procesos de alta, baja y modificación de usuarios.

Para alcanzar este objetivo general, se plantean los siguientes objetivos específicos:

- Analizar las necesidades habituales de gestión de usuarios en un grupo de investigación con múltiples servicios informáticos.
- Diseñar un modelo de datos que represente usuarios, afiliaciones y servicios de forma clara y extensible.
- Desarrollar una aplicación web que permita dar de alta, modificar y consultar usuarios.
- Gestionar afiliaciones con fechas de inicio y fin, facilitando el control de la caducidad de accesos.

- Proporcionar una interfaz de usuario intuitiva y coherente que centralice la administración de usuarios.
- Incorporar mecanismos básicos de automatización y trazabilidad de las operaciones realizadas.
- Diferenciar claramente la lógica desarrollada en el proyecto de las herramientas y tecnologías externas utilizadas como soporte.

2 Tecnologías utilizadas

El desarrollo de la plataforma se ha realizado utilizando un conjunto de tecnologías consolidadas en el ámbito del desarrollo de aplicaciones web, seleccionadas por su estabilidad, amplia documentación y adecuación al contexto académico del proyecto.

El backend de la aplicación se ha implementado en **Java** [6] utilizando el framework **Spring Boot**, que facilita la creación de aplicaciones web estructuradas siguiendo el patrón Modelo–Vista–Controlador (MVC) [1], [2]. Spring Boot permite gestionar de forma eficiente las dependencias del proyecto, la configuración del entorno y la definición de controladores, servicios y repositorios, contribuyendo a una arquitectura clara y mantenible.

La interfaz de usuario se ha desarrollado mediante **Thymeleaf** como motor de plantillas, lo que permite generar vistas HTML dinámicas a partir de los datos proporcionados por el backend [3]. Para el diseño visual y la coherencia de la interfaz se ha utilizado **Bootstrap**, obteniendo una apariencia profesional y homogénea sin necesidad de desarrollar hojas de estilo personalizadas complejas [4].

La persistencia de datos se ha resuelto mediante **SQLite**, una base de datos relacional ligera que se integra fácilmente en el proyecto y resulta adecuada para el alcance del sistema desarrollado [5]. SQLite permite almacenar de forma estructurada la información relativa a usuarios, afiliaciones, servicios, documentos y tareas, evitando la necesidad de configurar un servidor de base de datos externo durante el desarrollo.

La gestión del ciclo de vida del proyecto y de sus dependencias se ha llevado a cabo mediante **Maven**, herramienta que permite definir de forma declarativa las librerías utilizadas, automatizar el proceso de compilación y facilitar la reproducibilidad del entorno de desarrollo [7].

Para el control de versiones se ha utilizado **Git**, lo que ha permitido gestionar la evolución del código, registrar los cambios realizados y facilitar la organización del trabajo durante el desarrollo del proyecto. El entorno de desarrollo integrado empleado ha sido **Visual Studio Code**, seleccionado por su ligereza y por el soporte de extensiones específicas para Java y Spring Boot.

Finalmente, para la automatización de determinadas operaciones se han utilizado los mecanismos proporcionados por el propio framework, permitiendo programar tareas y simular la integración con servicios externos. Todas estas tecnologías actúan como soporte para la solución desarrollada, siendo el diseño del sistema, la definición de la arquitectura y la implementación de la lógica de negocio responsabilidad directa del autor.

3 Análisis de los requisitos

El presente capítulo recoge el análisis de requisitos del sistema desarrollado en este Trabajo Fin de Grado. El objetivo de este análisis es identificar y describir de forma estructurada las funcionalidades que debe ofrecer la plataforma, así como las restricciones y características no funcionales que condicionan su diseño e implementación.

Los requisitos se han identificado a partir del análisis del funcionamiento de un grupo de investigación real, de las necesidades de gestión de usuarios y servicios detectadas, y de las decisiones tomadas durante el desarrollo del proyecto. Para su descripción se han utilizado plantillas de requisitos que permiten detallar, para cada caso, los actores implicados, las precondiciones, el flujo normal de funcionamiento y las posibles alternativas.

3.1 Alcance del sistema

El sistema desarrollado en este Trabajo Fin de Grado tiene como objetivo proporcionar una plataforma centralizada para la gestión de usuarios pertenecientes a un grupo de investigación, así como de sus afiliaciones y accesos a distintos servicios asociados. La plataforma está orientada a facilitar las tareas administrativas relacionadas con el alta, modificación y baja de usuarios, así como al control temporal de su pertenencia al grupo.

El alcance del sistema incluye la gestión de información básica de los usuarios, la definición de afiliaciones con fechas de inicio y fin, la administración de accesos a servicios y la asociación de documentación relevante. Asimismo, el sistema contempla mecanismos de automatización y trazabilidad que permiten simular la provisión de servicios de forma controlada.

Quedan fuera del alcance de este trabajo la integración completa y operativa con los servicios reales del grupo de investigación, así como la ejecución directa de acciones administrativas sobre sistemas productivos. No obstante, el diseño de la plataforma contempla explícitamente cómo se llevarían a cabo dichas integraciones en un entorno real.

3.2 Identificación de requisitos

Los requisitos del sistema se han identificado a partir del análisis de las necesidades reales de gestión de usuarios en un grupo de investigación, así como de las conversaciones mantenidas con el tutor del proyecto y de las decisiones tomadas durante el desarrollo del sistema.

A partir de este análisis, los requisitos se han clasificado en dos grandes grupos:

- **Requisitos funcionales**, que describen las funcionalidades que debe ofrecer la plataforma.

- **Requisitos no funcionales**, que recogen aspectos relacionados con la calidad, seguridad, usabilidad y restricciones del sistema.

En las siguientes secciones se detallan los principales requisitos funcionales y no funcionales identificados, utilizando una plantilla estructurada que permite describir cada requisito de forma clara y homogénea.

3.3 Requisitos funcionales

A continuación se describen los principales requisitos funcionales del sistema, utilizando una plantilla homogénea que permite detallar el comportamiento esperado de cada funcionalidad:

Alta de usuario en el sistema

La Tabla 3.1 muestra la interfaz utilizada para el alta de nuevos usuarios en el sistema, en la que se introducen los datos identificativos necesarios para su registro y posterior gestión dentro de la plataforma.

Tabla 3.1: Requisito funcional RF-01 – Alta de usuarios

Campo	**Descripción**
Nombre	RF-01 – Alta de usuario en el sistema
Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	Permitir al administrador dar de alta un nuevo usuario en la plataforma, registrando su información básica y dejándolo disponible para la posterior gestión de afiliaciones y accesos a servicios.
Actores	Administrador del sistema
Precondiciones	El administrador debe estar autenticado en la plataforma y disponer de permisos de gestión de usuarios.
Flujo Normal	1. El administrador accede a la opción de creación de usuario. 2. El sistema muestra un formulario para introducir los datos básicos del usuario. 3. El administrador introduce la información requerida (nombre, correo electrónico, identificador, etc.). 4. El sistema valida los datos introducidos. 5. El sistema registra el nuevo usuario en la base de datos. 6. El usuario queda disponible para su gestión dentro de la plataforma.
Flujo Alternativo	4a. Si los datos introducidos no son válidos o están incompletos, el sistema muestra un mensaje de error y solicita su corrección. 4b. Si ya existe un usuario con el mismo identificador, el sistema informa del conflicto y cancela el alta.
Nivel de Importancia	**Alto**
Postcondiciones	El usuario queda registrado en el sistema y puede ser asociado a afiliaciones, servicios y documentación.

Gestión de afiliaciones de usuarios

La Tabla 3.2 ilustra la gestión de afiliaciones de un usuario, permitiendo consultar el estado de la afiliación activa y las fechas de inicio y fin asociadas, facilitando el control de la pertenencia temporal al grupo de investigación.

Tabla 3.2: Requisito funcional RF-02 – Gestión de afiliaciones

Campo	**Descripción**
Nombre	RF-02 – Gestión de afiliaciones de usuarios
Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	Permitir al administrador gestionar las afiliaciones de un usuario al grupo de investigación, definiendo el tipo de afiliación y su periodo de validez mediante fechas de inicio y fin.
Actores	Administrador del sistema
Precondiciones	El usuario debe estar previamente registrado en el sistema.
Flujo Normal	1. El administrador accede a la vista de detalle de un usuario. 2. El sistema muestra las afiliaciones asociadas al usuario. 3. El administrador añade una nueva afiliación o modifica una existente. 4. El administrador introduce el tipo de afiliación y las fechas de inicio y fin. 5. El sistema valida las fechas introducidas. 6. El sistema guarda la afiliación asociada al usuario.
Flujo Alternativo	5a. Si la fecha de fin es anterior a la de inicio, el sistema muestra un mensaje de error. 5b. Si los datos están incompletos, el sistema solicita su corrección.
Nivel de Importancia	**Alto**
Postcondiciones	El usuario queda asociado a una afiliación válida que determina su pertenencia temporal al grupo de investigación.

Gestión de accesos a servicios

La Tabla 3.3 presenta la gestión de accesos a servicios asociada a un usuario, mostrando el estado de cada acceso y permitiendo su concesión o revocación de forma individualizada desde la interfaz de administración.

Tabla 3.3: Requisito funcional RF-03 – Gestión de accesos a servicios

Campo	**Descripción**
Nombre	RF-03 – Gestión de accesos a servicios
Autor	Jaime Martín Borregón
Fecha	30/12/2025

Descripción	Permitir al administrador conceder o revocar el acceso de un usuario a los distintos servicios gestionados por la plataforma, registrando el estado de cada acceso de forma individualizada.
Actores	Administrador del sistema
Precondiciones	El usuario debe existir en el sistema y disponer, en su caso, de una afiliación activa.
Flujo Normal	1. El administrador accede a la vista de servicios de un usuario. 2. El sistema muestra el listado de servicios disponibles y su estado actual. 3. El administrador selecciona conceder o revocar un acceso. 4. El sistema registra la operación solicitada. 5. El sistema actualiza el estado del acceso al servicio.
Flujo Alternativo	4a. Si la operación no puede realizarse, el sistema informa del error. 4b. Si el acceso ya se encuentra en el estado solicitado, no se realiza ninguna acción.
Nivel de Importancia	**Alto**
Postcondiciones	El estado de acceso del usuario al servicio queda actualizado y registrado en el sistema.

Gestión de documentación asociada a usuarios

La Tabla 3.4 muestra la sección de gestión de documentación asociada a un usuario, desde la que es posible subir, consultar y eliminar documentos, incorporando información descriptiva que facilita su identificación.

Tabla 3.4: Requisito funcional RF-04 – Gestión de Documentación

Campo	**Descripción**
Nombre	RF-04 – Gestión de documentación asociada a usuarios
Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	Permitir al administrador asociar documentación relevante a un usuario, facilitando su subida, consulta y eliminación desde la plataforma.
Actores	Administrador del sistema
Precondiciones	El usuario debe estar previamente registrado en el sistema.
Flujo Normal	1. El administrador accede a la vista de detalle de un usuario. 2. El sistema muestra la sección de documentación asociada. 3. El administrador selecciona la opción de añadir un nuevo documento. 4. El administrador introduce un texto identificativo y selecciona el archivo a subir. 5. El sistema almacena el documento y lo asocia al usuario correspondiente.

	6. El documento queda disponible para su consulta desde la plataforma.
Flujo Alternativo	4a. Si no se selecciona ningún archivo o el formato no es válido, el sistema muestra un mensaje de error. 5a. Si el administrador elimina un documento, el sistema lo desvincula del usuario y lo elimina del almacenamiento.
Nivel de Importancia	**Medio**
Postcondiciones	El usuario dispone de documentación asociada accesible desde su vista de detalle.

Automatización basada en fechas de afiliación

La Tabla 3.5 muestra el registro de tareas recientes del sistema, utilizado para aportar trazabilidad sobre las operaciones de provisión de servicios solicitadas o programadas para el usuario.

Tabla 3.5: : Requisito funcional RF-05 – Automatización basada en fechas de Afiliación

Campo	**Descripción**
Nombre	RF-05 – Automatización basada en fechas de afiliación
Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	Permitir al sistema automatizar determinadas acciones en función de las fechas de inicio y fin de las afiliaciones de los usuarios, facilitando la gestión del ciclo de vida de los accesos.
Actores	Sistema
Precondiciones	El usuario debe disponer de al menos una afiliación con fechas de inicio y/o fin definidas.
Flujo Normal	1. El sistema revisa periódicamente las afiliaciones registradas. 2. El sistema identifica afiliaciones próximas a su fecha de finalización o ya caducadas. 3. El sistema registra las acciones correspondientes como tareas internas. 4. El sistema actualiza el estado de las afiliaciones cuando corresponde.
Flujo Alternativo	2a. Si no existen afiliaciones con fechas relevantes, el sistema no realiza ninguna acción. 3a. Si se produce un error durante la generación de tareas, el sistema lo registra para su posterior revisión.
Nivel de Importancia	**Medio**
Postcondiciones	Las acciones relacionadas con el ciclo de vida de las afiliaciones quedan registradas y gestionadas de forma automatizada.

3.4 Requisitos no funcionales

Además de los requisitos funcionales descritos en la sección anterior, el sistema debe cumplir una serie de requisitos no funcionales que condicionan su diseño, implementación y uso. Estos requisitos no describen funcionalidades concretas, sino características de calidad y restricciones del sistema.

A continuación se describen los principales requisitos no funcionales del sistema, utilizando la misma plantilla empleada para los requisitos funcionales:

Seguridad del sistema

En la Tabla 3.6 se recoge el requisito no funcional relativo a la seguridad del sistema, en el que se establecen las condiciones necesarias para garantizar que la gestión de accesos y operaciones no comprometa la infraestructura real del grupo de investigación.

Tabla 3.6: Requisito no funcional RNF-01 – Seguridad del sistema

Campo	**Descripción**
Nombre	RNF-01 – Seguridad del sistema
Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	El sistema debe garantizar que la gestión de usuarios y accesos no comprometa la seguridad de la infraestructura real del grupo de investigación.
Actores	Sistema
Precondiciones	El sistema se encuentra operativo y gestiona solicitudes relacionadas con usuarios y servicios.
Flujo Normal	1. El sistema registra las solicitudes de gestión de accesos como tareas internas. 2. Las operaciones se gestionan de forma controlada sin ejecutarse directamente sobre servicios reales. 3. Las acciones quedan registradas para su supervisión.
Flujo Alternativo	No aplica.
Nivel de Importancia	**Alto**
Postcondiciones	Las operaciones quedan registradas sin afectar a sistemas productivos ni a la infraestructura real.

Trazabilidad de las operaciones

La Tabla 3.7 describe el requisito no funcional asociado a la trazabilidad de las operaciones realizadas en el sistema, definiendo la necesidad de registrar y consultar el estado de las acciones relacionadas con la provisión de servicios.

Tabla 3.7: Requisito no funcional RNF-02 - Trazabilidad de las operaciones

Campo	**Descripción**
Nombre	RNF-02 – Trazabilidad de las operaciones

Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	El sistema debe mantener un registro detallado de las operaciones realizadas sobre los usuarios y sus accesos a servicios.
Actores	Sistema
Precondiciones	Existen usuarios y operaciones de gestión dentro del sistema.
Flujo Normal	1. El sistema registra cada operación solicitada como una tarea interna. 2. Se almacena información sobre el tipo de operación, el usuario afectado y su estado. 3. El historial de operaciones queda disponible para su consulta desde la interfaz.
Flujo Alternativo	2a. Si se produce un error durante el registro de la operación, el sistema almacena la información del fallo.
Nivel de Importancia	**Alto**
Postcondiciones	El sistema dispone de un historial de operaciones que permite la supervisión y auditoría de las acciones realizadas.

Usabilidad de la interfaz

En la Tabla 3.8 se detalla el requisito no funcional de usabilidad, orientado a garantizar que la interfaz de usuario resulte clara, intuitiva y adecuada para tareas administrativas de gestión de usuarios y servicios.

Tabla 3.8: Requisito no funcional RNF-03 – Usabilidad de la interfaz

Campo	**Descripción**
Nombre	RNF-03 – Usabilidad de la interfaz
Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	La plataforma debe ofrecer una interfaz clara e intuitiva que facilite la gestión administrativa de usuarios y servicios.
Actores	Administrador del sistema
Precondiciones	El administrador accede a la plataforma a través de un navegador web.
Flujo Normal	1. El sistema presenta una interfaz coherente y homogénea entre vistas. 2. Las operaciones habituales pueden realizarse de forma directa y comprensible. 3. El administrador recibe información clara sobre el estado del sistema.
Flujo Alternativo	No aplica.
Nivel de Importancia	**Medio**
Postcondiciones	El administrador puede utilizar la plataforma de forma eficiente y sin necesidad de formación específica.

Extensibilidad del sistema

La Tabla 3.9 presenta el requisito no funcional relativo a la extensibilidad del sistema, estableciendo las condiciones que permiten incorporar nuevos servicios o funcionalidades con un impacto mínimo sobre la arquitectura existente.

Tabla 3.9: Requisito no funcional RNF-04 – Extensibilidad del sistema

Campo	**Descripción**
Nombre	RNF-04 – Extensibilidad del sistema
Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	El sistema debe permitir la incorporación de nuevos servicios o funcionalidades con cambios mínimos en la arquitectura existente.
Actores	Sistema
Precondiciones	El sistema cuenta con una arquitectura modular y estructurada en capas.
Flujo Normal	1. El sistema define interfaces y servicios desacoplados. 2. Se incorporan nuevos servicios reutilizando la lógica existente. 3. La funcionalidad se integra sin afectar al resto del sistema.
Flujo Alternativo	No aplica.
Nivel de Importancia	**Medio**
Postcondiciones	El sistema puede evolucionar incorporando nuevos servicios de forma controlada.

Separación de responsabilidades

En la Tabla 3.10 se define el requisito no funcional relacionado con la mantenibilidad del sistema, haciendo énfasis en la separación de responsabilidades entre las distintas capas y componentes de la aplicación.

Tabla 3.10: Requisito no funcional RNF-05 – Separación de responsabilidades

Campo	**Descripción**
Nombre	RNF-05 – Separación de responsabilidades
Autor	Jaime Martín Borregón
Fecha	30/12/2025
Descripción	La arquitectura del sistema debe garantizar una separación clara entre las capas de presentación, lógica de negocio y acceso a datos.
Actores	Sistema
Precondiciones	El sistema sigue una arquitectura web basada en capas.
Flujo Normal	1. Los controladores gestionan la interacción con la interfaz. 2. Los servicios implementan la lógica de negocio. 3. Los repositorios gestionan el acceso a datos.

Flujo Alternativo	No aplica.
Nivel de Importancia	**Medio**
Postcondiciones	El código del sistema es más mantenible, comprensible y fácil de extender.

4 Diseño del sistema

4.1 Arquitectura del sistema

La plataforma desarrollada sigue una **arquitectura web basada en el patrón Modelo–Vista–Controlador (MVC)**, ampliamente utilizado en aplicaciones web por su capacidad para separar responsabilidades y facilitar el mantenimiento del sistema.

Esta arquitectura permite diferenciar claramente entre la gestión de la interfaz de usuario, la lógica de negocio y el acceso a los datos, lo que resulta especialmente adecuado para una plataforma orientada a la administración de usuarios y servicios.

De forma general, la arquitectura del sistema se organiza en las siguientes capas principales:

- **Capa de presentación (Vista)**: responsable de la interacción con el usuario y de la representación de la información.
- **Capa de control (Controlador)**: actúa como intermediaria entre la vista y la lógica de negocio, gestionando las peticiones recibidas.
- **Capa de lógica de negocio (Modelo / Servicios)**: implementa las reglas de negocio del sistema y coordina las operaciones sobre los datos.
- **Capa de persistencia (Repositorios)**: gestiona el acceso a la base de datos y el almacenamiento de la información.

A continuación se describen con mayor detalle cada una de estas capas y su papel dentro de la arquitectura del sistema.

La Figura 4.1 muestra la arquitectura por capas basada en MVC. La capa de presentación implementa las vistas con Thymeleaf; los controladores (Spring MVC) orquestan las peticiones; la lógica reside en servicios; la persistencia se abstrae mediante repositorios JPA sobre SQLite. Las integraciones con servicios externos se encapsulan en conectores, invocados desde la capa de servicios, lo que permite desacoplar el core del sistema de la tecnología concreta de cada servicio.

Figura 4.1: Arquitectura general de la plataforma de gestión de usuarios y servicios.

4.1.1 Capa de presentación

La capa de presentación está formada por la interfaz web de la plataforma, desarrollada mediante plantillas HTML renderizadas en el servidor. Esta capa es responsable de mostrar la información al administrador y de recoger las acciones realizadas desde la interfaz.

Su función se limita exclusivamente a la representación de datos y a la captura de eventos de usuario, evitando la inclusión de lógica de negocio. Este enfoque contribuye a mantener una separación clara de responsabilidades y facilita la evolución del diseño de la interfaz sin afectar al resto del sistema.

4.1.2 Capa de controladores

La capa de controladores actúa como punto de entrada de las peticiones realizadas desde la interfaz web. Los controladores reciben las solicitudes HTTP, validan los parámetros recibidos y delegan la ejecución de la lógica correspondiente en la capa de servicios.

Cada controlador está asociado a un conjunto concreto de funcionalidades (usuarios, afiliaciones, servicios, documentación, etc.), lo que permite estructurar el código de forma modular y comprensible. Una vez procesada la solicitud, el controlador devuelve la respuesta adecuada a la capa de presentación.

4.1.3 Capa de servicios (lógica de negocio)

La lógica principal del sistema se concentra en la capa de servicios. Esta capa es responsable de implementar las reglas de negocio asociadas a la gestión de usuarios, afiliaciones y accesos a servicios.

Entre sus responsabilidades se incluyen la validación de operaciones, el cálculo del estado de las afiliaciones en función de las fechas definidas y la coordinación de la creación de tareas internas relacionadas con la provisión de servicios. Al centralizar esta lógica en una capa independiente, se evita su dispersión en los controladores y se mejora la mantenibilidad del sistema.

4.1.4 Capa de persistencia

El acceso a los datos se realiza a través de una capa de persistencia basada en repositorios. Esta capa abstrae la interacción con la base de datos, permitiendo realizar operaciones de consulta y modificación sin acoplar el resto del sistema a los detalles de almacenamiento.

Este diseño facilita la evolución del modelo de datos y permite adaptar la aplicación a distintos motores de base de datos con cambios mínimos en el código.

4.2 Diseño de la integración con servicios

En este apartado se describe el diseño propuesto para la integración de la plataforma con los distintos servicios utilizados por el grupo de investigación.

Aunque no todas las integraciones han podido implementarse de forma completamente operativa durante el desarrollo del proyecto, se detalla cómo se llevarían a cabo en un entorno real, las tecnologías implicadas y las limitaciones encontradas.

El objetivo de este diseño es demostrar la viabilidad técnica de la solución y justificar las decisiones adoptadas a lo largo del trabajo, especialmente en aquellos casos en los que el acceso a servicios reales no ha sido posible por restricciones de permisos o seguridad.

En la Figura 4.2 se resume la infraestructura física y lógica del laboratorio, incluyendo los servidores y los elementos de red implicados en la provisión de servicios.

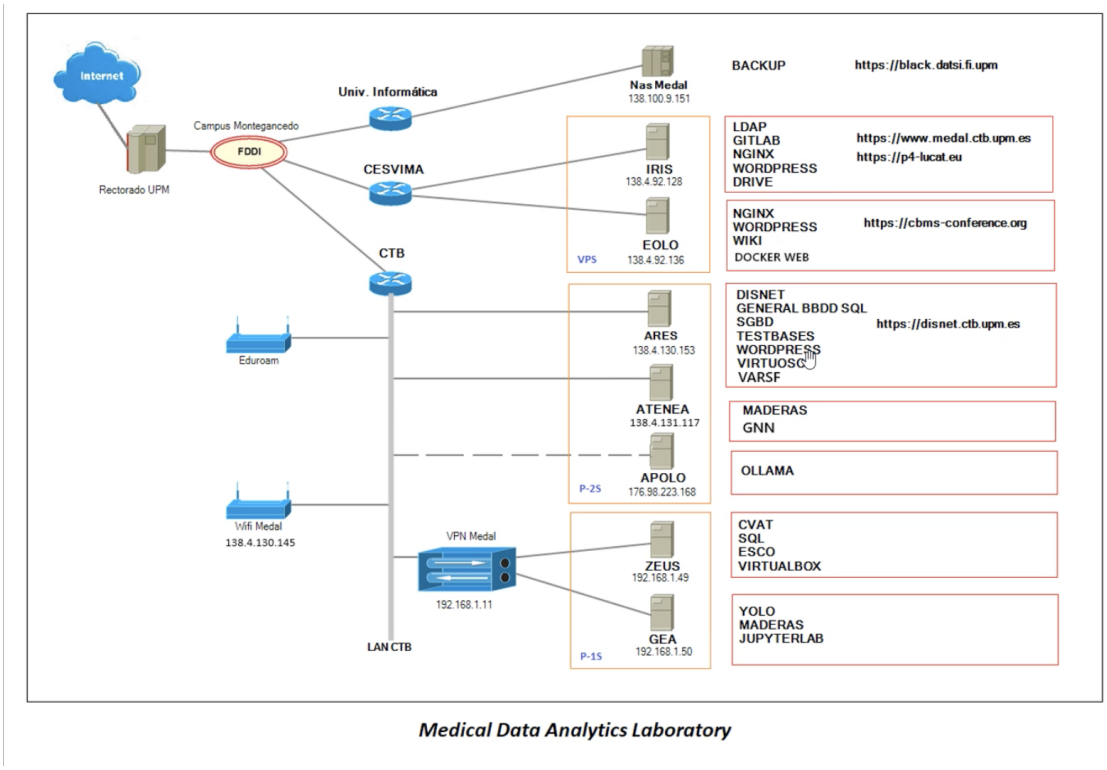

Figura 4.2: Infraestructura del laboratorio

Esta figura se utiliza como referencia para justificar las decisiones de diseño adoptadas en la integración con los distintos servicios, así como la necesidad de mecanismos de control como la restricción de hosts, el uso de cuentas técnicas y la automatización del ciclo de vida de accesos.

4.2.1 Servicio de directorio e identidad (LDAP)

Objetivo de la integración

El servicio LDAP constituye el **repositorio central de identidades** del laboratorio. La integración con LDAP permite automatizar la **creación, modificación y eliminación de usuarios**, así como su **asignación a grupos**, de forma coherente con el ciclo de vida de las afiliaciones gestionadas por la plataforma.

Esta integración es especialmente relevante, ya que otros servicios del laboratorio (por ejemplo, GitLab) dependen directa o indirectamente de la existencia del usuario en LDAP, por lo que este servicio actúa como **fuente primaria de identidad**.

Tecnología empleada y justificación

Para la integración LDAP desde Java se propone el uso de la **UnboundID LDAP SDK**, una librería ampliamente utilizada, ligera y orientada a operaciones programáticas directas sobre servidores LDAP.

Las razones principales para esta elección son:

- API de bajo nivel, clara y explícita, adecuada para describir el funcionamiento real de LDAP.
- Soporte completo de operaciones estándar: add, modify, delete, bind.
- Uso habitual en entornos corporativos y académicos.

La integración se implementa mediante una **cuenta técnica LDAP** con permisos administrativos **limitados**, configurada exclusivamente para realizar las operaciones necesarias sobre las ramas del directorio correspondientes a los usuarios y grupos del laboratorio.

Autenticación y gestión de credenciales

La autenticación contra el servidor LDAP se realiza mediante un **simple bind**, utilizando:

- Un **Bind DN** (en este caso, cn=tfgadmin, ou=system, dc=lab, dc=upm, dc=es)
- Una contraseña asociada a dicha cuenta técnica.

Estas credenciales:

- **No se almacenan en el código fuente**
- Se inyectan en tiempo de ejecución mediante **variables de entorno** o configuración externa. En este caso dejamos todo preparado en nuestro 'application.yml' como se puede ver en el Listado 4.1, y las credenciales/contraseñas desde variables de entorno en el terminal, antes de iniciar el Spring Boot, como vemos en el Listado 4.2.
- **Están restringidas por ACLs del servidor LDAP** (*Access Control Lists*, reglas de control de acceso) **para limitar de forma granular** qué operaciones (lectura/búsqueda/alta/modificación/borrado) puede realizar la cuenta técnica y sobre qué ramas/atributos del directorio, evitando accesos fuera del ámbito del proyecto.

Este enfoque permite simular un entorno realista de producción sin comprometer la seguridad del directorio.

```
# ============================
# Configuración propia LDAP provisioning
# ============================
ldap:
  url: "ldap://medal.ctb.upm.es:389"
  base: "dc=medal,dc=ctb,dc=upm,dc=es"
  bindDn: "cn=admin,dc=medal,dc=ctb,dc=upm,dc=es"
  bindPassword: "<valor_real_en_variables_de_entorno>"
  usersOu: "ou=Gitlab"
  loginAttr: "uid"
  defaultGidNumber: "500"
  defaultShell: "/bin/bash"
  homeBase: "/home/users"
  hash: "md5"
```

Listado 4.1: fragmento del yml para LDAP

El Listado 4.2 muestra las credenciales que habría que añadir en nuestro terminal antes de arrancar el Spring Boot. Por motivos de seguridad, son secretas en esta memoria.

```
$env:LDAP_HOST="172.30.x.x"
$env:LDAP_PORT="389"
$env:LDAP_BIND_DN="cn=tfgadmin,ou=system,dc=lab,dc=upm,dc=es"
$env:LDAP_BIND_PASSWORD="***"
$env:LDAP_BASE_DN="dc=lab,dc=upm,dc=es"
```

Listado 4.2: credenciales en PowerShell, Windows

Operaciones LDAP implementadas

1. **Alta de usuario en LDAP**

La operación de alta de usuario consta de **dos pasos claramente diferenciados**:

1) Creación de la entrada LDAP del usuario
2) Asignación del usuario a uno o varios grupos LDAP

El Listado 4.3 muestra un fragmento de código representativo que ilustra la operación para dar de alta un usuario utilizando la librería UnboundID LDAP SDK.

```
import com.unboundid.ldap.sdk.*;

public class LdapProvisioningConnector {

    private final String host;
    private final int port;
    private final String bindDn;
    private final String bindPassword;

    public LdapProvisioningConnector(String host, int port, String bindDn, String
bindPassword) {
        this.host = host;
        this.port = port;
        this.bindDn = bindDn;
```

```
        this.bindPassword = bindPassword;
    }

    public void createUserAndAddToGroup(
            String userDn,
            String uid,
            String cn,
            String sn,
            String mail,
            String groupDn
    ) throws LDAPException {

        try (LDAPConnection conn = new LDAPConnection(host, port, bindDn,
bindPassword)) {

            // 1) Crear entrada del usuario
            Entry user = new Entry(userDn);
            user.addAttribute("objectClass",
                    "top",
                    "person",
                    "organizationalPerson",
                    "inetOrgPerson"
            );
            user.addAttribute("uid", uid);
            user.addAttribute("cn", cn);
            user.addAttribute("sn", sn);
            user.addAttribute("mail", mail);

            conn.add(new AddRequest(user));

            // 2) Añadir usuario al grupo LDAP
            Modification addMember =
                    new Modification(ModificationType.ADD, "member", userDn);

            conn.modify(new ModifyRequest(groupDn, addMember));
        }
    }
}
```

Listado 4.3: alta y asignación a grupo en LDAP

2. Operación inversa: baja de usuario en LDAP

Tal y como exige el diseño del sistema, toda operación de alta tiene definida su operación opuesta. En el caso de LDAP, la baja de un usuario implica:

1. Eliminar la pertenencia del usuario a los grupos LDAP
2. Eliminar la entrada LDAP del usuario

Este proceso es especialmente relevante cuando:

- La afiliación del usuario expira
- Se produce una baja manual desde la interfaz de administración

El Listado 4.4 muestra un fragmento de código representativo que ilustra la operación para dar de baja un usuario utilizando la librería UnboundID LDAP SDK.

```
public void removeUserFromGroupAndDelete(String userDn, String groupDn) throws
LDAPException {
```

```
    try (LDAPConnection conn = new LDAPConnection(host, port, bindDn, bindPassword))
{

        // 1) Eliminar pertenencia al grupo
        Modification removeMember =
                new Modification(ModificationType.DELETE, "member", userDn);

        conn.modify(new ModifyRequest(groupDn, removeMember));

        // 2) Eliminar entrada del usuario
        conn.delete(new DeleteRequest(userDn));
    }
}
```

Listado 4.4: baja de usuario en LDAP

Gestión de errores y casos límite

Durante la ejecución de estas operaciones pueden producirse diversos errores, tales como:

- **Entrada duplicada** (entryAlreadyExists) al crear un usuario ya existente
- **Permisos insuficientes** (insufficientAccessRights) si la cuenta técnica no dispone de ACLs adecuadas
- **No existencia del usuario o grupo** durante la operación de baja

Estos errores son capturados y registrados por el sistema, asociándose a una entidad de tipo ProvisioningTask, lo que permite:

- Auditoría completa de las operaciones
- Reintentos controlados
- Diagnóstico de fallos sin intervención manual directa

Integración con la lógica del sistema

Las operaciones LDAP no se invocan directamente desde los controladores, sino que se encapsulan en un **conector de integración** que es utilizado por la capa de servicios.

La ejecución efectiva de estas operaciones se realiza a través de:

- Tareas de provisión (ProvisioningTask)
- Procesadas por jobs programados que revisan:
    - Fechas de inicio de afiliación (alta)
    - Fechas de fin de afiliación (baja)

Este diseño permite desacoplar la lógica de negocio de los detalles específicos del protocolo LDAP y facilita la extensión del sistema a otros servicios.

Como evidencia del funcionamiento de los temporizadores y notificaciones, la Figura 4.3 muestra un ejemplo del correo al usuario llamado 'TEST', generado automáticamente cuando la afiliación de un usuario entra en el umbral de aviso (7 días antes de su expiración).

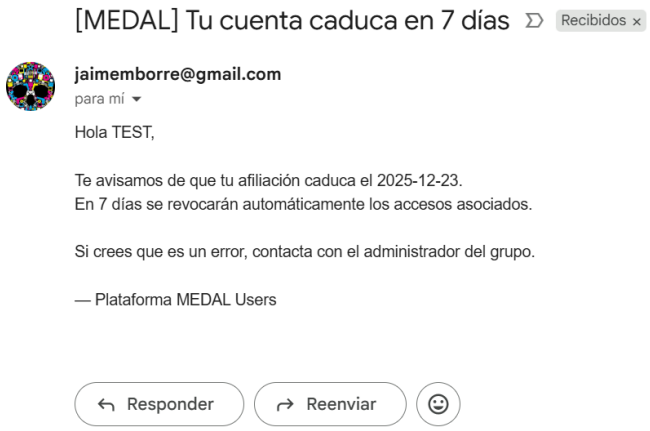

Figura 4.3: captura del aviso de fin de afiliación

Consideraciones de seguridad

Dado que las operaciones LDAP pueden ser críticas, el sistema incorpora las siguientes medidas:

- Uso de cuentas técnicas con permisos mínimos
- Configuración de un modo **dry-run** para entornos de prueba
- Registro detallado de todas las operaciones realizadas o simuladas
- Posibilidad de desactivar temporalmente la ejecución real sin modificar el código

4.2.2 Integración con servidores Linux mediante SSH

Objetivo de la integración

El objetivo de la integración con los servidores Linux del laboratorio es automatizar la gestión del acceso de los usuarios a los sistemas computacionales, permitiendo:

- Alta de cuentas de usuario en los servidores.
- Asignación a grupos del sistema.
- Definición de fechas de expiración asociadas a la afiliación.
- Desactivación o eliminación de cuentas cuando la afiliación expira o se produce una baja manual.

Esta integración resulta clave para mantener la coherencia entre el estado interno del sistema y los accesos reales a la infraestructura.

Tecnología empleada y justificación

La comunicación con los servidores se realiza mediante el protocolo **SSH**, ampliamente utilizado para la administración remota de sistemas Unix/Linux.

Para su uso desde Java se propone la librería **Apache MINA SSHD**, una implementación moderna y mantenida que permite:

- Establecer sesiones SSH seguras.
- Ejecutar comandos remotos.
- Capturar salida estándar y errores.
- Integrarse fácilmente en aplicaciones Java sin dependencias externas del sistema operativo.

Esta librería permite ilustrar de forma clara cómo se ejecutarían realmente las operaciones administrativas desde la plataforma.

Modelo de seguridad y cuenta técnica

Las operaciones administrativas sobre los servidores se realizan a través de una **cuenta técnica dedicada** (en mi caso, tfgadmin), configurada con las siguientes restricciones:

- Acceso SSH limitado únicamente desde el servidor donde se ejecuta la plataforma.
- Uso de un fichero sudoers con permisos **NOPASSWD** exclusivamente para un conjunto reducido de comandos (useradd, usermod, userdel, chage).
- Aplicación de una **"allowlist" de hosts**, restringiendo la ejecución únicamente a los servidores del laboratorio definidos en la configuración.

Este modelo minimiza el impacto de un posible compromiso y se ajusta a prácticas habituales en entornos de producción.

Ejecución remota de comandos desde el orquestador

La ejecución de operaciones administrativas sobre los servidores Linux no se realiza de forma manual ni interactiva. Todas las acciones se lanzan desde el backend de la plataforma, que actúa como orquestador del ciclo de vida de los usuarios.

Desde el código Java del backend, el sistema establece una conexión SSH saliente hacia el servidor objetivo utilizando la librería Apache MINA SSHD. Una vez autenticada la sesión mediante la cuenta técnica configurada, el orquestador abre un canal de ejecución remota (exec channel) que permite lanzar comandos o scripts administrativos de forma no interactiva.

Este mecanismo permite al backend:
- ejecutar comandos del sistema como `useradd`, `usermod`, `userdel` o `chage`,
- capturar la salida estándar, los errores y el código de retorno,
- y registrar el resultado de cada operación en la tarea de provisión correspondiente.

De este modo, el backend mantiene el control completo sobre cuándo, cómo y con qué resultado se ejecutan las operaciones, sin requerir intervención manual directa sobre los servidores.

Alta de usuarios en servidores Linux

La creación de un usuario en un servidor Linux se realiza mediante la ejecución remota del comando `useradd`.

Un aspecto especialmente relevante es la **definición de la fecha de expiración** de la cuenta, que se establece utilizando el parámetro `-e`, de acuerdo con la fecha de fin de afiliación introducida en la interfaz de la plataforma.

Esto permite que el propio sistema operativo bloquee automáticamente la cuenta cuando dicha fecha se alcanza.

El Listado 4.5 muestra un fragmento de código representativo que ilustra cómo la plataforma establece una conexión SSH con un servidor Linux y ejecuta comandos administrativos para la creación de usuarios. El objetivo del listado no es presentar una implementación completa, sino evidenciar el uso de una librería real y el flujo técnico necesario para llevar a cabo estas operaciones desde una aplicación Java.

```
import org.apache.sshd.client.SshClient;
import org.apache.sshd.client.channel.ClientChannel;
import org.apache.sshd.client.channel.ClientChannelEvent;
import org.apache.sshd.client.session.ClientSession;

import java.io.ByteArrayOutputStream;
import java.nio.charset.StandardCharsets;
import java.time.LocalDate;
import java.util.EnumSet;

public class SshCommandExecutor {

    public record CommandResult(int exitCode, String stdout, String stderr) {}

    /**
     * Ejecuta un comando remoto en el servidor, capturando stdout/stderr y exit
code.
     */
    public CommandResult exec(ClientSession session, String command) throws Exception
{
        try (ByteArrayOutputStream out = new ByteArrayOutputStream();
             ByteArrayOutputStream err = new ByteArrayOutputStream();
             ClientChannel channel = session.createExecChannel(command)) {

            channel.setOut(out);
            channel.setErr(err);

            channel.open().verify();
            channel.waitFor(EnumSet.of(ClientChannelEvent.CLOSED), 30_000); //
timeout 30s

            Integer status = channel.getExitStatus();
            int exit = (status != null) ? status : -1;

            return new CommandResult(
                    exit,
                    out.toString(StandardCharsets.UTF_8),
                    err.toString(StandardCharsets.UTF_8)
            );
        }
    }

    /**
     * Ejemplo: alta y baja de un usuario Linux desde el orquestador mediante SSH.
     */
    public void provisionLinuxUser(String host, int port, String adminUser, String
password,
```

```
                                    String username, LocalDate expiryDate) throws
Exception {

        SshClient client = SshClient.setUpDefaultClient();
        client.start();

        try (ClientSession session = client.connect(adminUser, host, port)
                                          .verify(30_000)
                                          .getSession()) {

            session.addPasswordIdentity(password);
            session.auth().verify(30_000);

            // Alta (comando directo) con fecha de expiración
            CommandResult create = exec(session,
                    "sudo useradd -m -e " + expiryDate + " " + username);

            // Alternativa equivalente: invocar un script wrapper de provisión (si se
prefiere)
            // CommandResult create = exec(session,
            //         "sudo /usr/local/bin/provision_user.sh --create " + username +
            //         " --expires " + expiryDate);

            // Comprobaciones: existencia de usuario + expiración configurada
            CommandResult checkId = exec(session, "id " + username);
            CommandResult checkExpiry = exec(session, "sudo chage -l " + username);

            if (create.exitCode() != 0) {
                throw new IllegalStateException("Fallo creando usuario: " +
create.stderr());
            }

            // Operación inversa (revocación): ejemplo de desactivación/borrado
            CommandResult disable = exec(session, "sudo usermod -L " + username);
            // CommandResult delete = exec(session, "sudo userdel -r " + username);

            // El orquestador registraría stdout/stderr/exitCode en la
ProvisioningTask para trazabilidad y auditoría.
        } finally {
            client.stop();
        }
    }
}
```

Listado 4.5: Conexión SSH desde el orquestador y ejecución remota de comandos/scripts (Apache MINA SSHD)

Tras la ejecución, el resultado de la operación se refleja en el propio sistema mediante la actualización del estado local del acceso: la tarea de provisión (ProvisioningTask) se marca como completada o fallida y, en caso de éxito, se actualiza el registro de acceso del usuario al servidor (p. ej., entidad de asignación/estado de servidor) para que la interfaz muestre de forma consistente si el usuario dispone de cuenta activa en dicho host. De este modo se mantiene coherencia entre la infraestructura y el estado persistido en la plataforma.

Operación inversa: desactivación y eliminación de usuarios

De acuerdo con el diseño del sistema, la plataforma contempla **dos estrategias de revocación**, seleccionables según la política del laboratorio:

1. **Desactivación (reversible)**
2. **Eliminación (destructiva)**

La operación inversa (revocación) se ejecuta reutilizando el mismo conector SSH mostrado en el Listado 4.4. En función de la política definida, el backend puede (i) desactivar la cuenta mediante usermod -L, o (ii) eliminarla completamente mediante userdel -r. Además, aunque useradd -e impide el acceso al alcanzar la fecha de expiración, el sistema genera igualmente una tarea de revocación al expirar la afiliación para cerrar el ciclo de vida y dejar evidencia auditada de la operación.

Integración con la lógica del sistema

Las operaciones sobre los servidores no se ejecutan directamente desde la interfaz de usuario. En su lugar:

- La plataforma genera **tareas de provisión** asociadas a cada afiliación.
- Un proceso programado revisa periódicamente las fechas de expiración.
- Al alcanzarse la fecha de fin, se genera automáticamente una tarea de revocación.
- Dicha tarea ejecuta la desactivación o eliminación del usuario en los servidores correspondientes.

Este enfoque desacopla la lógica de negocio de la ejecución concreta sobre los sistemas y permite una trazabilidad completa de las acciones realizadas.

Caso práctico: servidor CVAT

Durante el desarrollo del proyecto, se realizó una integración real con el servidor CVAT del laboratorio, utilizado como plataforma de anotación de datos. Este servidor fue empleado como entorno de pruebas para validar la viabilidad de la integración mediante SSH descrita en este apartado.

En este caso, la plataforma establecía conexión SSH con el servidor CVAT utilizando una cuenta técnica previamente configurada, desde la cual se ejecutaban comandos administrativos para la gestión de usuarios del sistema. Estas operaciones permitieron verificar en la práctica:

- La correcta autenticación remota mediante SSH.
- La ejecución de comandos administrativos desde la aplicación.
- La creación y verificación de cuentas de usuario en un servidor real.
- La correcta aplicación de permisos y restricciones del sistema.

Este caso práctico sirvió para validar el enfoque general adoptado para la integración con servidores Linux, confirmando que la automatización propuesta es técnicamente viable en entornos reales del laboratorio.

Por motivos de seguridad y alcance del proyecto, esta integración se limitó a un servidor concreto (CVAT), empleándose el resto de servidores únicamente a nivel teórico, aunque siguiendo exactamente el mismo modelo de diseño y ejecución.

Gestión de errores y auditoría

Durante la ejecución de comandos remotos pueden producirse distintos errores:

- Usuario ya existente.
- Permisos insuficientes en sudoers.
- Host inaccesible o fallo de autenticación SSH.

Todos los errores y salidas de los comandos se registran y se asocian a la tarea correspondiente, permitiendo:

- Auditoría posterior.
- Diagnóstico de incidencias.
- Reintentos controlados sin intervención manual directa.

Consideraciones de seguridad adicionales

Dado el carácter sensible de estas operaciones, el sistema incorpora:

- Modo **dry-run**, que permite simular la ejecución sin aplicar cambios reales.
- Registro detallado de cada comando ejecutado o simulado.
- Separación estricta entre entorno de pruebas y entorno real.

4.2.3 Integración con GitLab

Objetivo de la integración

El servicio GitLab se utiliza en el laboratorio como plataforma de control de versiones y colaboración. La integración con GitLab tiene como objetivo **automatizar la gestión de accesos** de los usuarios a los distintos grupos y proyectos, en coherencia con su afiliación al grupo de investigación.

Es importante destacar que, en el entorno del laboratorio, **GitLab está integrado con LDAP**, por lo que la plataforma **no gestiona la creación de identidades en GitLab**, sino exclusivamente la **asignación y revocación de permisos** sobre grupos y proyectos.

Tecnología empleada y justificación

Para la integración con GitLab desde Java se propone el uso de la librería **GitLab4J API**, un cliente oficial no mantenido por GitLab pero ampliamente adoptado, que encapsula las llamadas a la API REST de GitLab y simplifica su uso en aplicaciones Java.

Las principales ventajas de esta librería son:

- Acceso directo a la API REST oficial de GitLab.
- Soporte para la gestión de usuarios, grupos y proyectos.
- Manejo transparente de autenticación mediante tokens.
- Código más legible y mantenible que las llamadas HTTP manuales.

Autenticación y gestión de credenciales

La autenticación contra GitLab se realiza mediante un **Personal Access Token (PAT)** o un **token de aplicación técnica**, configurado con los permisos mínimos necesarios.

En este caso, el token debe disponer, como mínimo, de los siguientes alcances (*scopes*):

- **api**: para la gestión de miembros en grupos y proyectos.

Al igual que en otras integraciones, el token:

- No se almacena en el código fuente.
- Se define mediante variables de entorno o configuración externa.
- Es gestionado por una cuenta técnica independiente de los usuarios finales.

Este enfoque evita el uso de credenciales personales y permite revocar o rotar el token sin afectar al resto del sistema.

Operaciones sobre GitLab

1. **Alta de usuario en un grupo de GitLab**

La operación principal consiste en **añadir un usuario existente** (sincronizado previamente desde LDAP) a un grupo de GitLab con un determinado nivel de acceso (Developer, Maintainer, etc.).

Para ello, es necesario:

1) Identificar el **ID del grupo** de GitLab.
2) Obtener el **ID interno del usuario** (a partir de su username o email).
3) Añadir el usuario como miembro del grupo con el rol correspondiente.

El Listado 4.6 muestra un fragmento de código representativo utilizando la librería GitLab4J:

```
import org.gitlab4j.api.GitLabApi;
import org.gitlab4j.api.Constants.AccessLevel;
import org.gitlab4j.api.models.User;

public class GitLabProvisioningConnector {

    private final GitLabApi gitLabApi;

    public GitLabProvisioningConnector(String gitlabUrl, String accessToken) {
        this.gitLabApi = new GitLabApi(gitlabUrl, accessToken);
    }

    public void addUserToGroup(int groupId, String username, AccessLevel accessLevel)
            throws Exception {

        // Obtener usuario GitLab a partir del username
        User user = gitLabApi.getUserApi().getUsers(username).get(0);

        // Añadir usuario al grupo con el nivel de acceso indicado
```

```
        gitLabApi.getGroupApi()
                .addMember(groupId, user.getId(), accessLevel);
    }
}
```

Listado 4.6: asignación de usuario a grupo GitLab

2. Operación inversa: revocación de acceso en GitLab

De forma análoga a las integraciones anteriores, la plataforma define explícitamente la **operación inversa** a la asignación de permisos en GitLab.

La revocación de acceso se realiza mediante:

- La eliminación del usuario como miembro del grupo.
- Opcionalmente, el bloqueo del usuario en GitLab, si así lo establece la política del laboratorio.

```
public void removeUserFromGroup(int groupId, int userId) throws Exception {
    gitLabApi.getGroupApi().removeMember(groupId, userId);
}
```

En escenarios donde se requiera una revocación más restrictiva, puede emplearse adicionalmente la operación de bloqueo del usuario:

```
gitLabApi.getUserApi().blockUser(userId);
```

Estas operaciones garantizan que el usuario pierde de forma inmediata el acceso a los repositorios asociados al grupo.

Gestión de errores y casos límite

Durante la interacción con la API de GitLab pueden producirse situaciones como:

- Usuario no encontrado (no sincronizado aún desde LDAP).
- Usuario ya miembro del grupo.
- Permisos insuficientes del token utilizado.
- Grupo inexistente o mal configurado.

Estos errores se capturan y registran en el sistema, quedando asociados a la tarea de provisión correspondiente para su posterior análisis o reintento controlado.

Integración con la lógica del sistema

Al igual que en el resto de servicios, la integración con GitLab se encapsula en un **conector específico**, invocado desde la capa de servicios.

La asignación y revocación de accesos se realiza:

- Automáticamente al inicio de la afiliación.

- Automáticamente al expirar la afiliación.
- Manualmente mediante acciones del administrador desde la interfaz web.

Este diseño garantiza que el estado de los permisos en GitLab sea coherente en todo momento con el estado interno del sistema.

Consideraciones de seguridad

La integración con GitLab incorpora las siguientes medidas:

- Uso exclusivo de cuentas técnicas y tokens dedicados.
- Aplicación del principio de privilegios mínimos.
- Registro detallado de todas las llamadas realizadas a la API.
- Posibilidad de ejecutar la integración en modo **dry-run** durante pruebas.

4.2.4 Integración con redes privadas virtuales (VPN)

Objetivo de la integración

El acceso VPN permite a los usuarios del grupo de investigación conectarse de forma segura a la red interna del laboratorio desde ubicaciones externas. La integración de este servicio en la plataforma tiene como objetivo **gestionar el ciclo de vida de las credenciales VPN** de forma coherente con las afiliaciones de los usuarios, permitiendo:

- Alta de cuentas VPN al inicio de la afiliación.
- Revocación automática del acceso al finalizar dicha afiliación.
- Gestión centralizada y auditada de los accesos remotos.

Infraestructura y sistema de gestión VPN

El servicio VPN del laboratorio se encuentra gestionado por un router **TP-Link ER706W Omada AX3000 Gigabit VPN Router**, que proporciona soporte para múltiples protocolos, entre ellos **OpenVPN**, utilizado en el entorno descrito.

La administración del servicio se realiza a través de una **interfaz web de gestión**, accesible desde la red interna del laboratorio en la URL:

http://192.168.1.11/webpages/login.html

Desde esta interfaz es posible:

- Autenticarse como administrador.
- Crear y eliminar usuarios VPN.
- Asociar cada usuario a un protocolo concreto (en este caso, OpenVPN).
- Consultar el listado de cuentas existentes.

Análisis de mecanismos de integración disponibles

Como parte del diseño de la integración con el servicio VPN, se analizó la posibilidad de gestionar el ciclo de vida de los usuarios mediante mecanismos distintos a la interfaz web de administración.

En primer lugar, se revisó la documentación técnica disponible para el dispositivo TP-Link ER706W Omada AX3000 con el objetivo de identificar la existencia de una API oficial de gestión o de interfaces programáticas soportadas para la administración de usuarios VPN. Tras dicho análisis, no se encontró una API pública documentada que permitiera realizar estas operaciones de forma directa desde una aplicación externa.

Adicionalmente, se estudió la viabilidad de una integración alternativa mediante acceso remoto al dispositivo (por ejemplo, a través de SSH u otros mecanismos de administración de bajo nivel). Sin embargo, este tipo de acceso no se encuentra habilitado ni documentado en el entorno del laboratorio, y su uso podría comprometer la estabilidad y seguridad del sistema, por lo que fue descartado en el contexto de este proyecto.

Como resultado de este análisis, la única vía viable para automatizar la gestión de usuarios VPN consiste en reproducir programáticamente las operaciones realizadas desde la interfaz web de administración, técnica que se detalla en los apartados siguientes.

Alta de usuarios VPN (gestión manual actual)

Actualmente, la creación de usuarios VPN se realiza manualmente desde la interfaz web del router.

Tal y como se muestra en la **Figura 4.3**, el proceso de alta consiste en:

1. Acceder al apartado *VPN -> Users*.
2. Seleccionar la opción *Add*.
3. Introducir:
    - Nombre de la cuenta VPN.
    - Contraseña asociada.
    - Protocolo VPN (OpenVPN).
    - Servidor VPN configurado (por ejemplo, MEDAL_VPN).
4. Confirmar la creación del usuario.

**Este proceso queda reflejado en las capturas aportadas a continuación en la Figura 4.4, donde se observa el formulario de creación y el listado posterior de usuarios VPN activos.*

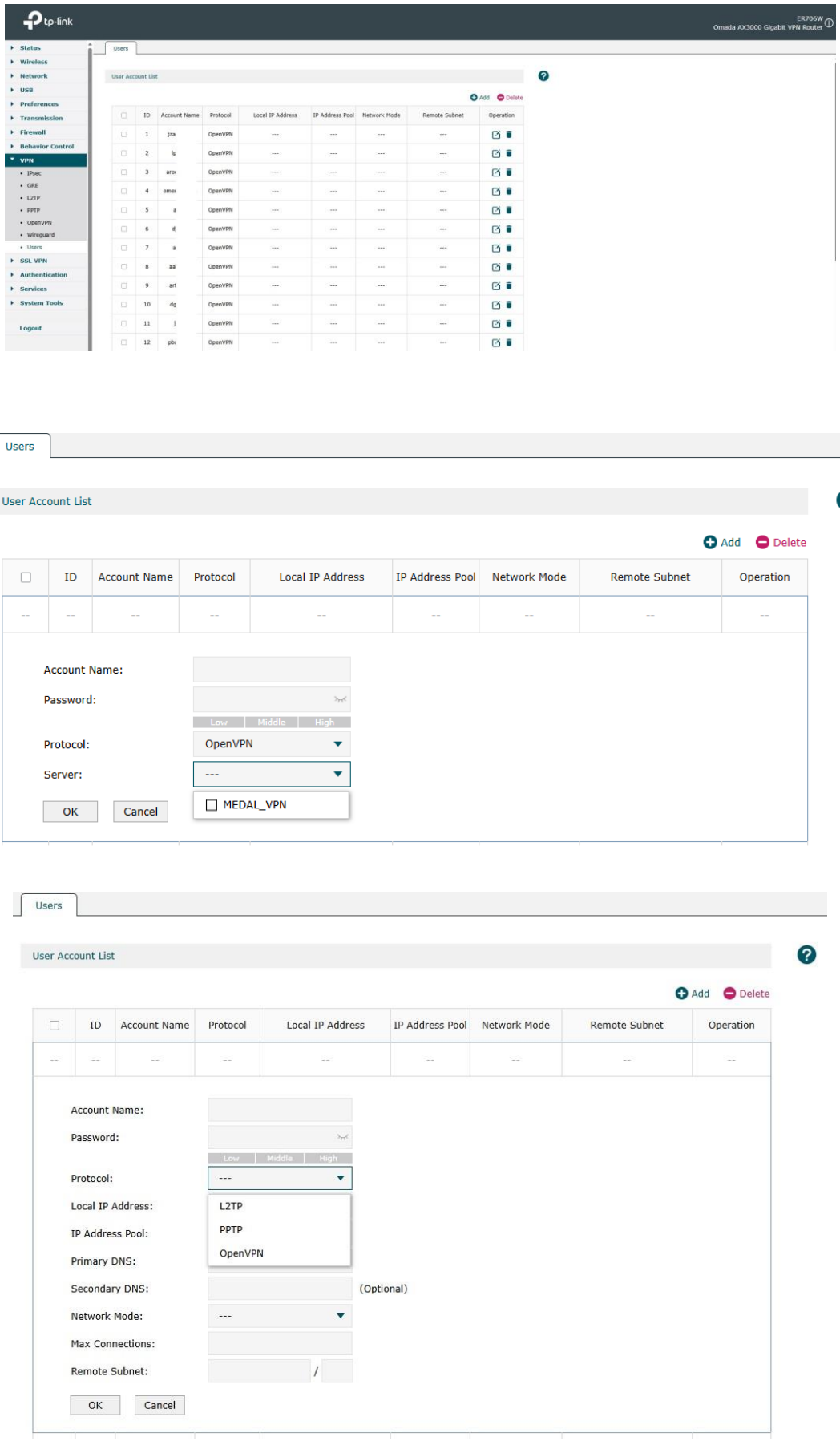

Figura 4.4: proceso de alta manual

Baja de usuarios VPN

La revocación del acceso VPN se realiza desde la misma interfaz de administración, eliminando explícitamente la cuenta del usuario.

El proceso consiste en:

1. Acceder al listado de usuarios VPN.
2. Seleccionar el usuario correspondiente.
3. Ejecutar la operación *Delete*.

Esta operación invalida de forma inmediata las credenciales del usuario, impidiendo nuevas conexiones VPN.

Planteamiento de integración automatizada mediante cliente HTTP

*A continuación se muestra un fragmento de código **teórico pero realista** que ilustra cómo podría automatizarse la gestión de usuarios VPN interactuando directamente con la interfaz web de administración del router.*

El objetivo del listado no es proporcionar una implementación completa ni funcional, sino evidenciar el conocimiento de los mecanismos técnicos necesarios para reproducir programáticamente las operaciones de autenticación y creación/eliminación de usuarios observadas en la interfaz real del sistema.

```
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class VpnProvisioningConnector {

    private final HttpClient client;
    private final String baseUrl;

    public VpnProvisioningConnector(String baseUrl) {
        this.client = HttpClient.newBuilder()
                .followRedirects(HttpClient.Redirect.NORMAL)
                .build();
        this.baseUrl = baseUrl;
    }

    /**
     * Autenticación contra la interfaz web del router VPN.
     * Se establece una sesión HTTP mediante cookies.
     */
    public void login(String username, String password) throws Exception {
        String body = "username=" + username + "&password=" + password;

        HttpRequest request = HttpRequest.newBuilder()
                .uri(new URI(baseUrl + "/webpages/login.html"))
                .header("Content-Type", "application/x-www-form-urlencoded")
                .POST(HttpRequest.BodyPublishers.ofString(body))
                .build();

        client.send(request, HttpResponse.BodyHandlers.discarding());
    }

    /**
     * Creación de un usuario VPN OpenVPN.
```

```
     */
    public void createVpnUser(String accountName, String password) throws Exception {
        String body =
                "accountName=" + accountName +
                "&password=" + password +
                "&protocol=OpenVPN" +
                "&server=MEDAL_VPN";

        HttpRequest request = HttpRequest.newBuilder()
                .uri(new URI(baseUrl + "/vpn/user/add"))
                .header("Content-Type", "application/x-www-form-urlencoded")
                .POST(HttpRequest.BodyPublishers.ofString(body))
                .build();

        client.send(request, HttpResponse.BodyHandlers.ofString());
    }
}
```

Listado 4.7: Autenticación y alta de usuario VPN

En este ejemplo se observa cómo la plataforma podría reproducir el flujo real de la interfaz web: primero estableciendo una sesión autenticada y, a continuación, enviando la petición correspondiente a la creación de un usuario VPN OpenVPN con los parámetros observados en el formulario de administración.

Operación inversa

De forma análoga a la operación de alta, la revocación del acceso VPN se realizaría reproduciendo la petición HTTP asociada a la eliminación de un usuario desde la interfaz web de administración.

```
public void deleteVpnUser(String userId) throws Exception {
    String body = "userId=" + userId;

    HttpRequest request = HttpRequest.newBuilder()
            .uri(new URI(baseUrl + "/vpn/user/delete"))
            .header("Content-Type", "application/x-www-form-urlencoded")
            .POST(HttpRequest.BodyPublishers.ofString(body))
            .build();

    client.send(request, HttpResponse.BodyHandlers.ofString());
}
```

Listado 4.8: Eliminación de usuario VPN

Esta operación se ejecuta reutilizando la misma sesión autenticada establecida previamente, evitando duplicación de lógica y garantizando la coherencia del proceso.

Este tipo de integración es sensible a cambios en la interfaz web o en el firmware del dispositivo. No obstante, se trata de una técnica habitualmente empleada cuando no se dispone de APIs públicas de gestión, y permite automatizar sistemas cerrados de forma controlada y documentada.

Integración con la lógica del sistema

En el contexto de la plataforma desarrollada, la gestión del acceso VPN se integraría de forma análoga al resto de servicios:

- La creación de una afiliación con acceso VPN generaría una tarea de provisión.
- Dicha tarea ejecutaría la operación de alta del usuario VPN.
- Al alcanzar la fecha de fin de afiliación, el sistema generaría automáticamente una tarea de revocación.
- Esta tarea eliminaría la cuenta VPN del usuario desde el sistema de gestión correspondiente.

De este modo, el acceso VPN quedaría completamente alineado con el ciclo de vida del usuario dentro del grupo de investigación.

Consideraciones de seguridad

Dado el carácter crítico del acceso VPN, se contemplan las siguientes medidas:

- Uso de contraseñas robustas generadas por el sistema.
- Eliminación inmediata de cuentas al finalizar la afiliación.
- Registro y auditoría de todas las operaciones de alta y baja.
- Posibilidad de ejecutar las operaciones en modo **dry-run** durante pruebas o validación del sistema.

Resumen

Aunque la integración con el servicio VPN no se ha implementado de forma automática en esta fase del proyecto, se ha realizado un **análisis detallado del sistema real**, documentando:

- El funcionamiento actual del servicio.
- El proceso real de alta y baja de usuarios.
- Las posibles estrategias de automatización.
- La integración conceptual con la lógica global de la plataforma.

Este enfoque permite justificar técnicamente la viabilidad de la integración y demuestra un conocimiento profundo de la infraestructura del laboratorio.

4.2.5 Integración con herramientas colaborativas (Microsoft Teams)

Objetivo de la integración

Microsoft Teams es utilizado en el grupo de investigación como herramienta de comunicación y colaboración, organizada en **equipos** asociados a proyectos o líneas de trabajo.

La integración con este servicio tiene como objetivo **gestionar de forma automática la pertenencia de los usuarios a los equipos**, de acuerdo con su afiliación al grupo de investigación.

En este contexto, la plataforma debe permitir:

- Añadir usuarios a equipos de Microsoft Teams al inicio de su afiliación.
- Eliminar usuarios de dichos equipos cuando la afiliación finaliza.
- Mantener la coherencia entre el estado interno del sistema y los permisos efectivos en la herramienta colaborativa.

Tecnología empleada y justificación

Microsoft Teams no ofrece una API específica e independiente para la gestión de usuarios. En su lugar, la administración de equipos y miembros se realiza a través de **Microsoft Graph**, la API unificada de Microsoft para acceder a recursos de Microsoft 365.

Desde el punto de vista técnico, un **equipo de Teams** corresponde internamente a un **Microsoft 365 Group**, por lo que la gestión de miembros se lleva a cabo operando sobre dicho grupo mediante la API de Microsoft Graph.

Este enfoque es el recomendado oficialmente por Microsoft y permite una integración realista y alineada con entornos corporativos.

Autenticación y obtención de credenciales

La autenticación contra Microsoft Graph se realiza mediante un flujo de **Client Credentials**, utilizando una **aplicación registrada en Entra ID (Azure Active Directory)**.

El proceso requiere:

1. Registrar una aplicación en Entra ID.
2. Obtener:
    - tenant_id
    - client_id
    - client_secret
3. Conceder permisos de tipo **Application**, entre ellos:
    - GroupMember.ReadWrite.All
    - Group.ReadWrite.All
4. Otorgar *admin consent* a dichos permisos.

Este modelo permite que la plataforma actúe como un **servicio backend autónomo**, sin necesidad de intervención de usuarios finales.

Las credenciales se gestionan externamente mediante variables de entorno y no se almacenan en el código fuente.

Estas credenciales se obtienen a través del portal de administración de Entra ID, donde un administrador registra una aplicación asociada al sistema, genera un secreto de cliente (*client secret*) y concede explícitamente los permisos necesarios mediante *admin consent*. Los valores obtenidos (tenant_id, client_id y client_secret) no se almacenan en el código fuente, sino que se gestionan como variables de entorno en el servidor donde se despliega la plataforma, reduciendo así el riesgo de exposición de credenciales sensibles.

Alta de usuarios en un equipo de Microsoft Teams

La operación de alta consiste en **añadir un usuario existente** al grupo de Microsoft 365 asociado a un equipo de Teams.

Para ello es necesario:

- El identificador del grupo (groupId).
- El identificador del usuario en Entra ID (userId, Object ID).

La operación se realiza mediante una llamada HTTP a la API de Microsoft Graph:

```
POST /groups/{groupId}/members/$ref
```

A continuación se muestra un fragmento de código **teórico pero realista**, basado en el uso directo de Microsoft Graph, que ilustra cómo se realizaría esta operación desde Java.

El Listado 4.9 muestra cómo la plataforma obtendría un token de acceso mediante el flujo de client credentials y lo utilizaría para añadir un usuario a un equipo de Microsoft Teams, interactuando directamente con la API de Microsoft Graph.

```
import com.azure.identity.ClientSecretCredential;
import com.azure.identity.ClientSecretCredentialBuilder;
import com.azure.core.credential.AccessToken;
import com.azure.core.credential.TokenRequestContext;
import okhttp3.*;

public class TeamsProvisioningConnector {

    private static final String GRAPH_SCOPE = "https://graph.microsoft.com/.default";
    private final OkHttpClient httpClient = new OkHttpClient();

    private final String tenantId;
    private final String clientId;
    private final String clientSecret;

    public TeamsProvisioningConnector(String tenantId, String clientId, String
clientSecret) {
        this.tenantId = tenantId;
        this.clientId = clientId;
        this.clientSecret = clientSecret;
    }

    private String acquireAccessToken() {
        ClientSecretCredential credential = new ClientSecretCredentialBuilder()
                .tenantId(tenantId)
                .clientId(clientId)
                .clientSecret(clientSecret)
                .build();

        TokenRequestContext ctx = new TokenRequestContext().addScopes(GRAPH_SCOPE);
        AccessToken token = credential.getToken(ctx).block();
        return token.getToken();
    }

    public void addUserToTeam(String groupId, String userId) throws Exception {
        String token = acquireAccessToken();

        String url = "https://graph.microsoft.com/v1.0/groups/"
```

```
                    + groupId + "/members/$ref";

        String jsonBody =
                "{ \"@odata.id\":
\"https://graph.microsoft.com/v1.0/directoryObjects/"
                + userId + "\" }";

        Request request = new Request.Builder()
                .url(url)
                .post(RequestBody.create(jsonBody,
                        MediaType.parse("application/json")))
                .addHeader("Authorization", "Bearer " + token)
                .build();

        httpClient.newCall(request).execute();
    }
}
```

Listado 4.9: alta de usuario en Teams mediante Microsoft Graph

Operación inversa: eliminación de usuarios de un equipo

Tal y como exige el diseño del sistema, la integración define explícitamente la **operación opuesta** a la asignación de un usuario a un equipo.

La eliminación de un usuario se realiza mediante la operación:

```
DELETE /groups/{groupId}/members/{userId}/$ref
```

De forma análoga a la operación de alta, la revocación del acceso a un equipo de Microsoft Teams se realiza eliminando la referencia del usuario dentro del grupo asociado.

```
public void removeUserFromTeam(String groupId, String userId) throws Exception {
    String token = acquireAccessToken();

    String url = "https://graph.microsoft.com/v1.0/groups/"
               + groupId + "/members/" + userId + "/$ref";

    Request request = new Request.Builder()
            .url(url)
            .delete()
            .addHeader("Authorization", "Bearer " + token)
            .build();

    httpClient.newCall(request).execute();
}
```

Listado 4.10: eliminación de usuario de Teams

Esta operación garantiza que el usuario pierde inmediatamente el acceso al equipo y a los recursos asociados.

Integración con la lógica del sistema

La gestión de accesos a Microsoft Teams se integra en la plataforma siguiendo el mismo patrón que el resto de servicios:

- Al iniciar una afiliación, se genera una tarea de provisión que añade al usuario a los equipos correspondientes.
- Al finalizar la afiliación, se genera automáticamente una tarea de revocación que elimina al usuario de dichos equipos.
- Las operaciones pueden ejecutarse de forma manual desde la interfaz de administración.

Este enfoque asegura la coherencia entre el estado interno del sistema y los accesos efectivos en la herramienta colaborativa.

Gestión de errores y consideraciones de seguridad

Durante la interacción con Microsoft Graph pueden producirse errores como:
- Identificadores incorrectos de usuario o grupo.
- Permisos insuficientes de la aplicación registrada.
- Limitaciones de cuota o latencia en la API.

Todos los errores se registran y asocian a la tarea de provisión correspondiente, permitiendo auditoría y diagnóstico.

Desde el punto de vista de seguridad:

- Se utilizan cuentas técnicas y permisos mínimos.
- Las credenciales se gestionan externamente.
- Las operaciones pueden ejecutarse en modo **dry-run** durante pruebas.

Resumen

Aunque la integración con Microsoft Teams no se ha desplegado de forma efectiva en el entorno real, el análisis realizado y el código presentado demuestran la **viabilidad técnica completa** de la integración, alineada con los mecanismos oficiales proporcionados por Microsoft y con el diseño global de la plataforma.

4.2.6 Notificaciones y avisos automáticos

Finalmente, el sistema incorpora el diseño de un mecanismo de notificaciones automáticas orientado a mejorar la gestión del ciclo de vida de las afiliaciones. En particular, se contempla el envío de avisos por correo electrónico asociados a la proximidad del fin de una afiliación.

Este mecanismo se basa en la ejecución periódica de tareas programadas que consultan las fechas de las afiliaciones registradas y generan las notificaciones correspondientes cuando se cumplen las condiciones definidas. Este diseño refuerza el carácter preventivo del sistema y contribuye a evitar accesos prolongados más allá del periodo previsto.

4.3 Gestión del ciclo de vida de usuarios en el backend

El backend de la plataforma actúa como el componente central encargado de coordinar la gestión del ciclo de vida de los usuarios y sus accesos a los distintos servicios del grupo de investigación.

A diferencia de una aplicación puramente CRUD, el backend no se limita a almacenar información, sino que implementa la lógica necesaria para **activar, mantener y revocar accesos** en función del estado de las afiliaciones definidas en el sistema.

Modelo de control basado en afiliaciones

El elemento fundamental que gobierna el comportamiento del backend es el concepto de **afiliación**, asociado a cada usuario.

Cada afiliación define:

- un tipo de pertenencia
- una fecha de inicio
- una fecha de fin
- un conjunto de servicios asociados

Estas fechas determinan de forma automática cuándo deben ejecutarse las operaciones de provisión (alta) o revocación (baja) sobre los distintos servicios integrados.

Orquestación de altas y bajas

El backend implementa un mecanismo de orquestación mediante el cual las acciones sobre los servicios no se ejecutan directamente desde la interfaz de usuario, sino que se transforman en **tareas internas de provisión**.

De este modo:

- Cuando una afiliación entra en vigor, el backend genera tareas de alta para los servicios asociados.
- Cuando una afiliación caduca, el backend genera automáticamente tareas de revocación que eliminan o desactivan los accesos correspondientes.

Este enfoque permite desacoplar la lógica de negocio de la ejecución concreta sobre cada servicio y garantiza un comportamiento consistente incluso en escenarios con errores o retrasos en la ejecución.

Gestión de expiraciones y borrado automático de usuarios

La caducidad de una afiliación constituye uno de los escenarios más críticos del sistema. Para gestionarlo, el backend incorpora procesos programados que revisan periódicamente las fechas de fin de afiliación.

Al detectarse una afiliación expirada:

- se bloquean o eliminan los accesos del usuario en los servicios correspondientes,
- se actualiza el estado interno del sistema,
- y se registra la operación para fines de auditoría.

Este proceso asegura que los usuarios no mantengan accesos más allá del periodo autorizado, incluso en ausencia de intervención manual por parte de un administrador.

Trazabilidad y control

Todas las operaciones ejecutadas por el backend quedan registradas, permitiendo conocer en todo momento qué acciones se han realizado, sobre qué usuario y con qué resultado.

Este registro facilita tanto la detección de errores como la justificación de las decisiones tomadas por el sistema en función del estado de las afiliaciones.

4.4 Diseño del modelo de datos

El modelo de datos de la plataforma se ha diseñado con el objetivo de almacenar de forma estructurada la información necesaria para la gestión de usuarios, afiliaciones, accesos a servicios y tareas asociadas.

El diseño del modelo refleja los requisitos funcionales identificados en el capítulo anterior y permite mantener la coherencia entre la información gestionada desde la interfaz y las operaciones internas del sistema. Para su implementación se ha utilizado una base de datos relacional, lo que facilita la integridad referencial y la trazabilidad de las operaciones realizadas.

4.4.1 Entidades principales del sistema

El modelo de datos se articula en torno a un conjunto de entidades principales que representan los conceptos fundamentales del dominio del problema. Las entidades más relevantes son:

- **Usuario**, que representa a las personas gestionadas por la plataforma.
- **Afiliación**, que define la pertenencia temporal de un usuario al grupo de investigación.
- **Servicio**, que representa los servicios gestionados por el sistema.
- **Asignación de servicio**, que relaciona usuarios y servicios, indicando el estado del acceso.
- **Documento**, que permite asociar documentación relevante a un usuario.
- **Tarea**, que representa las operaciones registradas por el sistema relacionadas con la provisión de servicios.

4.4.2 Diagrama del modelo de datos

La Figura 4.5 muestra el diagrama del modelo de datos de la plataforma, en el que se representan las entidades principales del sistema, sus atributos y las relaciones existentes entre ellas.

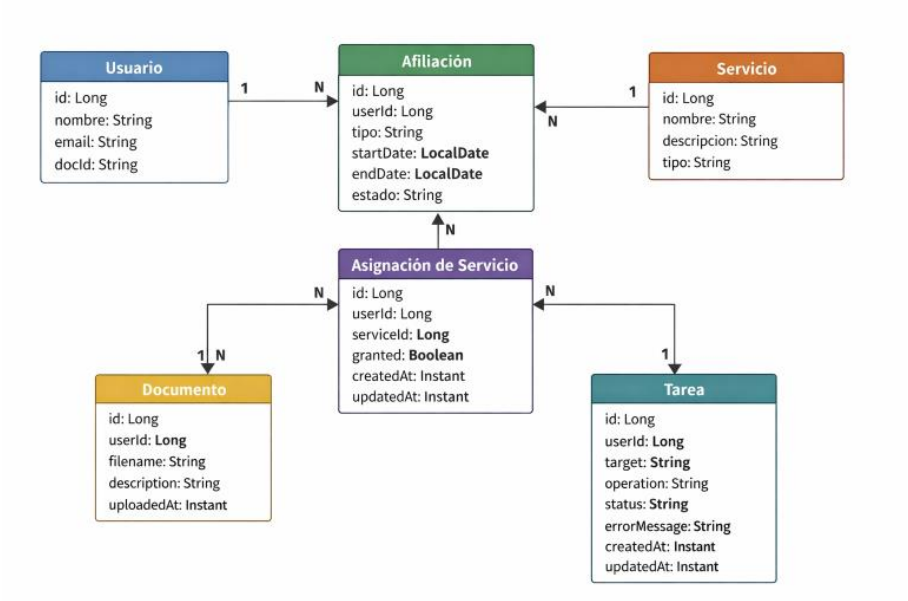

Figura 4.5: Modelo de datos de plataforma de gestión

4.4.3 Descripción de entidades y atributos

Entidad: Usuario

En la Tabla 4.1 se muestra la estructura de la entidad Usuario, incluyendo sus atributos principales y tipos de datos.

Tabla 4.1: Estructura de la entidad Usuario

Atributo	Tipo de dato	Descripción
id	Long	Identificador único del usuario
nombre	String	Nombre completo del usuario
email	String	Dirección de correo electrónico
docId	String	Identificador documental del usuario

Entidad: Afiliación

En la Tabla 4.2 se muestra la estructura de la entidad Afiliación, incluyendo sus atributos principales y tipos de datos.

Tabla 4.2: Estructura de la entidad Afiliación

Atributo	Tipo de dato	Descripción
id	Long	Identificador único de la afiliación
userId	Long	Referencia al usuario
tipo	String	Tipo de afiliación
startDate	LocalDate	Fecha de inicio
endDate	LocalDate	Fecha de fin
estado	String	Estado de la afiliación

Entidad: Servicio

En la Tabla 4.3 se muestra la estructura de la entidad Servicio, incluyendo sus atributos principales y tipos de datos.

Tabla 4.3: Estructura de la entidad Servicio

Atributo	Tipo de dato	Descripción
id	Long	Identificador del servicio
nombre	String	Nombre del servicio
descripcion	String	Descripción funcional
tipo	String	Tipo de servicio

Entidad: Asignación de servicio

En la Tabla 4.4 se muestra la estructura de la entidad Asignación de servicio, incluyendo sus atributos principales y tipos de datos.

Tabla 4.4:Estructura de la entidad Asignación de Servicio

Atributo	Tipo de dato	Descripción
id	Long	Identificador de la asignación
userId	Long	Usuario asociado
serviceId	Long	Servicio asociado
granted	Boolean	Estado del acceso
createdAt	Instant	Fecha de creación
updatedAt	Instant	Última modificación

Entidad: Documento

En la Tabla 4.5 se muestra la estructura de la entidad Documento, incluyendo sus atributos principales y tipos de datos.

Tabla 4.5: Estructura de la entidad Documento

Atributo	Tipo de dato	Descripción
id	Long	Identificador del documento
userId	Long	Usuario asociado
filename	String	Nombre del archivo
description	String	Texto identificativo
uploadedAt	Instant	Fecha de subida

Entidad: Tarea

En la Tabla 4.6 se muestra la estructura de la entidad Tarea de Provisión, incluyendo sus atributos principales y tipos de datos.

Tabla 4.6: Estructura de la entidad Tarea de Provisión

Atributo	Tipo de dato	Descripción

id	Long	Identificador de la tarea
userId	Long	Usuario afectado
target	String	Servicio o recurso
operation	String	Tipo de operación
status	String	Estado de la tarea
errorMessage	String	Mensaje de error
createdAt	Instant	Fecha de creación
updatedAt	Instant	Última actualización

4.4.4 Justificación del modelo

El modelo de datos diseñado permite representar de forma clara las relaciones entre usuarios, afiliaciones y servicios, manteniendo la trazabilidad de las operaciones realizadas por el sistema. La separación entre entidades facilita la extensibilidad del modelo y permite incorporar nuevos servicios o funcionalidades sin necesidad de realizar cambios estructurales significativos.

Este diseño proporciona una base sólida para la implementación del sistema y se alinea con la arquitectura definida en el apartado anterior.

4.5 Diseño de la interfaz de usuario

El diseño de la interfaz de usuario se abordó **como una fase previa al desarrollo**, con el objetivo de definir la estructura general de la aplicación, las vistas necesarias y los flujos de navegación principales, antes de implementar la interfaz final.

Este diseño previo permitió validar la coherencia de la interfaz con los requisitos funcionales y sirvió como base directa para la implementación posterior descrita en el capítulo de desarrollo.

4.5.1 Prototipo de interfaz (boceto previo)

Antes de iniciar el desarrollo de la interfaz web, se realizó un **prototipo preliminar de la interfaz de usuario**, elaborado de forma manual mediante bocetos en papel.

Este prototipo tiene como objetivo definir la estructura general de la aplicación, las vistas principales y los flujos de navegación entre ellas, de manera independiente a cualquier tecnología concreta de implementación.

En el boceto se identifican explícitamente las vistas de gestión de usuarios, detalle de usuario, afiliaciones y asignación de servicios, así como las acciones principales disponibles en cada una de ellas (alta, edición, eliminación y consulta de información).

Este diseño previo (lógicamente muy básico y conceptual) permitió validar la organización funcional de la interfaz y sirvió como base directa para la implementación posterior descrita en el capítulo de desarrollo.

En la Figura 4.5 se muestra el boceto original empleado durante la fase de diseño, donde se reflejan las vistas y flujos principales del sistema.

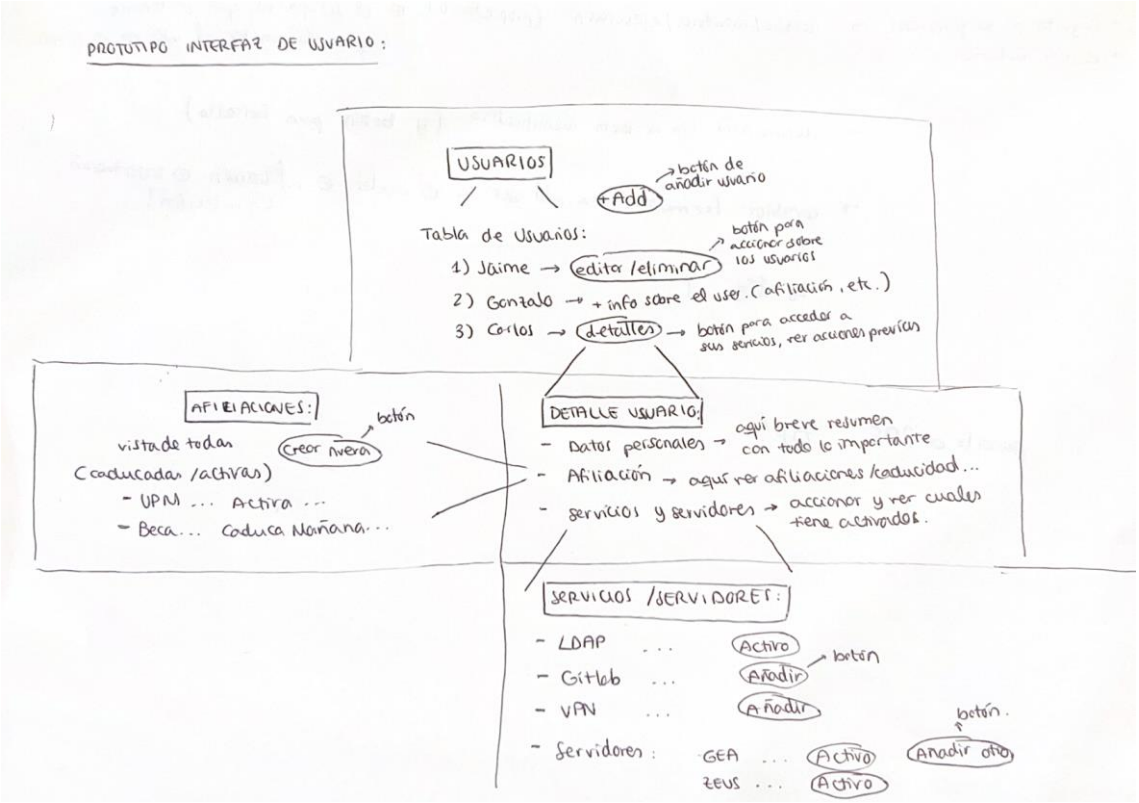

Figura 4.6: Boceto preliminar de la interfaz de usuario

4.5.2 Diseño conceptual de las vistas

A partir del análisis de requisitos, se identificaron las principales vistas que debía ofrecer la plataforma:

- **Vista de listado de usuarios**, desde la que se puede consultar el conjunto de usuarios registrados y acceder al detalle de cada uno de ellos.
- **Vista de detalle de usuario**, que centraliza la información básica del usuario y permite gestionar afiliaciones, accesos a servicios y documentación asociada.
- **Formularios de creación y edición**, utilizados para el alta de usuarios y la gestión de afiliaciones.

Este conjunto reducido de vistas permite concentrar la mayoría de las operaciones habituales en un número limitado de pantallas, evitando una navegación compleja y mejorando la eficiencia en el uso del sistema.

4.5.3 Criterios de diseño de la interfaz

Durante el diseño de la interfaz se han tenido en cuenta una serie de criterios orientados a mejorar la experiencia de uso y reducir errores en la gestión administrativa:

- **Consistencia visual**: se ha mantenido una estructura homogénea entre las distintas vistas, utilizando componentes reutilizables y una disposición similar de los elementos.

- **Claridad de la información**: la información relevante se presenta de forma estructurada mediante tablas y secciones claramente diferenciadas.

- **Acciones explícitas**: las operaciones de concesión o revocación de accesos se representan mediante botones claramente identificables, mostrando en todo momento el estado actual del servicio.

- **Reducción de contexto**: se ha priorizado que las operaciones más frecuentes puedan realizarse desde la vista de detalle de usuario, evitando cambios constantes de pantalla.

4.5.4 Relación entre diseño y resultado final

El diseño prototipo definido en esta fase ha servido como guía para la implementación de la interfaz final de la plataforma. Aunque durante el desarrollo se realizaron ajustes y mejoras en función de las necesidades detectadas, la estructura general de vistas y flujos de navegación se mantuvo conforme al diseño inicial.

En el capítulo dedicado al desarrollo de la interfaz se muestran las vistas finales implementadas, que reflejan las decisiones de diseño adoptadas en esta fase y su adaptación a un entorno web funcional.

5 Desarrollo

5.1 Estructura del proyecto y organización del código

El desarrollo de la plataforma se ha llevado a cabo siguiendo una estructura modular y organizada, alineada con el patrón arquitectónico Modelo–Vista–Controlador (MVC). Esta organización permite separar claramente las responsabilidades del sistema, facilitando su comprensión, mantenimiento y posible evolución futura.

El proyecto se ha estructurado en distintos paquetes, cada uno de ellos con una responsabilidad bien definida dentro del sistema. Esta separación permite aislar la lógica de negocio, el acceso a datos y la gestión de la interfaz de usuario, evitando dependencias innecesarias entre componentes.

El paquete **web** contiene los controladores responsables de gestionar las peticiones HTTP provenientes de la interfaz web. Estos controladores se encargan de preparar los datos necesarios para su renderizado mediante las vistas y de coordinar las acciones solicitadas por el usuario.

El paquete **service** agrupa la lógica de negocio de la aplicación. En él se implementan los servicios encargados de gestionar usuarios, afiliaciones, accesos a servicios y la creación de tareas internas relacionadas con la provisión de servicios. Esta capa actúa como intermediaria entre los controladores y el acceso a datos.

El paquete **repo** contiene los repositorios de acceso a datos, implementados mediante Spring Data JPA. Estos repositorios permiten realizar operaciones de consulta y persistencia sobre la base de datos sin acoplar el resto del sistema a los detalles de almacenamiento.

El paquete **domain** incluye las entidades del modelo de datos, que representan los conceptos principales del sistema, como usuarios, afiliaciones, servicios, documentos y tareas. Estas entidades están mapeadas a la base de datos relacional utilizada por la aplicación.

El paquete **jobs** agrupa los componentes encargados de ejecutar tareas programadas, como la comprobación de la caducidad de afiliaciones o el procesamiento de operaciones pendientes. Estos elementos permiten automatizar procesos internos del sistema de forma controlada.

Finalmente, se incluyen paquetes auxiliares destinados a funcionalidades específicas, como el almacenamiento de documentos o la integración simulada con servicios externos.

Esta organización del proyecto refleja las decisiones de diseño adoptadas en capítulos anteriores y permite mantener una separación clara entre las distintas responsabilidades del sistema. Gracias a esta estructura, el desarrollo del backend y de la interfaz se ha podido realizar de forma independiente, facilitando la incorporación progresiva de nuevas funcionalidades.

5.2 Implementación del backend

El backend de la plataforma constituye el núcleo funcional del sistema y es el encargado de implementar la lógica necesaria para la gestión de usuarios, afiliaciones y accesos a servicios. Su desarrollo se ha realizado utilizando el framework Spring Boot, apoyándose en Spring MVC para la gestión de peticiones web y en Spring Data JPA para el acceso a la base de datos.

La implementación del backend sigue la estructura definida en el capítulo de diseño, separando claramente la capa de controladores, la lógica de negocio y el acceso a datos. Esta separación permite mantener un código modular y facilita la comprensión del funcionamiento interno del sistema.

5.2.1 Controladores

Los controladores del sistema se encargan de gestionar las peticiones HTTP generadas desde la interfaz de usuario y coordinar la ejecución de las operaciones correspondientes. Estos componentes actúan como punto de entrada al backend y delegan la lógica de negocio en los servicios internos.

Entre los controladores implementados destacan aquellos encargados de la gestión de usuarios y de la visualización de su información detallada, ya que concentran gran parte de la funcionalidad principal de la plataforma.

El Listado 5.1 muestra un fragmento representativo del controlador encargado de gestionar la vista de detalle de usuario. En él se observa cómo se recupera la información del usuario, sus afiliaciones, accesos a servicios y tareas asociadas, y cómo se prepara dicha información para su renderizado en la interfaz web.

```
// Controlador MVC para listado y detalle de usuarios (UI)

@Controller
@RequestMapping("/ui/users")
public class UserUiController {

    private final UserRepo userRepo;
    private final ServiceAssignmentRepo assignmentRepo;
    private final AffiliationRepo affiliationRepo;
    private final ProvisioningTaskRepo taskRepo;
    private final UserDocumentRepo documentRepo;

    public UserUiController(UserRepo userRepo,
                            ServiceAssignmentRepo assignmentRepo,
                            AffiliationRepo affiliationRepo,
                            ProvisioningTaskRepo taskRepo,
                            UserDocumentRepo documentRepo) {
        this.userRepo = userRepo;
        this.assignmentRepo = assignmentRepo;
        this.affiliationRepo = affiliationRepo;
        this.taskRepo = taskRepo;
        this.documentRepo = documentRepo;
    }

    @GetMapping
    public String listUsers(Model model) {
        List<UserSummaryDto> users = userRepo.findAll().stream().map(u -> {
            int activos =
assignmentRepo.findByUserIdAndGrantedTrue(u.getId()).size();
```

```
            return new UserSummaryDto(u.getId(), u.getNombre(), u.getEmail(), null,
null, activos);
        }).toList();

        model.addAttribute("users", users);
        return "users";
    }

    @GetMapping("/{id}")
    public String userDetail(@PathVariable Long id, Model model) {
        var user = userRepo.findById(id).orElse(null);
        if (user == null) return "user-not-found";

        model.addAttribute("user", user);

        model.addAttribute("affiliations",
                affiliationRepo.findByUserId(id).stream()
                        .map(a -> new AffiliationDto(
                                a.getId(), a.getUser().getId(), a.getTipo(),
a.getInstitucion(),
                                a.getStartDate(), a.getEndDate(), a.getEstado()
                        ))
                        .toList()
        );

        model.addAttribute("tasks",
                taskRepo.findTop20ByUserIdOrderByCreatedAtDesc(id).stream()
                        .map(t -> new TaskDto(
                                t.getId(), t.getTarget(), t.getOperation(),
                                t.getStatus(), t.getErrorMessage(),
                                t.getCreatedAt(), t.getUpdatedAt()
                        ))
                        .toList()
        );

        model.addAttribute("documents",
documentRepo.findByUserIdOrderByUploadedAtDesc(id));
        return "user-detail";
    }
}
```

Listado 5.1: Controlador MVC para la vista de listado y detalle de usuarios.

5.2.2 Servicios y lógica de negocio

La lógica principal del sistema se concentra en la capa de servicios, donde se implementan las reglas de negocio asociadas a la gestión de usuarios, afiliaciones y accesos a servicios. Esta capa se encarga de validar las operaciones solicitadas, calcular el estado de las afiliaciones en función de las fechas establecidas y coordinar la creación de tareas internas relacionadas con la provisión de servicios.

Un aspecto clave del backend es que las operaciones relacionadas con servicios externos no se ejecutan directamente, sino que se registran como tareas internas. Este enfoque permite simular la integración con servicios reales y aporta trazabilidad sobre las acciones solicitadas desde la plataforma.

El Listado 5.2 muestra un fragmento del servicio encargado de gestionar las operaciones de provisión de servicios. En este fragmento se observa cómo una

solicitud de concesión o revocación de acceso se transforma en una tarea interna registrada en el sistema.

```
// Encolado y procesamiento de tareas de provisión

@Service
public class ProvisioningService {

    private final ProvisioningTaskRepo taskRepo;
    private final UserRepo userRepo;
    private final ServiceAssignmentRepo assignmentRepo;

    public ProvisioningService(ProvisioningTaskRepo taskRepo, UserRepo userRepo,
ServiceAssignmentRepo assignmentRepo) {
        this.taskRepo = taskRepo;
        this.userRepo = userRepo;
        this.assignmentRepo = assignmentRepo;
    }

    public ProvisioningTask enqueueGrant(Long userId, ServiceTarget target, String
payload) {
        return enqueue(userId, target, TaskType.GRANT, payload);
    }

    public ProvisioningTask enqueueRevoke(Long userId, ServiceTarget target, String
payload) {
        return enqueue(userId, target, TaskType.REVOKE, payload);
    }

    private ProvisioningTask enqueue(Long userId, ServiceTarget target, TaskType op,
String payload) {
        ProvisioningTask t = new ProvisioningTask();
        t.setUserId(userId);
        t.setTarget(target);
        t.setOperation(op);
        t.setPayload(payload);
        t.setStatus(TaskStatus.PENDING);
        return taskRepo.save(t);
    }

    /** Procesa hasta 10 tareas PENDING (MVP). */
    @Transactional
    public int processPendingBatch() {
        List<ProvisioningTask> pending =
                taskRepo.findTop10ByStatusOrderByCreatedAtAsc(TaskStatus.PENDING);

        int done = 0;

        for (ProvisioningTask t : pending) {
            try {
                t.setStatus(TaskStatus.RUNNING);
                taskRepo.save(t);

                var user = userRepo.findById(t.getUserId())
                        .orElseThrow(() -> new IllegalStateException("Usuario no
existe: " + t.getUserId()));

                // En el MVP, la integración con servicios externos se simula.
                // El resultado se refleja actualizando el estado local
(ServiceAssignment).

                ServiceAssignment ass = assignmentRepo
                        .findByUserIdAndTarget(t.getUserId(), t.getTarget())
                        .orElse(null);
```

```
                switch (t.getOperation()) {
                    case GRANT -> {
                        if (ass == null) {
                            ass = new ServiceAssignment();
                            ass.setUser(user);
                            ass.setTarget(t.getTarget());
                        }
                        ass.setGranted(true);
                        assignmentRepo.save(ass);
                    }
                    case REVOKE -> {
                        if (ass != null) {
                            ass.setGranted(false);
                            assignmentRepo.save(ass);
                        }
                    }
                    case CHECK -> { /* no-op en MVP */ }
                }

                t.setStatus(TaskStatus.DONE);
                t.setErrorMessage(null);
                taskRepo.save(t);
                done++;

            } catch (Exception ex) {
                t.setAttempts(t.getAttempts() + 1);
                t.setErrorMessage(ex.getMessage());
                t.setStatus(t.getAttempts() >= 3 ? TaskStatus.FAILED :
TaskStatus.PENDING);
                taskRepo.save(t);
            }
        }
        return done;
    }
}
```

Listado 5.2: Encolado y procesamiento por lotes de tareas de provisión (MVP).

5.2.3 Persistencia y acceso a datos

El acceso a los datos se realiza mediante repositorios implementados con Spring Data JPA, que permiten abstraer la interacción con la base de datos relacional utilizada por el sistema. Para el almacenamiento de la información se ha empleado SQLite, lo que facilita el despliegue y la portabilidad de la aplicación durante la fase de desarrollo.

Las entidades del sistema están mapeadas mediante anotaciones JPA, permitiendo definir de forma explícita las relaciones entre usuarios, afiliaciones, servicios y tareas. Este enfoque garantiza la integridad de los datos y facilita la implementación de consultas complejas de forma estructurada.

El Listado 5.3 muestra un fragmento de una entidad JPA utilizada en el sistema, en la que se definen los atributos principales y su correspondencia con la base de datos relacional.

```
// Entidad JPA para tareas de provisión y trazabilidad

@Entity
@Table(name = "provisioning_tasks")
```

```
public class ProvisioningTask {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "user_id", nullable = false)
    private Long userId;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private ServiceTarget target;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private TaskType operation;

    @Lob
    private String payload;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private TaskStatus status = TaskStatus.PENDING;

    @Column(nullable = false)
    private int attempts = 0;

    @Column(name = "error_message")
    private String errorMessage;

    @Column(name = "created_at")
    private Long createdAt;

    @Column(name = "updated_at")
    private Long updatedAt;

    @PrePersist
    void onCreate() {
        long now = System.currentTimeMillis();
        this.createdAt = now;
        this.updatedAt = now;
    }

    @PreUpdate
    void onUpdate() {
        this.updatedAt = System.currentTimeMillis();
    }

    // getters/setters omitidos por brevedad
}
```

Listado 5.3: Entidad JPA (ProvisioningTask) para el registro y trazabilidad de operaciones.

5.3 Implementación de la interfaz de usuario

La interfaz de usuario de la plataforma se ha implementado como una aplicación web renderizada en el servidor, utilizando plantillas HTML y el motor de plantillas Thymeleaf. Esta aproximación permite integrar de forma directa la lógica del backend con la generación de las vistas, manteniendo una arquitectura coherente con el patrón MVC definido previamente.

La implementación de la interfaz sigue el diseño conceptual descrito en el capítulo anterior y permite al administrador gestionar usuarios, afiliaciones, accesos a servicios y documentación desde un entorno unificado.

5.3.1 Vista de listado de usuarios

La vista principal de la plataforma muestra el listado de usuarios registrados en el sistema. Desde esta vista es posible consultar la información básica de cada usuario y acceder de forma directa a su vista de detalle.

El listado se presenta mediante una tabla que resume la información más relevante, incluyendo el nombre, el correo electrónico y el número de servicios activos asociados a cada usuario. Este enfoque permite obtener una visión rápida del estado general del sistema.

La Figura 5.1 muestra la vista de listado de usuarios implementada en la plataforma.

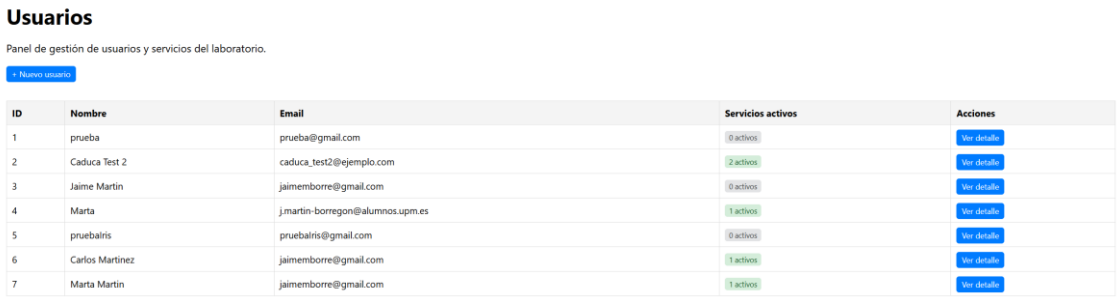

Figura 5.1: Vista principal con listado de usuarios

5.3.2 Vista de detalle de usuario

La vista de detalle de usuario constituye uno de los elementos centrales de la interfaz de la plataforma. Desde esta vista es posible consultar la información básica del usuario, así como gestionar los distintos aspectos relacionados con su pertenencia al grupo de investigación y sus accesos a los servicios disponibles.

En la parte superior de la vista se muestran los datos identificativos del usuario, incluyendo su nombre, correo electrónico e identificador documental. Asimismo, se presenta la información asociada a la cuenta Linux del usuario, como el nombre de usuario, el grupo al que pertenece y los permisos configurados. Esta información permite al administrador disponer de una visión inmediata del estado general del usuario dentro del sistema.

La Figura 5.2 muestra la vista de detalle de usuario con la información general y la afiliación activa.

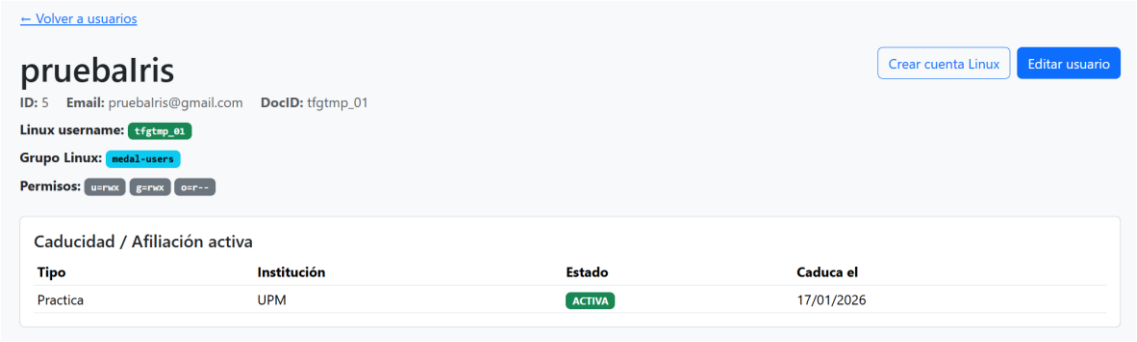

Figura 5.2: Vista de detalle de usuario

5.3.3 Gestión de afiliaciones

Para la gestión de las afiliaciones, la interfaz ofrece un bloque específico desde el que es posible consultar la afiliación activa del usuario al grupo de investigación. En esta sección se muestran de forma estructurada el tipo de afiliación, la institución asociada, el estado actual y la fecha de caducidad.

El uso explícito de fechas de inicio y fin permite controlar de forma precisa la pertenencia temporal del usuario, facilitando la detección de afiliaciones próximas a su vencimiento y contribuyendo a una gestión más ordenada del ciclo de vida de los usuarios dentro del sistema.

La Figura 5.3 ilustra la gestión de afiliaciones de un usuario, incluyendo la visualización de su estado y la fecha de caducidad asociada.

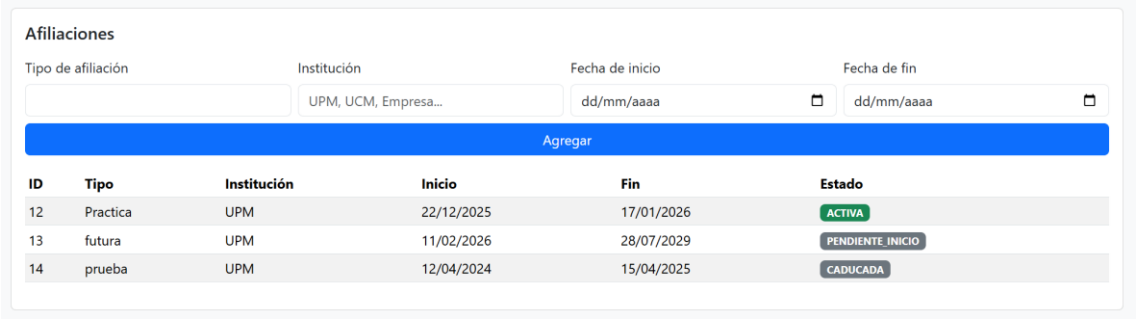

Figura 5.3: Gestión de afiliaciones de un usuario

5.3.4 Gestión de accesos a servicios

La plataforma incluye una sección específica destinada a la gestión de accesos a los distintos servicios disponibles. Desde esta sección, el administrador puede conceder o revocar accesos de forma individualizada para cada usuario, visualizando en todo momento el estado actual de cada servicio.

El diseño de esta vista permite identificar de forma clara qué servicios se encuentran activos para un usuario concreto, facilitando la supervisión y reduciendo el riesgo de errores en la gestión de permisos.

La Figura 5.4 muestra la sección de gestión de accesos a servicios de un usuario.

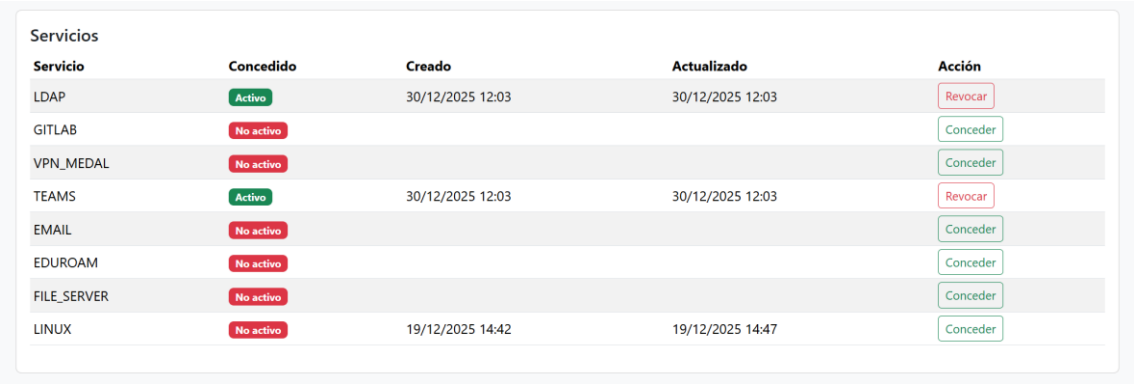

Figura 5.4: Gestión de accesos a servicios de la plataforma

5.3.5 Gestión de accesos a servidores Linux

La administración de accesos a servidores Linux se realiza mediante una sección dedicada dentro de la vista de detalle de usuario. En ella se muestra el conjunto de servidores gestionados por la plataforma y el estado de la cuenta del usuario en cada uno de ellos.

En el contexto del proyecto, la operativa real ha sido validada sobre el servidor CVAT, que se utiliza como entorno de demostración. El resto de servidores se incluyen en la interfaz con el objetivo de evidenciar la extensibilidad del sistema y su capacidad para gestionar múltiples destinos de forma centralizada, aunque sin ejecutar acciones reales sobre ellos.

Desde esta sección es posible definir el grupo Linux al que pertenece el usuario, así como los permisos básicos asociados. El diseño de la interfaz hace explícito el estado de cada acceso, permitiendo conceder o revocar permisos de forma individualizada y reduciendo el riesgo de errores en la gestión administrativa.

La Figura 5.5 muestra la gestión de accesos a servidores Linux, incluyendo la asignación de grupo y permisos, así como el estado de las cuentas en cada servidor.

Figura 5.5: Gestión de accesos a servidores Linux

5.3.6 Gestión de documentación asociada

La plataforma incluye una sección específica dedicada a la gestión de documentación asociada a los usuarios. Desde la interfaz es posible subir documentos en formato PDF, asignarles una descripción identificativa y gestionar su eliminación cuando dejan de ser necesarios.

La información de cada documento se presenta de manera estructurada, incluyendo su nombre, tipo, tamaño y fecha de subida, lo que facilita su consulta y contribuye a centralizar información relevante dentro del propio sistema. Esta funcionalidad permite al administrador disponer de documentación adicional relacionada con el usuario sin necesidad de recurrir a herramientas externas.

La Figura 5.6 muestra la sección de gestión de documentación asociada a un usuario.

Figura 5.6: Gestión de documentos asociados

5.3.7 Registro de tareas y trazabilidad

Con el objetivo de aportar trazabilidad sobre las operaciones realizadas o programadas, la interfaz de usuario incluye un bloque dedicado al registro de tareas recientes del sistema. En esta sección se visualizan las tareas asociadas al usuario, indicando el tipo de operación, el servicio afectado y el estado de ejecución.

Este registro permite supervisar de forma sencilla las acciones solicitadas desde la plataforma y facilita la identificación de posibles errores o incidencias durante el proceso de provisión de servicios. La visualización de estas tareas refuerza el carácter controlado y auditable del sistema.

La Figura 5.7 muestra el registro de tareas recientes del sistema asociado a un usuario.

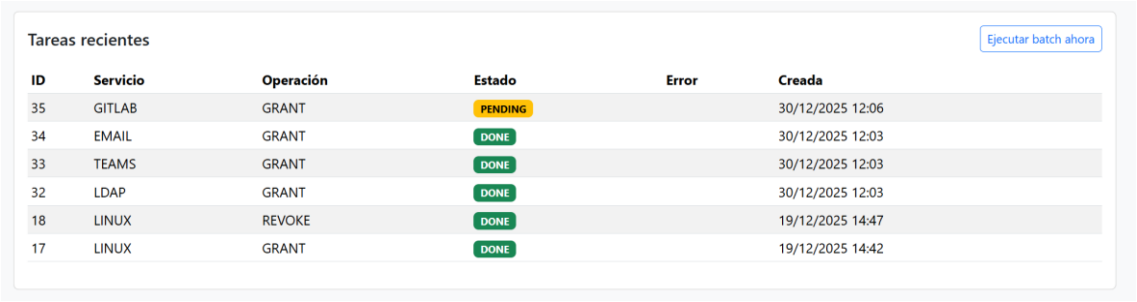

Figura 5.7: Registro de tareas recientes

5.4 Automatización y gestión de tareas

Mientras que en el apartado anterior se ha descrito cómo estas tareas se visualizan desde la interfaz de usuario, en este apartado se detalla el funcionamiento interno del sistema de automatización y gestión de dichas tareas.

La plataforma incorpora mecanismos de automatización orientados a facilitar la gestión de accesos a servicios y a mejorar la trazabilidad de las operaciones realizadas. Dado que muchas de las acciones relacionadas con la provisión de servicios pueden tener impacto sobre infraestructuras reales, el sistema ha sido diseñado para registrar dichas acciones de forma controlada y procesarlas de manera diferida.

Las operaciones de concesión y revocación de accesos no se ejecutan directamente sobre los servicios externos. En su lugar, el sistema registra cada solicitud como una tarea interna, que encapsula la información necesaria para llevar a cabo la operación solicitada. Estas tareas incluyen datos como el usuario afectado, el servicio implicado, el tipo de operación y el estado de ejecución.

Este enfoque permite separar claramente la solicitud de una acción de su ejecución efectiva, aportando un mayor control sobre el sistema y reduciendo riesgos asociados a la automatización directa sobre servicios productivos. Además, facilita la simulación de integraciones reales en un contexto académico, donde el acceso directo a determinados sistemas no siempre es posible.

Las tareas registradas se procesan de forma periódica mediante componentes internos del sistema, que evalúan su estado y ejecutan la lógica correspondiente. Durante este proceso, se actualiza el estado de cada tarea, permitiendo identificar aquellas que se han completado correctamente y aquellas que han fallado o requieren reintentos. Este mecanismo contribuye a la robustez del sistema y permite gestionar errores de forma controlada.

Un aspecto relevante de este diseño es la trazabilidad de las operaciones realizadas. Al mantener un registro histórico de todas las tareas asociadas a un usuario, la plataforma permite consultar las acciones solicitadas, el momento en que se produjeron y su resultado. Esta información resulta especialmente útil tanto para la supervisión del sistema como para la depuración de posibles incidencias.

Adicionalmente, el sistema contempla la automatización de tareas relacionadas con el ciclo de vida de las afiliaciones. Entre estas tareas se incluye la comprobación periódica de la caducidad de las afiliaciones y la generación de acciones asociadas, como la notificación o la actualización del estado del usuario dentro del sistema. Este comportamiento refuerza el carácter preventivo de la plataforma y contribuye a evitar situaciones en las que los accesos permanezcan activos más allá del periodo previsto.

En conjunto, el mecanismo de automatización y gestión de tareas proporciona una base sólida para la evolución futura del sistema hacia un entorno plenamente operativo. Aunque en el contexto del proyecto muchas integraciones se implementan de forma demostrativa, la arquitectura adoptada permite sustituir fácilmente la lógica simulada por integraciones reales, manteniendo un control exhaustivo y auditable sobre las operaciones realizadas.

5.5 Trazabilidad de requisitos

La siguiente tabla resume la relación entre los requisitos funcionales definidos en el capítulo de análisis y su implementación en el sistema desarrollado, indicando el estado de cada requisito y la sección de la memoria en la que se evidencia su cumplimiento.

Tabla 5.1: Trazabilidad entre requisitos funcionales e implementación

Requisito	Descripción	Estado de implementación	Evidencia en la memoria
RF-01	Alta de usuario en el sistema	Implementado	Secciones 5.2.1 y 5.3.1
RF-02	Gestión de afiliaciones de usuario	Implementado	Sección 5.3.3
RF-03	Gestión de accesos a servicios	Implementado (MVP)	Secciones 5.3.4 y 5.4
RF-04	Gestión de documentación asociada a usuarios	Implementado	Sección 5.3.5
RF-05	Automatización y registro de tareas	Implementado (MVP)	Sección 5.4

6 Pruebas

6.1 Objetivo de las pruebas

El objetivo de este capítulo es describir las pruebas realizadas sobre el sistema desarrollado con el fin de verificar su correcto funcionamiento y validar que los requisitos definidos han sido implementados de forma adecuada.

Dado el carácter académico del proyecto y las limitaciones existentes para la integración real con determinados servicios externos, las pruebas se han centrado principalmente en comprobar el comportamiento funcional del sistema, la coherencia de la interfaz de usuario y la correcta gestión interna de las operaciones y tareas registradas.

6.2 Entorno de pruebas

Las pruebas del sistema se han realizado en un entorno de desarrollo local, utilizando un servidor web embebido proporcionado por Spring Boot y una base de datos SQLite para el almacenamiento de la información.

El acceso a la aplicación se ha realizado a través de un navegador web, permitiendo validar tanto el comportamiento del backend como la correcta renderización de la interfaz de usuario. Este entorno de pruebas reproduce de forma fiel el funcionamiento previsto del sistema en un contexto real, sin necesidad de desplegar la aplicación sobre infraestructura productiva.

6.3 Estrategia de pruebas

La estrategia de pruebas adoptada combina pruebas manuales de tipo funcional y pruebas de verificación del flujo de operaciones internas. Estas pruebas permiten comprobar que las acciones realizadas desde la interfaz de usuario producen los efectos esperados en el sistema y que la información mostrada se mantiene coherente.

No se han implementado pruebas automatizadas debido a la naturaleza del proyecto y al enfoque demostrativo de determinadas integraciones. No obstante, las pruebas manuales realizadas permiten validar de forma suficiente el comportamiento del sistema en relación con los requisitos establecidos.

6.4 Pruebas funcionales

6.4.1 Alta y visualización de usuarios

Se ha verificado el correcto funcionamiento del proceso de alta de usuarios mediante la interfaz de administración. Tras la creación de un nuevo usuario, se comprueba que este aparece en el listado principal y que es posible acceder a su vista de detalle, donde se muestran correctamente sus datos identificativos.

Esta prueba valida el requisito funcional RF-01 y confirma la correcta integración entre la interfaz de usuario, los controladores y la capa de persistencia.

En la Tabla 6.1 se resume la prueba realizada para verificar el alta y visualización de usuarios en el sistema.

Tabla 6.1: Prueba de alta y visualización de usuarios

Campo	**Descripción**
Identificador	PF-01
Funcionalidad	Alta y visualización de usuarios
Descripción	Verificar que un usuario puede darse de alta y visualizarse correctamente en el listado y vista de detalle
Pasos realizados	Alta de un nuevo usuario desde la interfaz y acceso a su vista de detalle
Resultado esperado	El usuario aparece en el listado y sus datos se muestran correctamente
Resultado obtenido	Correcto
Requisito asociado	RF-01

6.4.2 Gestión de afiliaciones

Se han realizado pruebas sobre la gestión de afiliaciones, verificando que es posible consultar la afiliación activa de un usuario y que la información relativa a fechas y estado se presenta de forma coherente en la interfaz.

Asimismo, se ha comprobado que el sistema refleja correctamente la caducidad de una afiliación, validando el comportamiento descrito en el requisito funcional RF-02.

La Tabla 6.2 recoge la prueba realizada sobre la gestión de afiliaciones de usuarios.

Tabla 6.2: Prueba de gestión de afiliaciones

Campo	**Descripción**
Identificador	PF-02
Funcionalidad	Gestión de afiliaciones
Descripción	Comprobar la correcta visualización del estado y fechas de afiliación
Pasos realizados	Consulta de la afiliación activa de un usuario
Resultado esperado	Se muestran correctamente las fechas y el estado de la afiliación
Resultado obtenido	Correcto
Requisito asociado	RF-02

6.4.3 Gestión de accesos a servicios

En relación con la gestión de accesos a servicios, se ha comprobado que el sistema permite conceder y revocar accesos de forma individualizada desde la interfaz de usuario. Cada acción genera el registro correspondiente en el sistema, reflejando el nuevo estado del acceso.

En el caso de los servidores Linux, se ha validado la operativa real sobre el servidor CVAT, mientras que el resto de servicios se gestionan de forma demostrativa, confirmando la extensibilidad del sistema y el correcto funcionamiento del modelo MVP planteado.

En la Tabla 6.3 se muestra la prueba correspondiente a la gestión de accesos a servicios.

Tabla 6.3: Prueba de gestión de accesos a servicios

Campo	**Descripción**
Identificador	PF-03
Funcionalidad	Gestión de accesos a servicios
Descripción	Verificar la concesión y revocación de accesos desde la interfaz
Pasos realizados	Activación y desactivación de accesos a servicios para un usuario
Resultado esperado	El estado del acceso se actualiza correctamente
Resultado obtenido	Correcto
Requisito asociado	RF-03

6.4.4 Gestión de documentación

Se han realizado pruebas sobre la funcionalidad de gestión de documentación asociada a usuarios, comprobando que es posible subir nuevos documentos, visualizarlos en la interfaz y eliminarlos cuando es necesario.

Estas pruebas confirman el cumplimiento del requisito funcional RF-04 y la correcta persistencia de la información asociada a los documentos.

La Tabla 6.4 resume la prueba realizada sobre la gestión de documentación asociada a usuarios.

Tabla 6.4: Prueba de gestión de documentación

Campo	**Descripción**
Identificador	PF-04
Funcionalidad	Gestión de documentación
Descripción	Comprobar la subida, visualización y eliminación de documentos
Pasos realizados	Subida y eliminación de un documento PDF

Resultado esperado	El documento se gestiona correctamente desde la interfaz
Resultado obtenido	Correcto
Requisito asociado	RF-04

6.5 Pruebas de automatización y tareas

Se han realizado pruebas sobre el mecanismo de automatización y gestión de tareas, verificando que las operaciones de concesión y revocación de accesos generan correctamente tareas internas con el estado adecuado.

Asimismo, se ha comprobado que el sistema procesa dichas tareas de forma periódica, actualizando su estado y manteniendo un registro histórico accesible desde la interfaz de usuario. Este comportamiento valida el diseño del sistema orientado a la trazabilidad y al control de las operaciones.

En la Tabla 6.5 se presenta la prueba realizada sobre el mecanismo de automatización y gestión de tareas.

Tabla 6.5: Prueba de automatización y registro de tareas

Campo	**Descripción**
Identificador	PF-05
Funcionalidad	Automatización y gestión de tareas
Descripción	Verificar la creación y procesamiento de tareas internas
Pasos realizados	Generación de tareas de provisión y revisión de su estado
Resultado esperado	Las tareas se registran y actualizan correctamente
Resultado obtenido	Correcto
Requisito asociado	RF-05

6.6 Resultados de las pruebas

Los resultados de las pruebas realizadas indican que el sistema funciona de forma coherente con los requisitos definidos y cumple los objetivos planteados para el proyecto. Las funcionalidades principales se han validado correctamente en el entorno de pruebas, y las integraciones demostrativas se comportan conforme a lo esperado.

Aunque algunas funcionalidades se han implementado en modo MVP, las pruebas realizadas confirman la viabilidad técnica de la solución propuesta y sientan una base sólida para su evolución futura hacia un sistema plenamente operativo.

7 Resultados y conclusiones

El desarrollo de este Trabajo Fin de Grado ha dado lugar a una plataforma web funcional para la gestión centralizada de usuarios y de sus accesos a los servicios de un grupo de investigación. La solución implementada permite registrar usuarios, asociar afiliaciones con información temporal, gestionar accesos a distintos servicios y mantener un historial de las operaciones realizadas, todo ello desde una única interfaz.

Uno de los principales resultados obtenidos es la definición de una arquitectura clara y estructurada que separa de forma adecuada la presentación, la lógica de negocio y el acceso a datos. Esta organización ha permitido desarrollar un sistema comprensible y mantenible, facilitando la incorporación progresiva de nuevas funcionalidades y la adaptación a distintos escenarios de uso. Asimismo, el diseño modular del sistema sienta las bases para una posible evolución futura hacia una integración real con servicios externos.

Desde el punto de vista funcional, la plataforma cumple con los objetivos planteados al inicio del proyecto. La interfaz de usuario permite gestionar de forma intuitiva la información relativa a los usuarios, sus afiliaciones y los servicios disponibles. La inclusión de mecanismos de automatización mediante tareas internas aporta trazabilidad a las operaciones realizadas y permite simular de forma segura la provisión de accesos, evitando modificaciones directas sobre infraestructuras productivas.

Otro resultado relevante es la correcta delimitación del alcance del proyecto. La solución desarrollada demuestra la viabilidad técnica de una herramienta de este tipo en un entorno académico, al tiempo que identifica claramente aquellos aspectos que quedarían fuera de un primer prototipo, como la integración completa en producción con todos los servicios o la implementación de sistemas avanzados de autenticación. Esta delimitación ha permitido centrar el esfuerzo en los elementos clave del sistema y garantizar un desarrollo coherente con el contexto de un Trabajo Fin de Grado.

Como conclusión general, el proyecto ha permitido aplicar de forma práctica los conocimientos adquiridos a lo largo del grado en el ámbito del desarrollo de software, especialmente en lo relativo a diseño de arquitecturas web, modelado de datos y desarrollo de interfaces. El trabajo realizado cumple los objetivos propuestos y constituye una base sólida sobre la que se podría construir una solución plenamente operativa en un entorno real, incorporando integraciones adicionales y mecanismos avanzados de seguridad y automatización.

8 Análisis de Impacto

8.1 Impacto del proceso de desarrollo y toma de decisiones

Durante la realización de este Trabajo Fin de Grado, el proceso de desarrollo ha estado marcado por una sucesión de pruebas, evaluaciones y decisiones técnicas que han condicionado de forma directa el resultado final del proyecto.

Lejos de seguir un camino lineal, el desarrollo ha requerido explorar distintas alternativas, descartar soluciones no viables y redefinir el enfoque inicial en función de los resultados obtenidos y de las recomendaciones del tutor.

Uno de los aspectos más relevantes ha sido **la evaluación de la viabilidad de la integración con servicios reales del laboratorio**, en particular con sistemas de autenticación y directorio. Tras múltiples pruebas y consultas, se constató que la implementación directa de ciertas integraciones, como LDAP, no resultaba viable en el contexto del proyecto debido a limitaciones de acceso, permisos y seguridad. Como resultado de este análisis, y tras discutirlo con el tutor, se tomó la decisión de implementar dichas integraciones en forma de *stub* o simulación, permitiendo validar el diseño y la arquitectura sin comprometer sistemas productivos.

Asimismo, el entorno de desarrollo influyó de manera significativa en el progreso del proyecto. Inicialmente se intentó trabajar con máquinas virtuales en un entorno macOS, lo que derivó en problemas de rendimiento y estabilidad que dificultaban el desarrollo y las pruebas. Tras evaluar esta situación, se decidió modificar el enfoque y utilizar un equipo con sistema Windows como servidor de desarrollo, manteniendo el equipo macOS como cliente. Esta decisión permitió mejorar el rendimiento del entorno y avanzar de forma más eficiente en la implementación.

Otro ejemplo relevante de toma de decisiones se produjo en relación con el **acceso a la red y los servicios mediante VPN**. Durante el desarrollo se intentó reproducir ciertas operaciones que, según la documentación proporcionada, eran posibles en el entorno del tutor, pero que no resultaron viables desde el entorno del autor debido a diferencias en permisos y configuración. Tras analizar estas discrepancias y contrastarlas con el tutor, se concluyó que automatizar dichas operaciones mediante código no era viable ni recomendable en el contexto del TFG, reforzando la decisión de mantener una aproximación simulada y controlada.

En el ámbito de la interfaz de usuario, las primeras versiones desarrolladas permitían validar la funcionalidad básica del sistema, pero no ofrecían una presentación clara ni profesional. A partir de la revisión crítica del estado de la interfaz y de los comentarios recibidos, se decidió realizar un rediseño completo de la UI, priorizando la claridad, la coherencia visual y la usabilidad. Este cambio tuvo un impacto positivo tanto en la calidad del resultado final como en la comprensión del sistema durante las pruebas y demostraciones.

Estas decisiones reflejan un proceso de desarrollo iterativo y consciente, en el que cada elección se ha fundamentado en pruebas reales, limitaciones técnicas

y consideraciones de seguridad. Lejos de constituir deficiencias, los caminos descartados y las soluciones adoptadas ponen de manifiesto la capacidad de análisis y adaptación del proyecto, aspectos esenciales en el desarrollo de sistemas software en entornos reales.

8.2 Impacto personal

La realización de este Trabajo Fin de Grado ha tenido un impacto significativo en el desarrollo personal y académico del autor. El proyecto ha permitido consolidar conocimientos técnicos adquiridos durante el grado, especialmente en áreas como el desarrollo de aplicaciones web, el diseño de arquitecturas software y el modelado de datos.

Además, el proceso de desarrollo ha supuesto un aprendizaje importante en la toma de decisiones técnicas fundamentadas. La necesidad de evaluar alternativas, descartar soluciones no viables y redefinir el enfoque del proyecto en función de las limitaciones encontradas ha contribuido al desarrollo de un criterio más maduro y cercano al ámbito profesional.

El trabajo realizado también ha reforzado habilidades transversales como la planificación, la gestión del tiempo y la comunicación técnica, especialmente a través de las reuniones y consultas con el tutor. En conjunto, este proyecto ha servido como una experiencia formativa integral que va más allá del resultado final, aportando una visión más realista del desarrollo de software en entornos complejos.

8.3 Impacto empresarial

Desde el punto de vista empresarial, la plataforma desarrollada presenta un impacto potencial positivo como herramienta base para la gestión de usuarios en organizaciones con infraestructuras informáticas complejas. La solución propuesta aborda un problema habitual en muchos entornos profesionales, como es la administración de accesos a múltiples servicios de forma centralizada y controlada.

El diseño modular y extensible del sistema facilita su adaptación a distintos contextos organizativos, permitiendo que la plataforma pueda evolucionar hacia una solución plenamente operativa mediante la incorporación de integraciones reales con servicios externos. Asimismo, el enfoque basado en tareas diferidas y trazabilidad de operaciones resulta especialmente relevante en entornos donde la seguridad y la auditoría son aspectos críticos.

Como posible impacto negativo, una implantación directa en un entorno empresarial requeriría un proceso exhaustivo de validación, pruebas y control de permisos, ya que una automatización incorrecta podría derivar en accesos indebidos o interrupciones del servicio. Este aspecto ha sido tenido en cuenta durante el desarrollo, optando por una aproximación conservadora y segura acorde al contexto del proyecto.

8.4 Impacto social

En el ámbito social y académico, la plataforma desarrollada contribuye a mejorar la transparencia y el control en el uso de los recursos tecnológicos de un grupo de investigación. Una gestión más clara y trazable de los accesos reduce el riesgo de usos indebidos y facilita la rendición de cuentas dentro de la organización.

La centralización de la información y la automatización controlada de procesos administrativos pueden contribuir a reducir la carga de trabajo manual de los administradores, permitiéndoles dedicar más tiempo a tareas de mayor valor añadido. Esto repercute positivamente en la eficiencia del grupo y en la calidad de los servicios prestados a sus miembros.

No obstante, una automatización mal diseñada podría generar efectos negativos, como errores en la concesión o revocación de accesos. Por este motivo, el proyecto ha puesto énfasis en la trazabilidad y en la supervisión de las operaciones, minimizando los riesgos asociados a la gestión automática de permisos.

8.5 Impacto económico

El impacto económico de la solución propuesta es, en general, positivo. La plataforma se basa en tecnologías abiertas y ampliamente utilizadas, lo que reduce los costes asociados a licencias de software y facilita su adopción en distintos entornos. Además, la automatización parcial de procesos administrativos puede suponer un ahorro de tiempo y recursos humanos en la gestión de usuarios y servicios.

Desde una perspectiva de costes, la implantación de la plataforma en un entorno real requeriría una inversión inicial en integración, pruebas y mantenimiento. Sin embargo, estos costes podrían verse compensados a medio y largo plazo por la mejora en la eficiencia operativa y la reducción de errores derivados de la gestión manual de accesos.

En el contexto del Trabajo Fin de Grado, el impacto económico es limitado, ya que el desarrollo se ha realizado utilizando recursos disponibles y sin necesidad de infraestructuras adicionales, lo que refuerza la viabilidad del proyecto como prototipo funcional.

8.6 Impacto medioambiental y cultural

El impacto medioambiental del proyecto es reducido, dado que se trata de una solución software que no implica la adquisición de hardware adicional ni un aumento significativo del consumo energético. De manera indirecta, la optimización de procesos administrativos y la reducción de tareas manuales pueden contribuir a un uso más eficiente de los recursos tecnológicos existentes.

Desde el punto de vista cultural y académico, el proyecto promueve buenas prácticas en el desarrollo de software, como la planificación, la trazabilidad de acciones y la consideración de la seguridad desde las fases iniciales del diseño.

Estas prácticas fomentan una cultura de desarrollo responsable y alineada con los estándares profesionales actuales.

Asimismo, el trabajo se alinea con varios **Objetivos de Desarrollo Sostenible (ODS)** de la Agenda 2030, en particular con el **ODS 4 (Educación de calidad)**, al fomentar el aprendizaje práctico y el desarrollo de competencias técnicas, y con el **ODS 9 (Industria, innovación e infraestructura)**, al proponer una solución tecnológica orientada a la mejora y modernización de infraestructuras digitales.

9 Bibliografía

[1] Pivotal Software, “Spring Boot Reference Documentation,” 2024. [Online]. Available: https://spring.io/projects/spring-boot

[2] VMware, “Spring Framework Documentation,” 2024. [Online]. Available: https://docs.spring.io/spring-framework/docs/current/reference/html/

[3] Thymeleaf Project, “Thymeleaf Documentation,” 2024. [Online]. Available: https://www.thymeleaf.org/documentation.html

[4] Bootstrap Team, “Bootstrap Documentation,” 2024. [Online]. Available: https://getbootstrap.com/docs/

[5] SQLite Consortium, “SQLite Documentation,” 2024. [Online]. Available: https://www.sqlite.org/docs.html

[6] Oracle Corporation, “Java Platform, Standard Edition Documentation,” 2024. [Online]. Available: https://docs.oracle.com/en/java/

[7] Apache Software Foundation, “Apache Maven Project,” 2024. [Online]. Available: https://maven.apache.org/

[8] OpenLDAP Project, “OpenLDAP Software Documentation,” 2024. [Online]. Available: https://www.openldap.org/doc/

[9] Git SCM, “Git Documentation,” 2024. [Online]. Available: https://git-scm.com/doc

[10] GitLab Inc., “GitLab Documentation,” 2024. [Online]. Available: https://docs.gitlab.com/

[11] OpenSSH Project, “OpenSSH Manual Pages,” 2024. [Online]. Available: https://www.openssh.com/manual.html

[12] The Linux Foundation, “Linux User Management Documentation,” 2024. [Online]. Available: https://www.kernel.org/doc/

[13] OpenVPN Inc., “OpenVPN Documentation,” 2024. [Online]. Available: https://openvpn.net/community-resources/

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Tue Jan 20 22:53:04 CET 2026
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)