UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo de Backend para una Plataforma Web Destinada a la Centralización y Trazabilidad de la Formación de Bomberos en Chile

Stian John Zamora Marchant

stian.zamora@usm.cl

Carlos Alten L.
Profesor Guía

Hernán Saavedra G.
Profesor Correferente

Resumen: La gestión de la formación académica de los Cuerpos de Bomberos de Chile presenta limitaciones asociadas al uso de procesos manuales y sistemas poco integrados, lo que genera pérdida de trazabilidad, duplicidad de información y dificultades en la administración de usuarios, cursos y evaluaciones, afectando la eficiencia operativa y la capacidad de escalar los procesos de capacitación en un contexto de creciente profesionalización. Frente a este escenario, el presente trabajo propone el diseño e implementación de un sistema backend modular orientado a centralizar la información académica, incorporar mecanismos de autenticación y autorización, y permitir la gestión segura y confiable de usuarios, roles, cursos y evaluaciones. La propuesta es validada mediante pruebas funcionales, de seguridad y de rendimiento, con el objetivo de verificar el cumplimiento de los requisitos planteados. Como resultado, se obtiene un sistema eficiente, escalable y seguro, capaz de optimizar la gestión académica de Bomberos, reducir la carga administrativa, mejorar la confiabilidad de los datos y establecer una base sólida para futuras mejoras e integraciones.

Palabras Clave: Backend, Gestión académica, Trazabilidad, Autenticación, Escalabilidad.

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): **X** Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Desarrollo de Backend para una Plataforma Web Destinada a la Centralización y Trazabilidad de la Formación de Bomberos en Chile
Nombre del candidato(a): Stian John Zamora Marchant
Carrera / Grado: Ingeniería en Informática
Campus: Sede de Viña del Mar
Departamento: ELINF

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Carlos Alten López, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

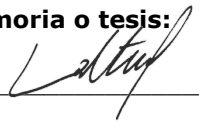
- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

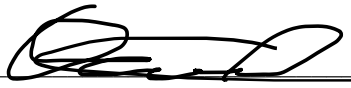
X El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.
☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:
☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 5 de marzo de 2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 05-03-2026 **Firma:** 

1 Introducción

1.1 Contexto y antecedentes.

Los Cuerpos de Bomberos de Chile constituyen una institución fundamental en la protección civil del país, con más de 313 cuerpos activos y aproximadamente 55.000 voluntarios [1]. La magnitud de esta labor exige procesos de capacitación y formación académica permanentes, orientados tanto a la preparación técnica como al cumplimiento de normativas de seguridad.

Sin embargo, gran parte de la gestión académica se sigue realizando mediante procesos manuales y con el uso de sistemas fragmentados, lo que genera duplicidad de información, pérdida de trazabilidad y dificultades para evaluar el progreso de los voluntarios**.** Esta situación limita la eficiencia administrativa y afecta la capacidad de escalar los procesos de capacitación en un contexto donde la profesionalización resulta cada vez más necesaria.

En el ámbito tecnológico, existen plataformas de gestión del aprendizaje (Learning Management Systems, LMS) como Moodle[2], Canvas[3] y Open edX[4], que han sido utilizadas en diversos entornos educativos para administrar cursos, usuarios y evaluaciones. No obstante, estas soluciones suelen presentar dificultades de adaptación a los requerimientos específicos de instituciones con jerarquías particulares y necesidades de trazabilidad operativa, como es el caso de los Cuerpos de Bomberos.

En consecuencia, se evidencia la necesidad de desarrollar una solución tecnológica a medida, que centralice la gestión académica y garantice seguridad, escalabilidad y adaptabilidad a los procesos internos de la institución.

1.2 Definición del problema.

Actualmente, la gestión académica de Bomberos de Chile presenta importantes limitaciones debido a la ausencia de un sistema unificado que consolide cursos, módulos, instructores, alumnos y certificaciones en una sola plataforma. Los procesos se realizan mayoritariamente mediante documentos independientes, planillas manuales y comunicaciones informales, lo que genera inconsistencias, duplicación de datos y dificultades para mantener información actualizada de manera confiable.

Esta falta de integración afecta directamente a la institución en varios niveles: dificulta la trazabilidad del progreso académico, complica la gestión de inscripciones, impide obtener reportes oportunos para la toma de decisiones y limita la capacidad de los instructores para monitorear el desempeño de los alumnos. Además, los estudiantes carecen de una herramienta que centralice su historial académico y les permita visualizar su avance de manera clara.

El problema central identificado es la carencia de un sistema académico digital, centralizado y escalable, capaz de administrar de forma eficiente la información académica institucional y soportar procesos fundamentales como registro, seguimiento, evaluación y certificación de los bomberos. Esta brecha tecnológica afecta la eficiencia operativa y el desarrollo académico de los usuarios, evidenciando la necesidad de una solución tecnológica integral.

1.3 Breve descripción sobre la propuesta de solución.

Como respuesta a las limitaciones actuales en la gestión académica de los Cuerpos de Bomberos de Chile, se propone el desarrollo de un sistema backend modular y centralizado que permita administrar de forma unificada la información académica institucional. La solución considera funcionalidades esenciales como la gestión de usuarios y roles, diferenciando perfiles de administradores, instructores y bomberos; la administración de cursos y módulos, permitiendo inscripciones, control de asistencia y seguimiento del progreso; y el almacenamiento estructurado de certificaciones y evaluaciones, asegurando la trazabilidad de los procesos formativos.

El sistema incorporará mecanismos de autenticación y control de accesos para resguardar la información y garantizar que cada usuario acceda únicamente a las funciones correspondientes a su perfil. La información será gestionada mediante una base de datos estructurada, proporcionando integridad y consistencia en los registros. Asimismo, el backend estará preparado para integrarse con interfaces de usuario externas, facilitando la interacción de forma clara y eficiente.

En conjunto, esta propuesta busca optimizar la administración académica, mejorar la trazabilidad del aprendizaje y establecer una plataforma escalable y segura, capaz de adaptarse a futuras necesidades institucionales y a la incorporación de nuevas funcionalidades.

1.4 Objetivos Generales y Específicos de la Tesina.

1.4.1 Objetivos General.

Diseñar e implementar un backend robusto, modular y seguro que permita a los Cuerpos de Bomberos de Chile gestionar de manera centralizada la información académica, superando las limitaciones de los procesos manuales y herramientas dispersas mediante mecanismos modernos de autenticación, autorización y trazabilidad de datos, y estableciendo una base tecnológica confiable y escalable para futuras integraciones.

1.4.2 Objetivos Específicos.

1. **Implementar un sistema de autenticación y autorización** que permita establecer un control seguro de accesos diferenciados por rol (administrador, instructor, bombero). Esto asegurará que cada usuario acceda únicamente a las funcionalidades que le correspondan, reduciendo riesgos de seguridad y malas prácticas en el manejo de datos.
2. **Diseñar, desarrollar y documentar una interfaz de servicios (API)** que incluya los principales módulos de gestión académica: usuarios, roles, cursos, evaluaciones y certificados. Esta interfaz permitirá la interoperabilidad con aplicaciones de usuario y la integración futura con otros sistemas o plataformas de apoyo a la formación.
3. **Optimizar el acceso y la gestión de los datos académicos**, aplicando principios de organización y buenas prácticas de modelado, con el objetivo de garantizar integridad, consistencia y eficiencia en el almacenamiento de la información.

4. **Implementar estrategias de validación y manejo de errores** que permitan prevenir inconsistencias en los datos, mejorar la experiencia de uso y aumentar la confiabilidad del sistema en escenarios de alta demanda.
5. **Asegurar la integración entre la capa de servicios y la interfaz de usuario**, garantizando una comunicación estable y segura que permita que la plataforma sea utilizada de manera práctica y accesible por bomberos, instructores y administradores.
6. **Validar la solución mediante pruebas técnicas** (funcionales, de seguridad y de rendimiento), que permitan comprobar la robustez del sistema, verificando el cumplimiento de los requerimientos, la resistencia frente a vulnerabilidades comunes y la capacidad de respuesta bajo condiciones de alta concurrencia.
7. **Establecer un enfoque de desarrollo modular**, que facilite la incorporación de nuevas funcionalidades en el futuro, como el seguimiento avanzado del progreso académico, la automatización de reportes y la integración con herramientas externas de formación o gestión.

1.5 Justificación del proyecto.

La presente sección expone las razones que sustentan el desarrollo del proyecto, abordando su importancia desde una perspectiva técnica y su aporte a la mejora de los procesos académicos de los Cuerpos de Bomberos de Chile. A continuación, se analizan los principales aspectos que justifican la implementación de la solución propuesta, considerando su impacto en la eficiencia, seguridad y evolución del sistema.

1.5.1 Relevancia Técnica.

La gestión académica de los Cuerpos de Bomberos de Chile enfrenta limitaciones debido a la ausencia de un sistema centralizado que asegure eficiencia y trazabilidad en los procesos de capacitación. Desde el punto de vista técnico, la construcción de un sistema modular resulta relevante porque permite organizar la lógica de negocio en componentes claros, reutilizables y fáciles de mantener.

Un sistema de este tipo permitirá la comunicación fluida entre la capa de servicios y futuras aplicaciones de usuario (por ejemplo, interfaces web o móviles), asegurando independencia entre componentes y capacidad de escalar en el tiempo. Asimismo, la incorporación de mecanismos de autenticación y autorización sólidos es un aspecto clave para resguardar la seguridad de la información y garantizar que cada usuario acceda únicamente a lo que le corresponde.

1.5.2 Beneficios.

Los beneficios esperados de este proyecto se proyectan hacia:

1. **Centralización de la información**: un sistema único para usuarios, cursos, evaluaciones y certificaciones.
2. **Mejora en la trazabilidad**: seguimiento claro y confiable del progreso académico de los bomberos.
3. **Optimización del tiempo administrativo**: reducción de tareas manuales y repetitivas.
4. **Mayor seguridad de datos**: control de accesos por roles y protección de información sensible.

5. **Escalabilidad futura**: posibilidad de extender la solución con nuevas funcionalidades o integraciones.

1.5.3 Innovación

La innovación del proyecto radica en proponer un sistema académico centralizado en un contexto donde la gestión se ha realizado tradicionalmente mediante procesos manuales o herramientas poco integradas. Aunque el sistema se concibe inicialmente bajo una arquitectura monolítica modular, se proyecta hacia un futuro escalamiento en entornos más avanzados, lo que abre la puerta a una solución sostenible y adaptable a largo plazo.

En este sentido, el proyecto constituye no solo una respuesta a una necesidad institucional, sino también un aporte académico y técnico al área del desarrollo de software, al integrar buenas prácticas, lineamientos modernos y proyecciones de mejora continua.

1.6 Metodología.

Para la construcción del sistema se adoptó un enfoque ágil, con el objetivo de garantizar flexibilidad, adaptación a cambios y entregas incrementales de valor. Dentro de este enfoque, se decidió utilizar una combinación de Scrum [5] y Kanban [6]**,** dos de las metodologías más empleadas en el desarrollo de software moderno.

Scrum se aplicó como marco de trabajo principal, estructurando el desarrollo en iteraciones cortas en las que se definieron objetivos específicos. Esto permitió entregar avances de manera progresiva, validar los resultados obtenidos y realizar ajustes en función de la retroalimentación recibida, asegurando que el proyecto evolucionara de forma controlada.

Por otra parte, Kanban se utilizó de manera complementaria, a través de un sistema visual de seguimiento de tareas. Este tablero permitió dar visibilidad al flujo de trabajo y al estado de cada actividad, favoreciendo la organización del equipo y la detección temprana de posibles cuellos de botella.

En complemento a las metodologías ágiles, se emplearon estrategias de control de versiones basadas en un flujo de trabajo por ramas [7]. Este esquema de gestión colaborativa permitió mantener la trazabilidad de los cambios, evitar conflictos en la integración y asegurar que solo los avances probados fueran incorporados a la versión principal del sistema.

La combinación de Scrum, Kanban y control de versiones estructurado proporcionó un marco de trabajo iterativo, organizado y colaborativo, que facilitó la planificación, el seguimiento y la integración continua de avances. Esto resultó especialmente útil en un proyecto académico, donde los tiempos son limitados y los requerimientos pueden evolucionar durante el desarrollo.

En conclusión, la metodología seleccionada aseguró que el sistema se desarrollara de forma adaptable, incremental y de acuerdo con buenas prácticas de ingeniería de software, contribuyendo a la calidad del producto final.

1.7 Organización del informe en capítulos.

El presente documento se encuentra estructurado en capítulos que siguen una secuencia lógica, facilitando la comprensión del desarrollo del proyecto desde su planteamiento inicial hasta las conclusiones.

- **Capítulo 1 – Introducción:** Presenta una visión general del proyecto, abordando su contexto, justificación, objetivos generales y específicos, así como la metodología empleada durante su desarrollo.
- **Capítulo 2 – Marco Teórico:** Expone los fundamentos conceptuales y técnicos que sustentan el proyecto. Incluye la descripción de los principios del desarrollo backend, las arquitecturas de software relevantes, y las tecnologías y frameworks considerados para la implementación del sistema.
- **Capítulo 3 – Diseño e Implementación:** Detalla el diseño técnico del sistema, incorporando diagramas de arquitectura, modelos de datos, y flujos de trabajo. Asimismo, describe las etapas de implementación, especificando las herramientas, lenguajes de programación y técnicas utilizadas para la construcción del backend.
- **Capítulo 4 – Conclusiones:** Expone las conclusiones obtenidas a partir del desarrollo del proyecto, junto con una reflexión crítica sobre los resultados alcanzados, las limitaciones identificadas y las proyecciones futuras.
- **Capítulo 5 – Agradecimientos:** Sección destinada al reconocimiento de las personas o instituciones que contribuyeron de manera significativa al desarrollo del trabajo.
- **Capítulo 6 – Referencias:** Incluye el listado de fuentes bibliográficas, artículos y recursos digitales consultados del proyecto.
- **Capítulo 7 – Anexos:** Contiene material complementario relevante para el proyecto, como diagramas extendidos, fragmentos de código, capturas de pantalla o documentación técnica adicional.

2 Marco Teórico

En este capítulo se presentan los fundamentos teóricos y tecnológicos que sirven como base para el desarrollo de sistemas backend modernos. El propósito es exponer un panorama general de las arquitecturas, herramientas y metodologías comúnmente utilizadas en la industria, permitiendo identificar distintas alternativas antes de seleccionar las tecnologías aplicadas en el proyecto.

Se revisan conceptos esenciales relacionados con el desarrollo backend, su rol dentro de la arquitectura de software y las estructuras más utilizadas para organizar aplicaciones, tales como la arquitectura monolítica, la arquitectura en capas, los enfoques serverless [8] y los microservicios [9]. Asimismo, se describen patrones de diseño relevantes como MVC, MVT y REST, los cuales permiten separar responsabilidades, facilitar la escalabilidad y estandarizar la comunicación entre componentes [10].

También se analizan tecnologías ampliamente empleadas en entornos backend, incluyendo frameworks orientados a APIs, motores de bases de datos relacionales y no relacionales, contenedores para despliegue y mecanismos generales de autenticación y autorización. Este apartado incorpora además una visión sobre metodologías ágiles de desarrollo, como Scrum, que permiten organizar proyectos de software mediante iteraciones cortas y colaborativas [11].

El objetivo de este marco es ofrecer una visión general del ecosistema tecnológico disponible para la construcción de un backend académico, sin seleccionar aún una tecnología específica. Estas bases conceptuales servirán posteriormente para justificar las decisiones técnicas adoptadas en el diseño e implementación del sistema.

2.1 Fundamentos del desarrollo backend.

El desarrollo backend constituye un elemento esencial en los sistemas y aplicaciones modernas, ya que se encarga de ejecutar la lógica de negocio, procesar datos y gestionar la interacción con bases de datos institucionales. A medida que los sistemas informáticos aumentaron en complejidad durante la década de 1990, surgió la necesidad de plataformas capaces de soportar un mayor volumen de usuarios, transacciones y operaciones internas. Inicialmente, los backends adoptaron arquitecturas monolíticas básicas, pero con el tiempo evolucionaron hacia enfoques más estructurados que incorporan mecanismos de escalabilidad, seguridad y modularidad.

2.1.1 El rol del backend en la Arquitectura de Software.

Dentro de la arquitectura de software, el backend cumple funciones esenciales que, aunque no visibles para el usuario final, determinan el funcionamiento del sistema. Entre sus principales responsabilidades se encuentran:

- **Procesamiento de datos:** Manejar solicitudes provenientes de aplicaciones web o móviles, procesar la información y entregar respuestas de forma estructurada.
- **Lógica de negocio:** Aplicar reglas y validaciones que aseguren que las operaciones académicas (como inscripciones, evaluaciones, roles o certificaciones) se ejecuten correctamente.
- **Manejo de bases de datos:** Controlar el almacenamiento, consulta y actualización de información en bases de datos relacionales, garantizando integridad y consistencia.

En conjunto, estos procesos permiten que la aplicación opere de forma estable, consistente y segura, incluso en escenarios de alta demanda o concurrencia.

2.1.2 Tendencias de la Industria.

El desarrollo backend ha incorporado diversas tecnologías y enfoques orientados a mejorar la eficiencia y la mantenibilidad de los sistemas. Entre las tendencias más relevantes se encuentran:

- **Arquitecturas modulares:** Permiten dividir el sistema en componentes independientes, facilitando su mantenimiento y evolución.
- **Contenedores:** Herramientas como Docker [12] posibilitan empaquetar aplicaciones con sus dependencias en un entorno consistente, facilitando el despliegue y asegurando un comportamiento uniforme entre desarrollo y producción.
- **Servicios en la nube [13]:** Plataformas como AWS ofrecen servicios de almacenamiento, cómputo y despliegue, lo que permite implementar sistemas más flexibles y con mayor disponibilidad.
- **Evolución hacia arquitecturas distribuidas:** Si bien el proyecto utiliza un backend monolítico modular, las tendencias actuales apuntan a sistemas capaces de escalar mediante microservicios o servicios serverless.

Estas tendencias han influido directamente en el diseño de sistemas académicos modernos, permitiendo mayor confiabilidad y capacidad de crecimiento.

2.1.3 Usos del Backend.

El backend es fundamental en cualquier sistema que dependa del manejo estructurado de datos o reglas internas complejas. Sus principales aplicaciones incluyen:

- Plataformas académicas para gestionar cursos, usuarios, certificaciones e inscripciones.
- Sistemas empresariales que administran información interna y procesos administrativos.
- Aplicaciones móviles y web que requieren comunicación constante con servicios externos o bases de datos.
- Portales de organizaciones que necesitan disponibilidad continua y seguridad en el manejo de datos sensibles.

2.1.4 Ventajas y Desventajas.

Ventajas

- **Escalabilidad:** Permite implementar sistemas capaces de manejar mayor cantidad de usuarios y operaciones.
- **Separación de responsabilidades:** Divide claramente la lógica interna del manejo de datos y la interfaz de usuario.
- **Integración:** Facilita la comunicación con servicios externos, APIs y mecanismos de autenticación.

Desventajas

- **Complejidad:** Requiere arquitectura bien definida y un diseño robusto, especialmente en sistemas con múltiples módulos.
- **Seguridad:** Demanda implementar mecanismos estrictos para proteger información sensible, especialmente datos académicos y personales.
- **Manejo de errores:** La detección y corrección de fallos en componentes internos puede ser difícil en sistemas con alta concurrencia.

2.2 Arquitectura de Software.

La arquitectura de software se refiere al conjunto de decisiones estructurales que definen la organización de un sistema y la forma en que interactúan sus componentes. Estas decisiones influyen directamente en atributos de calidad como la mantenibilidad, escalabilidad, seguridad y control de los flujos de información [9], siendo especialmente relevantes en sistemas académicos y plataformas de gestión institucional. Una arquitectura bien definida permite separar responsabilidades, reducir el acoplamiento entre componentes y facilitar la evolución del sistema a lo largo del tiempo.

En el contexto de sistemas backend orientados a cliente web, la arquitectura en capas constituye un enfoque ampliamente utilizado, ya que organiza el sistema en niveles con responsabilidades claramente delimitadas. Este modelo distingue la interacción con el usuario, la orquestación de solicitudes, la lógica de negocio y el acceso a los datos, promoviendo un diseño modular y comprensible.

2.2.1 Arquitecturas utilizadas en sistemas backend.

1) Arquitectura monolítica modular

En este enfoque, la aplicación se ejecuta como una única unidad lógica, pero organizada internamente en módulos que agrupan funcionalidades específicas, tales como gestión de usuarios, administración académica o manejo de notificaciones [9].

- **Ventajas:** simplicidad de implementación, despliegue rápido y menor costo de infraestructura.
- **Desventajas:** menor flexibilidad para escalar y mayor dependencia entre componentes.

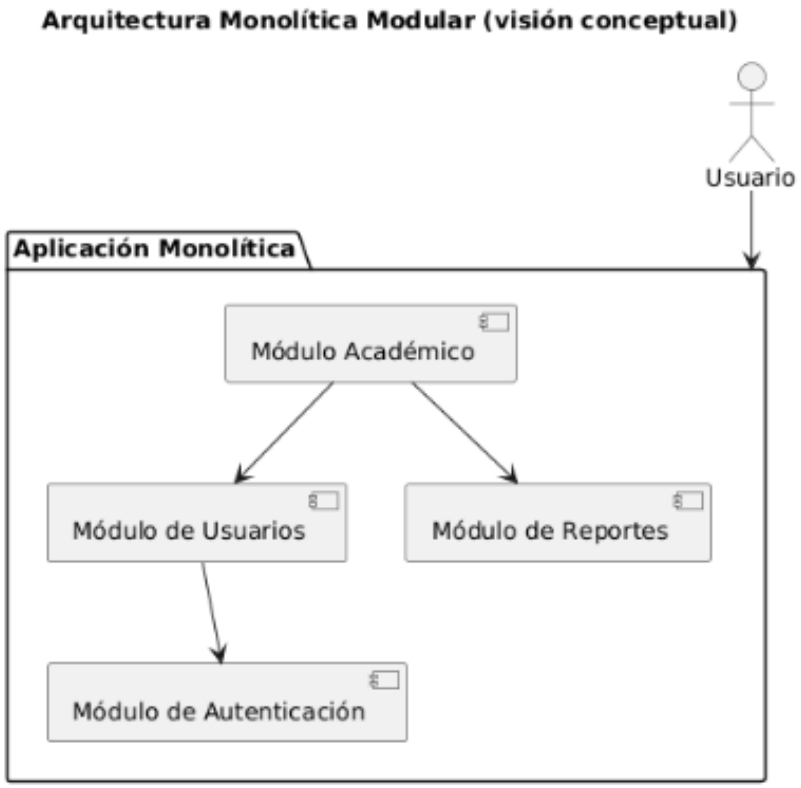


Figura 2-1. Arquitectura Monolítica Modular.
Fuente: Elaboración propia.

2) Arquitectura en capas

Uno de los enfoques más utilizados en sistemas empresariales y académicos es la arquitectura en capas, la cual separa el sistema en niveles con funciones específicas:

1. **Capa de presentación:** donde interactúan los usuarios a través de aplicaciones web o móviles.
2. **Capa de lógica de negocio:** procesa reglas académicas, validaciones y flujos administrativos.

3. **Capa de persistencia:** encargada del acceso y almacenamiento en bases de datos.

La separación de responsabilidades facilita las pruebas, el mantenimiento y la evolución del sistema [11].

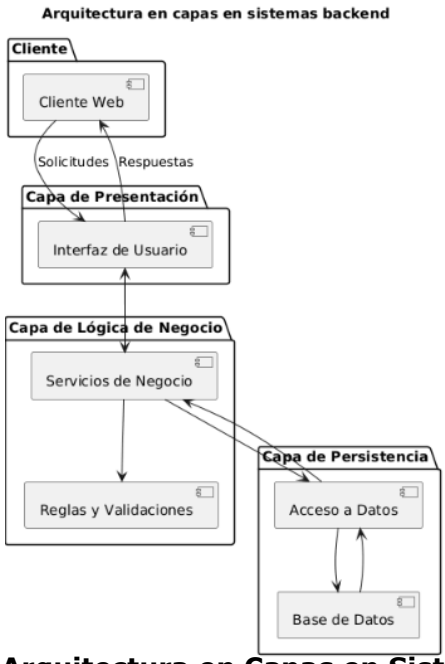


Figura 2-2. Arquitectura en Capas en Sistemas Backend.
Fuente: Elaboración propia.

2.2.2 Patrones de diseño relevantes.

1) Patrón MVT (Model–View–Template)

Presente en frameworks como Django, el patrón MVT organiza los sistemas web en tres componentes:

- **Model:** estructura de datos académicos.
- **View:** procesamiento y reglas del sistema.
- **Template:** componente de presentación.

Garantiza modularidad, claridad en el flujo de datos y reutilización del código [11].

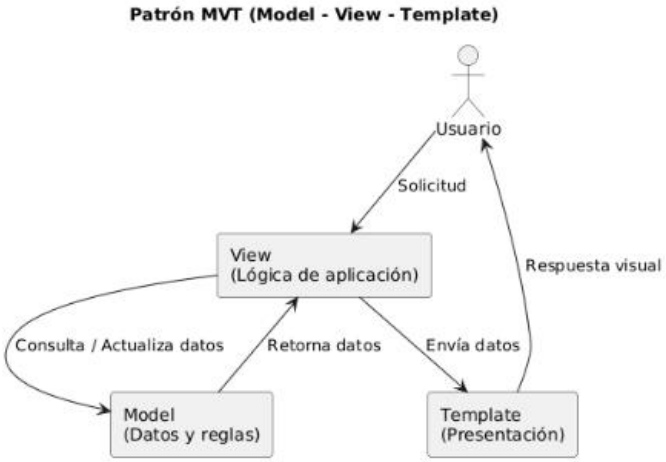


Figura 2-3. Patron MVT (Model-View-Template).
Fuente: Elaboración propia.

2.2.3 Estilos arquitectónicos de comunicación.

1) APIs RESTful

REST es un estándar ampliamente utilizado para conectar frontends, aplicaciones móviles y sistemas externos. Sus principios cliente-servidor, ausencia de estado y acceso mediante URL permiten construir servicios escalables, ligeros y fáciles de integrar [3][4].

Beneficios: interoperabilidad, flexibilidad, mantenimiento simple y soporte para múltiples tipos de clientes.

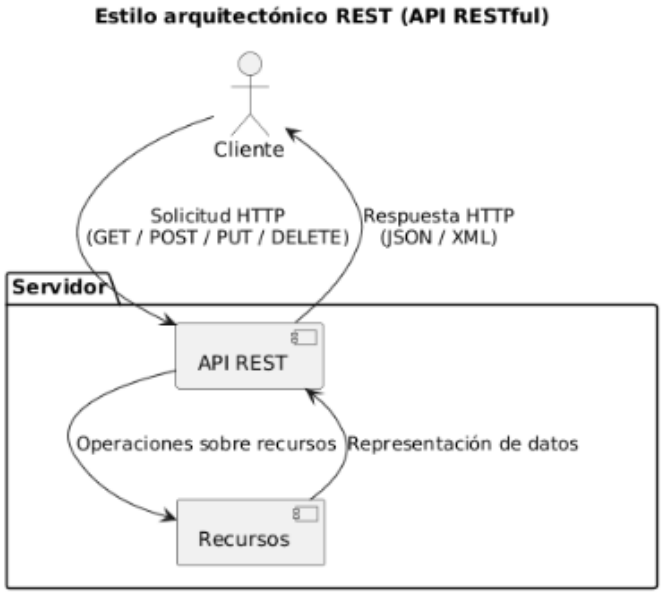


Figura 2-4. Estilo de comunicación APIs RESTful.
Fuente: Elaboración propia.

2.2.4 Arquitecturas emergentes y proyección futura.

Además de los enfoques tradicionales, existen arquitecturas que están ganando relevancia en la industria:

- **Microservicios:** permiten dividir el sistema en servicios independientes con despliegue autónomo.
- **Serverless:** posibilitan ejecutar funciones bajo demanda sin gestionar servidores.

Estas alternativas proyectan caminos de escalabilidad para sistemas académicos que crecen en volumen de datos o número de usuarios [8][9].

2.3 Tecnologías y frameworks utilizados.

El desarrollo de un backend moderno puede abordarse a través de diversas tecnologías, lenguajes y frameworks que ofrecen características específicas según los requerimientos del sistema. En esta sección se presentan las alternativas más utilizadas en la industria para la construcción de APIs, la gestión de datos, la seguridad de usuarios y la infraestructura de despliegue. Este análisis sirve como base conceptual antes de seleccionar las herramientas definitivas para la implementación del sistema académico.

2.3.1 Lenguaje de programación.

Los lenguajes más utilizados para construir sistemas backend escalables son:

- **Python [14]**: Conocido por su sintaxis simple y su ecosistema maduro, es ampliamente utilizado en plataformas académicas, científicas y de administración institucional.
- **JavaScript / TypeScript (Node.js)**: Su ejecución del lado del servidor permite aplicaciones altamente concurrentes. Es una opción común en arquitecturas basadas en microservicios y aplicaciones de alta demanda.
- **Java y C#:** Lenguajes robustos, típicamente utilizados en soluciones empresariales de gran escala, con herramientas avanzadas de seguridad, concurrencia y despliegue corporativo.

Cada uno de ellos permite construir APIs RESTful y manejar grandes volúmenes de datos, siendo adecuados para sistemas de gestión académica.

2.3.2 Framework principal.

Los frameworks proporcionan herramientas que simplifican la implementación de endpoints, validaciones y lógica de negocio. Entre los más relevantes se encuentran:

- **Django REST Framework (Python) [15]:** Ofrece serialización automática, manejo de permisos, autenticación integrada y una arquitectura modular orientada a datos.
- **Express.js / NestJS (JavaScript / TypeScript)**: Frameworks flexibles, utilizados en plataformas de alta concurrencia. NestJS incorpora arquitectura en módulos y principios de diseño orientados a empresas.
- **Spring Boot (Java)**: Framework robusto para aplicaciones empresariales con gestión avanzada de transacciones, seguridad y despliegue en producción.

Estas herramientas permiten construir APIs REST estandarizadas, escalables y compatibles con clientes web o móviles.

2.3.3 Base de datos.

La persistencia de datos es un componente clave en plataformas académicas, donde se requiere integridad, relación entre entidades y trazabilidad. Los sistemas más utilizados son:

- **Relacionales (SQL):**

1) PostgreSQL [16]
2) MySQL
3) SQLite (usado en entornos de desarrollo)

Estos motores permiten modelar datos estructurados mediante relaciones entre usuarios, cursos, módulos, certificados e inscripciones.

- **No relacionales (NoSQL):**
  1) MongoDB
  2) Firebase Firestore

Utilizados principalmente en aplicaciones con estructuras de datos más flexibles o con necesidades de escalabilidad horizontal.

2.3.4 Mecanismos de seguridad.

Los sistemas backend deben asegurar el acceso controlado a la información. Entre los mecanismos más utilizados se encuentran:

- **JWT (JSON Web Token) [17]**: Token ligero utilizado para identificar a un usuario en cada solicitud dentro de sistemas distribuidos.
- **OAuth 2.0**: Protocolo usado para autenticación delegada mediante proveedores externos (Google, Facebook, etc.).
- **Sesiones tradicionales**: Utilizan cookies y almacenamiento de sesión en servidor; apropiadas para sistemas pequeños o de ámbito local.

2.3.5 Comunicación con el frontend.

La comunicación entre el backend y la interfaz de usuario es un elemento fundamental en arquitecturas modernas, ya que permite la interoperabilidad entre aplicaciones web, móviles o de escritorio. Entre los mecanismos más utilizados destacan:

- **RESTful API**: basada en operaciones HTTP y uso de JSON como formato de intercambio. Es la alternativa más extendida debido a su simplicidad, escalabilidad y compatibilidad con múltiples plataformas.
- **GraphQL**: permite realizar consultas flexibles y optimizadas, adecuadas para sistemas que requieren obtener datos muy específicos o reducir el número de solicitudes.
- **WebSockets**: habilita comunicación bidireccional en tiempo real, útil para aplicaciones que requieren actualización inmediata, como chats o sistemas de monitoreo.

2.3.6 Testing y validación

Las pruebas de software constituyen un proceso fundamental para asegurar la calidad, estabilidad y confiabilidad de un sistema. En el desarrollo backend, estas pruebas se agrupan generalmente en cuatro categorías:

- **Pruebas unitarias:** verifican el correcto funcionamiento de funciones o módulos individuales.
- **Pruebas de integración:** evalúan la interacción entre componentes o servicios.

- **Pruebas funcionales:** confirman que el sistema cumple los requerimientos definidos por los usuarios.
- **Pruebas de rendimiento:** analizan la capacidad de respuesta del sistema bajo diferentes niveles de carga o concurrencia.

2.3.7 Control de versiones y CI/CD.

El control de versiones es un componente esencial en el desarrollo moderno de software, pues permite administrar el código de forma colaborativa, mantener un historial de cambios y facilitar la integración de nuevas funcionalidades. Git se ha consolidado como el estándar de la industria, soportado por plataformas como GitHub [18], GitLab y Bitbucket, que proporcionan herramientas para revisar cambios, gestionar ramas y coordinar el trabajo entre distintos desarrolladores.

Complementariamente, las prácticas de Integración Continua y Despliegue Continuo (CI/CD) permiten automatizar pruebas, integraciones y despliegues, garantizando entregas más rápidas y con mayor estabilidad. Soluciones como GitHub Actions, GitLab CI/CD o Jenkins permiten ejecutar pipelines que validan el estado del proyecto, verifican la calidad del código y despliegan nuevas versiones de manera controlada.

En el contexto del proyecto, se considera adecuado el uso de un flujo de trabajo basado en ramas estructuradas —como Gitflow— junto con la futura incorporación de pipelines CI/CD para automatizar procesos críticos de validación y despliegue.

2.3.8 Entorno de despliegue

El entorno de despliegue corresponde a la infraestructura donde se ejecuta el sistema en producción. Entre las alternativas más utilizadas se encuentran los servidores locales, utilizados comúnmente en etapas iniciales; los contenedores Docker, que permiten empaquetar aplicaciones y asegurar consistencia entre entornos; y las plataformas en la nube como AWS, Azure o Google Cloud, que ofrecen escalabilidad, alta disponibilidad y flexibilidad operativa.

Estas opciones representan diferentes niveles de complejidad, costo y capacidad de crecimiento. En el contexto de plataformas académicas, es habitual iniciar con entornos locales y evolucionar posteriormente hacia soluciones contenedorizadas o infraestructura en la nube, a medida que aumentan los requerimientos de disponibilidad y escalabilidad.

2.3.9 Tecnologías emergentes.

En los últimos años han surgido arquitecturas emergentes que buscan aumentar la escalabilidad y la resiliencia:

- **Microservicios** → dividen el sistema en servicios independientes, facilitando la modularidad.
- **Serverless** → permite ejecutar funciones en la nube bajo demanda.

3 Diseño e Implementación.

En este capítulo se presenta el diseño y desarrollo del sistema backend orientado a centralizar y gestionar los procesos académicos de los Cuerpos de Bomberos. Se describe la arquitectura en capas utilizada, detallando los componentes principales y las funcionalidades implementadas en cada módulo del sistema, tales como la administración de usuarios y roles, la gestión de cursos y módulos formativos, el registro de inscripciones y la carga de certificados.

Este capítulo se organiza en torno a los requerimientos del sistema, el diseño técnico de la solución, la implementación de sus componentes principales y el conjunto de pruebas aplicadas para validar su funcionamiento.

Antes de abordar el diseño e implementación del backend, se presentan los requerimientos funcionales y no funcionales que guiaron el desarrollo del sistema. Estos requerimientos definen el comportamiento esperado del backend y establecen los criterios técnicos necesarios para garantizar seguridad, trazabilidad, escalabilidad y consistencia en la gestión académica de los Cuerpos de Bomberos.

Requerimientos Funcionales.

RF-01 Gestión de usuarios y roles: El sistema académico backend DEBE proporcionar al administrador del sistema la capacidad de crear, modificar, eliminar y consultar usuarios, así como asignar roles que determinen los permisos de acceso y las acciones disponibles.
RF-02 Administración de cursos y módulos: El sistema académico backend DEBE proporcionar al administrador académico la capacidad de registrar cursos, estructurarlos en módulos formativos y gestionar su información asociada.
RF-03 Inscripciones académicas: El sistema académico backend DEBE proporcionar al usuario bombero la capacidad de inscribirse en cursos y módulos formativos, validando los prerrequisitos definidos cuando corresponda.
RF-04 Gestión de certificados: El sistema académico backend DEBE proporcionar al usuario bombero la capacidad de consultar y descargar certificados académicos asociados a su historial formativo.
RF-05 Control de acceso según perfil: El sistema académico backend DEBE restringir automáticamente el acceso a las funcionalidades del sistema según el rol asignado al usuario autenticado.
RF-06 Sistema de notificaciones: El sistema académico backend DEBE proporcionar al administrador académico o instructor la capacidad de enviar y consultar notificaciones internas asociadas a cursos, módulos o roles académicos.

Requerimientos No Funcionales.

RNF-01 Seguridad: El sistema académico backend DEBE implementar mecanismos de autenticación y autorización basados en JWT, asegurando que el 100% de las solicitudes protegidas requieran credenciales válidas y que los datos sensibles se transmitan de forma segura.
RNF-02 Escalabilidad: El sistema académico backend DEBE permitir la incorporación de nuevos módulos e integraciones sin degradar los tiempos de respuesta en más de un 20% respecto a la línea base, manteniendo una tasa de errores inferior al 1% y soportando al menos 100 usuarios concurrentes.

RNF-03 Rendimiento: El sistema académico backend DEBE mantener tiempos de respuesta inferiores a 500 ms en operaciones de consulta bajo condiciones normales de uso y soportar al menos 100 usuarios concurrentes sin fallos críticos.
RNF-04 Confiabilidad: El sistema académico backend DEBE asegurar la integridad y consistencia de los datos, manteniendo una tasa de errores de transacción inferior al 1% durante las operaciones de creación, actualización y consulta.
RNF-05 Mantenibilidad: El sistema académico backend DEBE estar organizado de forma modular, permitiendo la incorporación de nuevas funcionalidades o correcciones sin requerir modificaciones en más del 20% del código existente.
RNF-06 Disponibilidad: El sistema académico backend DEBE poder desplegarse en entornos contenedorizados, garantizando una disponibilidad mínima del 99% en condiciones normales de operación.

Estos requerimientos sirvieron como base para el diseño de la arquitectura, la estructura modular del backend y la planificación de los sprints de desarrollo.

3.1 Diseño de Componentes.

En esta sección se describe el diseño de los principales componentes que conforman el backend del sistema académico. Se presentan la arquitectura adoptada, la organización modular del backend y la forma en que los distintos componentes interactúan entre sí para implementar la lógica de negocio definida por los requerimientos del sistema. Asimismo, se detallan los módulos principales, sus responsabilidades y relaciones, junto con los patrones de diseño utilizados para favorecer la mantenibilidad, escalabilidad y claridad estructural del sistema.

3.1.1 Arquitectura y estructura de los componentes.

La plataforma se construyó bajo una arquitectura en capas con enfoque monolítico modular, lo que permite organizar el sistema en componentes independientes, facilitar el mantenimiento y asegurar una futura ampliación de funcionalidades. Esta estructura divide el sistema en tres niveles principales: presentación, lógica de negocio y persistencia de datos.

La capa de presentación, desarrollada externamente en Angular, consume los servicios expuestos por el backend mediante solicitudes HTTP estándar. La capa de lógica de negocio, implementada en Django REST Framework, gestiona procesos como autenticación, administración de usuarios y roles, gestión de cursos, módulos, certificados y notificaciones. Cada funcionalidad se encapsula en módulos separados, favoreciendo una estructura clara y escalable.

Finalmente, la capa de datos se compone de un motor de base de datos relacional que almacena usuarios, cursos, inscripciones y certificados. El acceso a esta capa se realiza mediante un ORM, garantizando integridad y consistencia en el manejo de información. La elección de esta arquitectura se fundamenta en su simplicidad, mantenibilidad y capacidad de evolucionar hacia arquitecturas distribuidas si la demanda del sistema lo requiere.

3.1.1.1 Elección de la arquitectura

Se optó por una arquitectura en capas con enfoque monolítico modular, debido a que permite agrupar la lógica del sistema en secciones bien definidas, facilitando el

mantenimiento, las pruebas y la incorporación futura de nuevas funcionalidades. La solución se estructura en tres capas principales:

- **Capa de presentación (Frontend)**

Corresponde a la aplicación cliente, implementada mediante Angular, la cual consume los servicios expuestos por el backend a través de una API REST. Esta capa gestiona la interacción directa con los usuarios finales —bomberos, instructores y administradores— permitiendo visualizar cursos, módulos, certificados, anuncios, notificaciones y demás elementos de la plataforma. Su construcción en Angular permite una experiencia ágil, modular y adaptable a distintos perfiles de usuario.

- **Capa de lógica de negocio (Backend)**

Implementada utilizando Django REST Framework, esta capa concentra la lógica central del sistema académico. Aquí se definen y procesan las reglas de negocio encargadas de:

  - Autenticación y autorización mediante JSON Web Tokens (SimpleJWT).
  - Gestión de usuarios, roles y perfiles.
  - Administración de cursos, módulos, clases e inscripciones.
  - Registro, almacenamiento y consulta de certificados asociados a cada bombero.
  - Publicación de anuncios y envío de notificaciones internas.

La interacción con el frontend se realiza mediante una API RESTful, utilizando solicitudes HTTP y mensajes en formato JSON, lo que favorece el desacoplamiento y la escalabilidad del sistema.

- **Capa de datos**

Se compone de una base de datos relacional, donde se almacena la información estructurada del sistema: usuarios, perfiles, cursos, módulos, inscripciones, certificados, anuncios y notificaciones.

Para el desarrollo local se utiliza SQLite3, una base de datos ligera que facilita la configuración y el testing.

En entornos de despliegue, se contempla el uso de Cloud SQL (PostgreSQL) como motor principal, debido a su robustez, seguridad, compatibilidad con entornos en la nube y capacidad de escalar según la demanda del sistema.

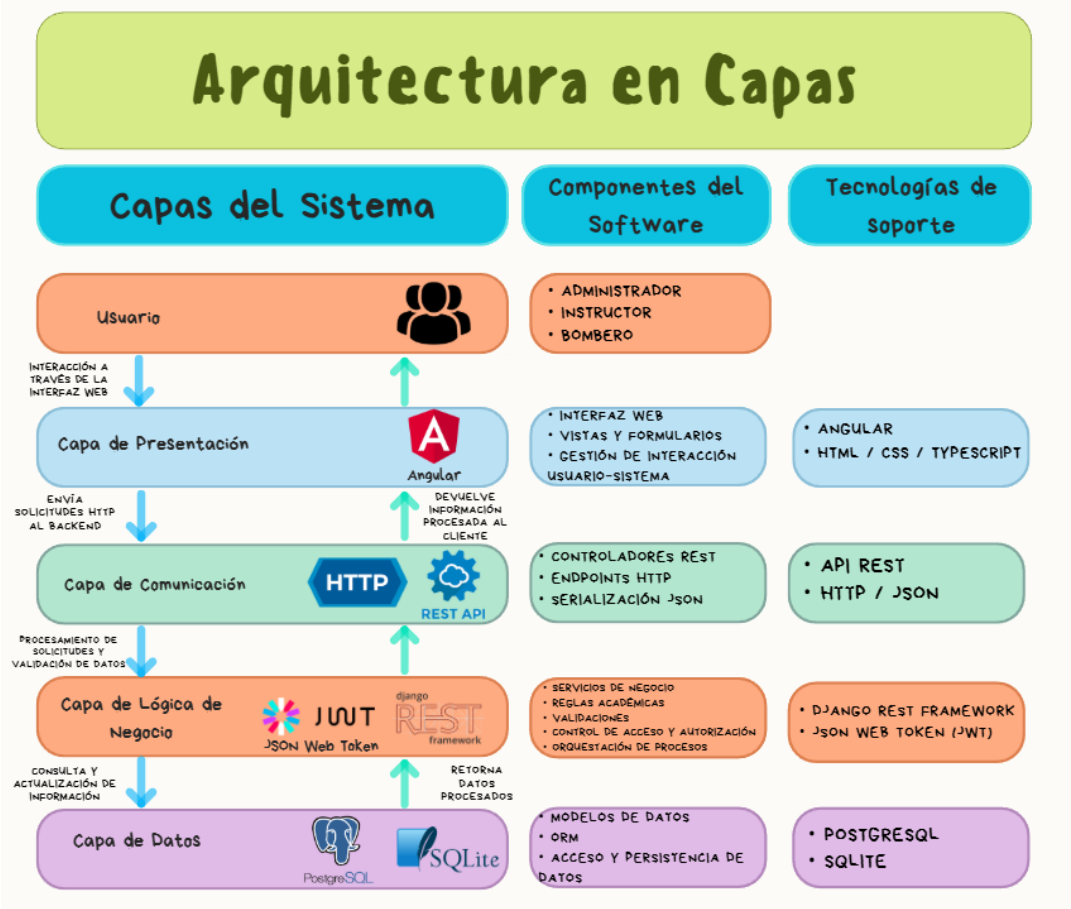


Figura 3-1. Arquitectura en Capas.
Fuente: Elaboración propia.

La arquitectura del sistema se basa en un enfoque por capas que separa de forma clara la presentación, la lógica de negocio y la persistencia de datos. La aplicación cliente, desarrollada en Angular, se comunica con el backend mediante solicitudes HTTP hacia una API REST implementada en Django REST Framework. En esta capa se gestionan procesos como autenticación, administración de usuarios, cursos, módulos y certificados. Finalmente, la información es almacenada en una base de datos relacional, utilizando SQLite3 en desarrollo y PostgreSQL en entornos de despliegue. Esta estructura modular asegura mantenibilidad, escalabilidad y una base sólida para futuras extensiones del sistema.

Para comprender la estructura interna del sistema y la forma en que se gestionan los datos académicos, administrativos y operativos, es necesario describir las entidades que conforman el modelo de información. Este modelo define cómo se organizan los datos, cómo se relacionan entre sí y cómo fluyen a través de los distintos módulos del backend, garantizando consistencia e integridad en todos los procesos.

El sistema utiliza un modelo de datos relacional que incluye entidades como usuarios, roles, cursos, módulos, clases, certificados, anuncios y notificaciones, junto con las relaciones que permiten mantener la trazabilidad académica y administrativa. Esta estructura proporciona una base sólida para las operaciones de consulta, actualización y almacenamiento de información, soportando la lógica de negocio implementada en la plataforma.

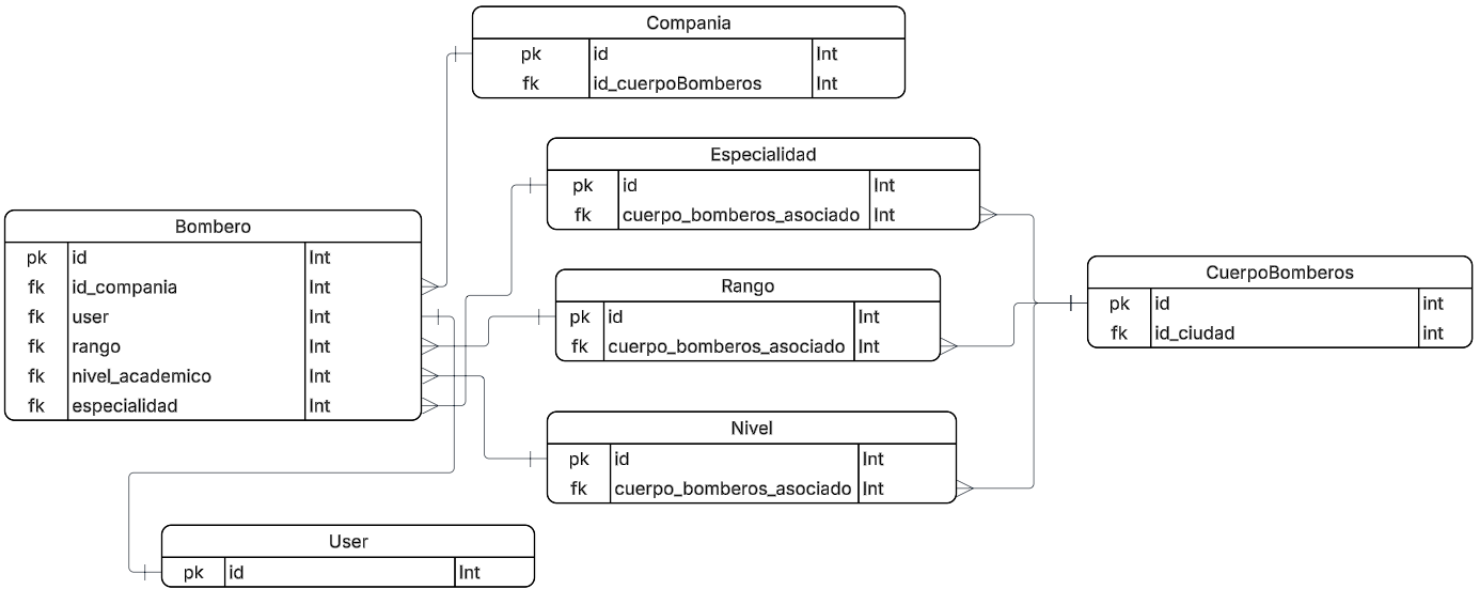


Figura 3-2. Relaciones de las Tablas Correspondientes a Autenticación.
Fuente: Elaboración propia.

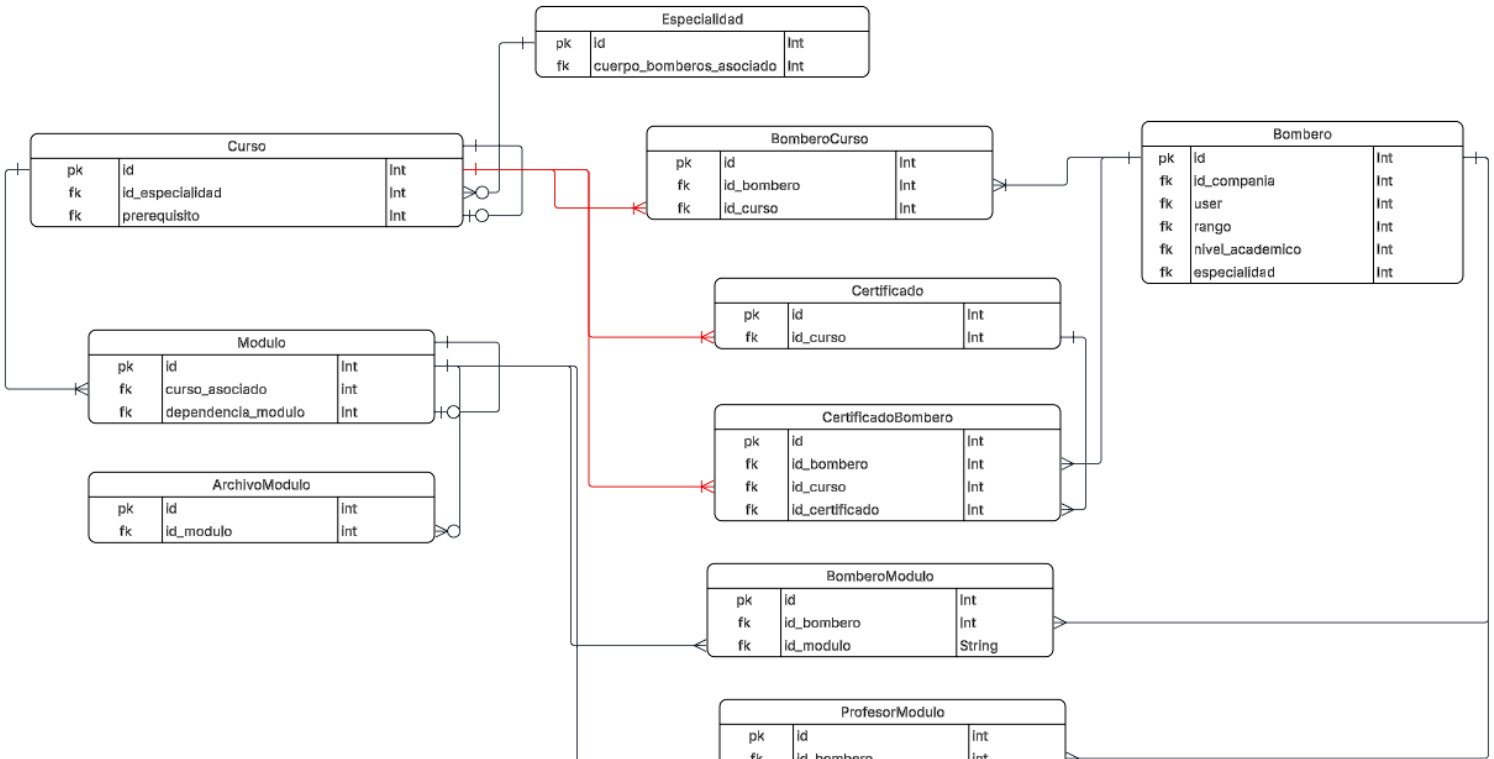


Figura 3-3. Relaciones de las Tablas correspondientes a Formación.
Fuente: Elaboración propia.

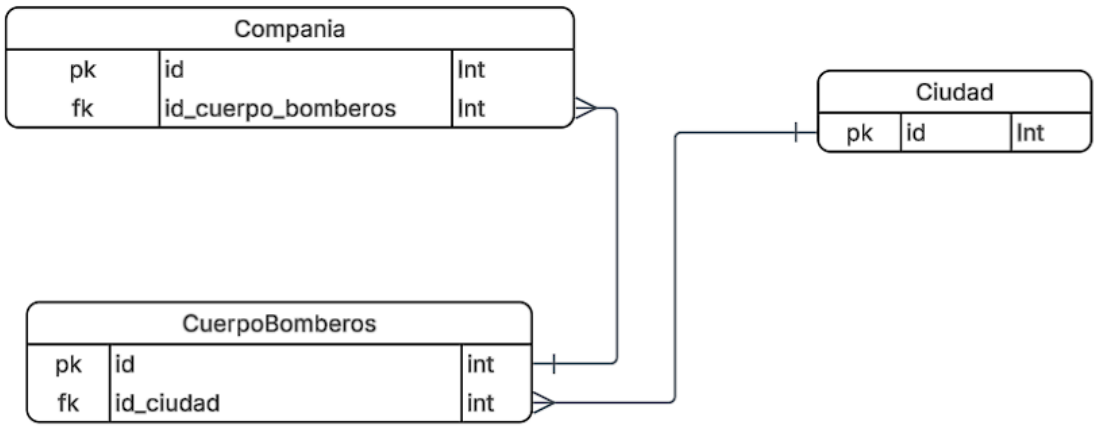


Figura 3-4. Relaciones de las Tablas correspondientes a Organización.
Fuente: Elaboración propia.

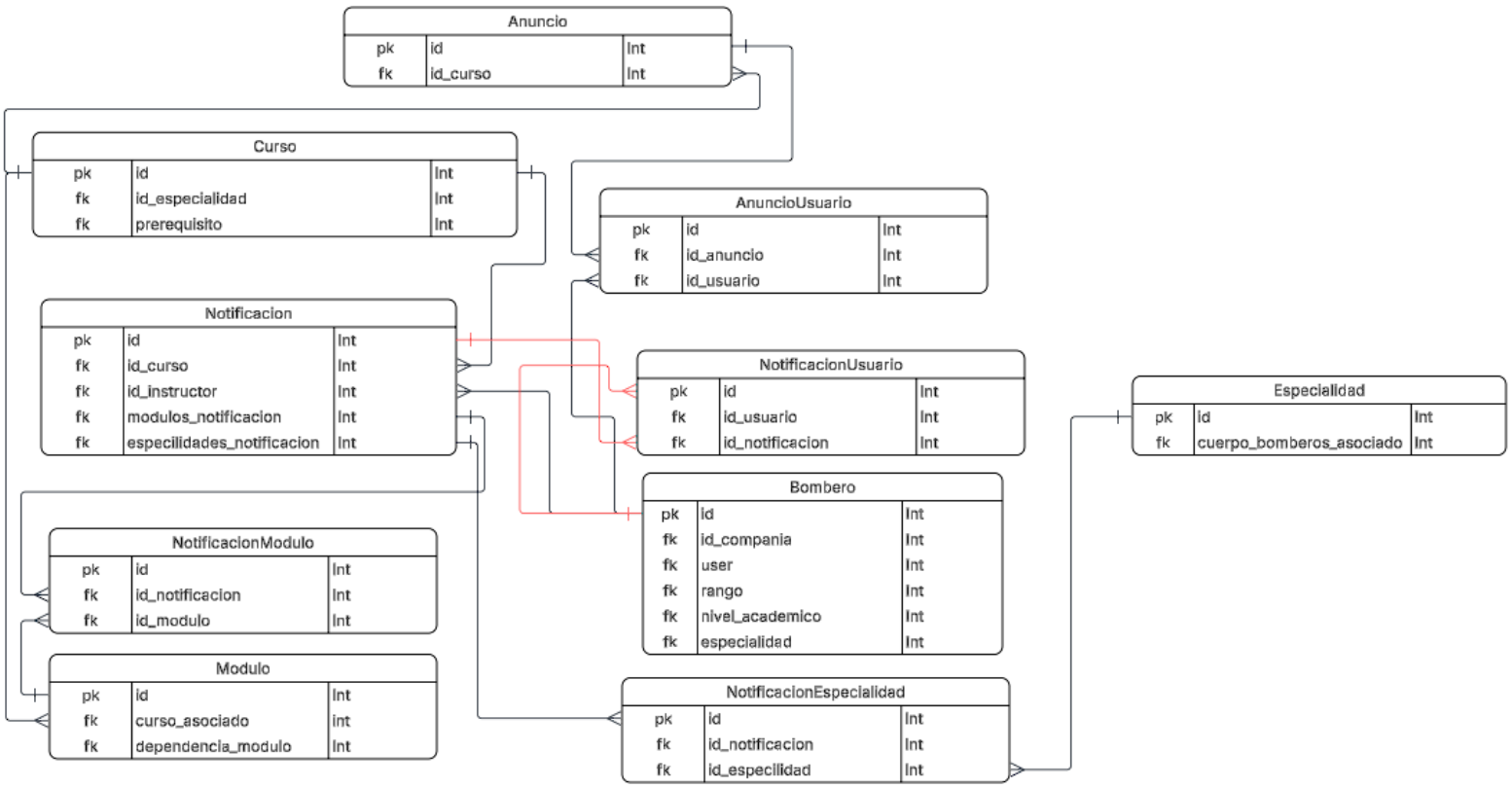


Figura 3-5. Relaciones de las Tablas correspondientes a Avisos.
Fuente: Elaboración propia.

La segunda representación corresponde al diagrama de clases, el cual describe el sistema desde una perspectiva orientada a objetos. A diferencia del modelo relacional, este diagrama muestra la estructura lógica del backend, incorporando las clases principales, sus atributos, métodos relevantes y las relaciones que permiten implementar las reglas de negocio. Esta visión facilita comprender cómo interactúan los componentes internos para gestionar usuarios, roles, cursos, módulos, asistencia, certificados y notificaciones.

Ambas vistas modelo relacional y diagrama de clases se complementan entre sí: mientras una organiza la información en términos de almacenamiento, la otra refleja cómo esa información es manipulada por el código. En conjunto, permiten entender la estructura modular del sistema y su coherencia interna.

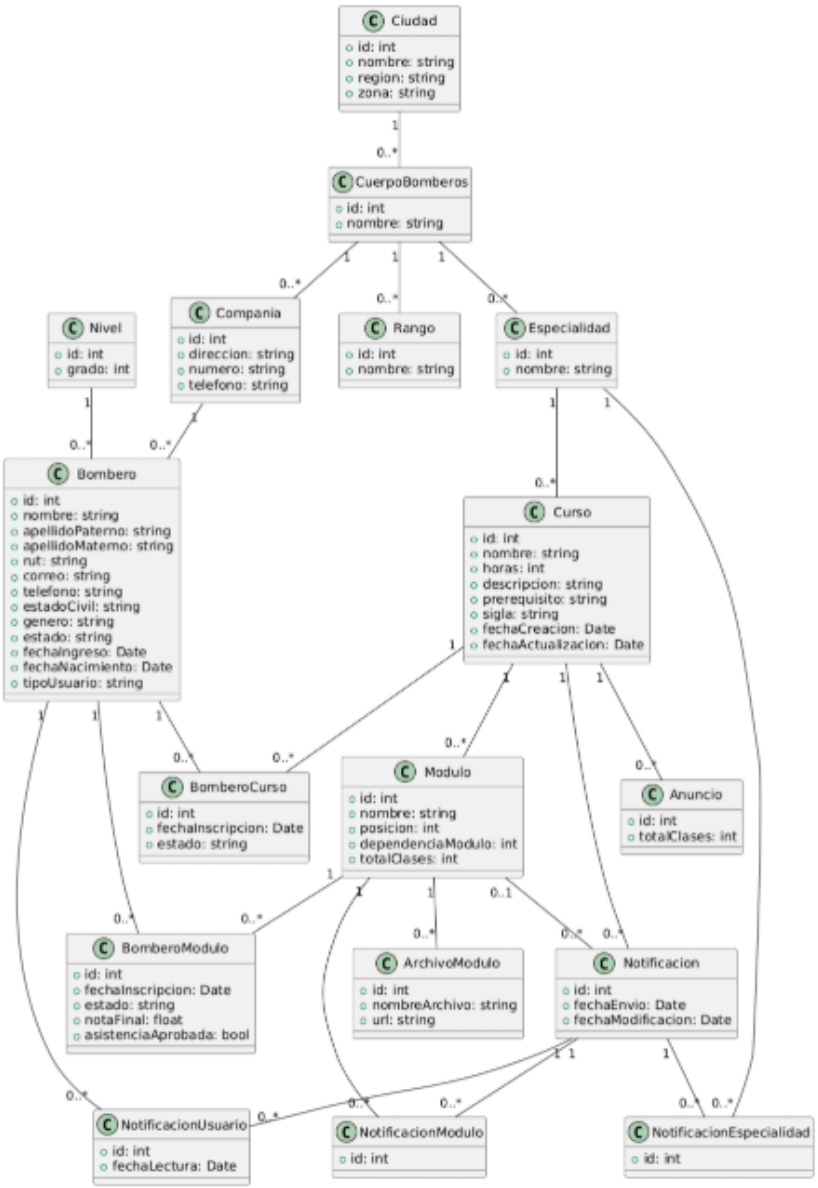


Figura 3-6. Diagrama de clases del sistema.
Fuente: Elaboración propia.

Los diagramas presentados permiten comprender de manera integral la estructura interna del sistema y la organización de la información dentro de la plataforma. El modelo relacional expone las principales entidades del dominio —usuarios, cursos, módulos, certificados y anuncios— junto con sus relaciones, facilitando la comprensión de cómo se almacena, vincula y asegura la integridad de los datos en la base de datos.
Por su parte, el diagrama de clases representa la arquitectura lógica del sistema desde una perspectiva orientada a objetos, mostrando cómo interactúan los distintos componentes para implementar los procesos académicos y administrativos definidos en el backend.

En conjunto, ambos modelos entregan una visión complementaria del funcionamiento interno del sistema: mientras el modelo relacional se centra en la estructura y organización de los datos, el diagrama de clases describe el comportamiento y las

interacciones entre las entidades. Esta visión integrada permite comprender la base conceptual sobre la cual se construye el backend y cómo cada parte se articula para dar respuesta a los requerimientos del proyecto.

3.1.1.2 Módulo de autenticación y control de acceso.

El módulo de autenticación y control de acceso constituye uno de los componentes centrales del sistema, ya que garantiza que cada usuario interactúe únicamente con las funciones que le corresponden según su rol dentro de la institución. Para este proyecto, los perfiles principales considerados son:

- Bombero (estudiante)
- Instructor
- Administrador

Cada uno de estos roles posee diferentes niveles de acceso y responsabilidades dentro de la plataforma, lo que permite organizar las operaciones académicas de manera más segura y controlada. Por ejemplo, los instructores pueden gestionar módulos, asistencia y notas; mientras que los administradores tienen facultades para crear cursos, asignar roles y administrar usuarios. Los bomberos, por su parte, pueden visualizar su avance académico de los módulos, revisar certificados y recibir notificaciones.

El proceso de autenticación se realiza mediante un mecanismo basado en tokens, donde el usuario ingresa sus credenciales y el sistema genera un identificador temporal que incluye información sobre su rol. Este token es verificado en cada solicitud realizada al backend, permitiendo determinar los permisos del usuario antes de ejecutar cualquier acción. De esta forma, el sistema protege la información sensible y previene accesos no autorizados a funcionalidades críticas, como la gestión de evaluaciones, la edición de cursos o la administración de certificados.

A continuación, se presenta el diagrama de componentes asociado a este módulo, el cual ilustra el flujo general de autenticación, la verificación del rol del usuario y la interacción con los distintos servicios internos del backend.

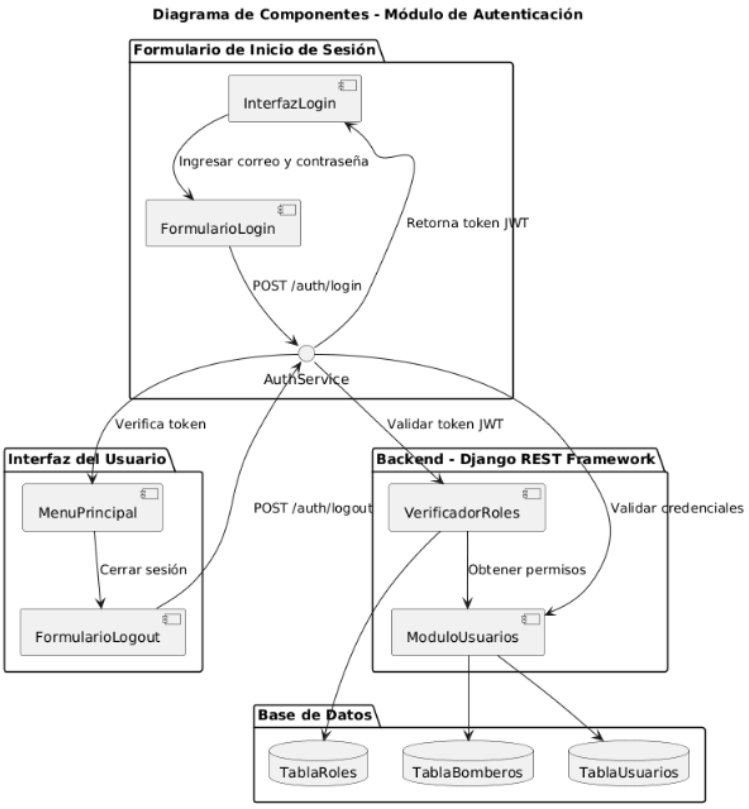


Figura 3-7. Diagrama de Componentes – Modulo de Autenticación.
Fuente: Elaboración propia.

El diagrama muestra el funcionamiento general del módulo de autenticación, donde el sistema utiliza un mecanismo único basado en credenciales locales (correo electrónico y contraseña). Cuando un usuario envía sus datos mediante el formulario de inicio de sesión, el backend valida las credenciales y genera un token JWT que incluye información esencial, como su identidad y rol dentro de la plataforma (bombero, instructor o administrador).

Este token se envía de vuelta al cliente y es utilizado en cada solicitud posterior, permitiendo al backend verificar permisos y controlar el acceso a los diferentes recursos del sistema. De este modo, el sistema garantiza que solo usuarios autenticados puedan interactuar con la plataforma y que cada uno visualice únicamente las funcionalidades que le corresponden según su rol. Este mecanismo centralizado de control de acceso contribuye a la seguridad del backend y a la correcta gestión de la información académica.

3.1.1.3 Módulo de gestión de usuarios.

El módulo de gestión de usuarios administra los perfiles de bomberos, instructores y administradores, garantizando que la información personal, académica y operacional de cada miembro se mantenga actualizada y correctamente vinculada a su rol dentro de la institución. Este módulo constituye uno de los pilares del sistema, ya que controla los accesos y organiza la participación de los usuarios en las distintas funciones de la plataforma.

- **Creación de usuarios**

Desde el frontend, los administradores completan un formulario de registro con los datos esenciales del usuario (correo, nombre, RUT, compañía, rol, entre otros). Esta información se envía mediante una solicitud POST al backend, donde se validan los datos, se crea la cuenta en el sistema de autenticación y se genera el perfil correspondiente (Bombero, Instructor o Administrador), asociando el rol que definirá sus permisos dentro de la plataforma.

- **Edición y actualización de datos**

A través de formularios de edición, los usuarios autorizados pueden modificar información del perfil, como teléfonos de contacto, dirección, compañía asignada o estado del usuario. Estas operaciones se realizan mediante solicitudes PUT/PATCH a endpoints de actualización (por ejemplo, `/api/usuarios/{id}/`), permitiendo corregir errores o mantener actualizados los datos relevantes para la gestión académica.

- **Eliminación de usuarios**

El sistema permite eliminar usuarios de forma directa, removiendo sus registros desde la administración de cuentas. La operación se ejecuta mediante una solicitud *DELETE* `(/api/usuarios/{id}/)`, eliminando completamente al usuario del sistema y restringiendo inmediatamente su acceso a la plataforma.

- **Administración de roles y perfiles**

Los administradores pueden modificar el rol de un usuario y ajustar atributos propios de su función dentro de la institución. Estos cambios impactan directamente en el control de acceso, permitiendo que el sistema muestre solo las funcionalidades habilitadas para cada perfil.

En conjunto, este módulo garantiza una administración ordenada, segura y escalable de los usuarios, asegurando consistencia en la información institucional y control adecuado sobre los permisos y acciones disponibles dentro de la plataforma.

3.1.1.4 Módulo de Cursos y módulos.

El módulo de gestión de cursos y módulos organiza y administra la estructura académica de la plataforma, permitiendo definir contenidos, estructurar instancias formativas y realizar un seguimiento completo del proceso educativo de los bomberos. Este módulo opera como el núcleo de la trazabilidad académica del sistema.

• Gestión de cursos

Los cursos representan instancias formales de formación. La plataforma permite: Mediante formularios específicos en la interfaz, los usuarios autorizados pueden:

- **Crear cursos**: definiendo nombre, descripción, nivel, duración estimada en horas, cupos y posibles prerrequisitos. Esta información se envía al backend mediante solicitudes *POST* a un endpoint de creación de cursos, donde se valida y se registra en la base de datos.

- **Editar cursos**: permitiendo actualizar datos como la descripción, el nivel, la duración, los cupos o el curso prerrequisito. Estas operaciones se realizan con solicitudes *PUT/PATCH* sobre un endpoint de actualización, manteniendo la información académica alineada con las necesidades formativas de la institución.

- **Eliminar cursos**: cuando un curso deja de ser impartido o debe retirarse del catálogo, los usuarios con permisos adecuados pueden solicitar su eliminación mediante operaciones *DELETE*. El backend valida que no existan dependencias críticas (como módulos activos o inscripciones vigentes) antes de completar la operación.

Con esto, el módulo asegura que el catálogo de cursos se mantenga consistente, actualizado y alineado con la oferta formativa de Bomberos.

• Gestión de módulos y clases
Cada curso puede descomponerse en uno o más módulos, que representan unidades temáticas o etapas del proceso formativo. El sistema permite:

- **Crear módulos** asociados a un curso, definiendo su nombre, posición dentro de la secuencia formativa, posibles dependencias entre módulos y cantidad de horas que lo componen.

- **Editar módulos**, ajustando su nombre, posición, dependencia o estado (por ejemplo, en progreso o completado) según el avance real del curso.

- **Eliminar módulos** cuando ya no forman parte de la estructura académica o han sido reemplazados por una nueva versión.

Esto permite planificar de manera ordenada la ejecución del curso y registrar la información necesaria para el seguimiento académico.

• Inscripción de bomberos y seguimiento académico
La plataforma registra la participación de los bomberos en cursos y módulos mediante entidades especializadas que permiten:

- **Inscripción a cursos**: permite asociar a un bombero con un curso determinado, registrando la fecha de inscripción y el estado de participación (iniciado, completado, cancelado).

- **Inscripción a módulos**: a partir de la inscripción a un curso, el sistema permite vincular al bombero con los módulos que lo componen, controlando requisitos como la inscripción previa al curso padre y el cumplimiento de secuencias formativas.

- **Registro de notas y asistencia**: para cada módulo se puede almacenar información de rendimiento, como nota final y cumplimiento de asistencia mínima.

- **Cierre de módulos y cursos**: una vez finalizadas las clases y evaluaciones, el sistema permite marcar módulos y cursos como completados, consolidando los resultados académicos y habilitando la generación o el registro de certificados en el módulo correspondiente.

Este módulo constituye un sistema completo de seguimiento académico que garantiza trazabilidad, control y consistencia en la formación institucional.

3.1.1.5 Módulo de anuncios y notificaciones.

El módulo de anuncios y notificaciones permite gestionar la comunicación interna de la plataforma académica. Los anuncios son mensajes generales publicados por administradores y dirigidos a todos los usuarios, utilizados para informar sobre aperturas de cursos, actualizaciones académicas o comunicados institucionales.

Las notificaciones, en cambio, poseen un carácter más específico y suelen ser emitidas por instructores. Estas se asocian a módulos o especialidades y permiten comunicar recordatorios, cambios de horario o indicaciones académicas puntuales. El sistema registra además el estado de lectura para cada usuario.

El módulo se estructura en una capa de presentación con vistas diferenciadas, una capa de lógica que gestiona la distribución de mensajes y una capa de datos que almacena anuncios, notificaciones y sus destinatarios, garantizando trazabilidad y control de la comunicación interna.

3.1.1.6 Módulo de certificados.

El módulo de certificados permite gestionar y almacenar de manera segura los documentos académicos emitidos a cada bombero. Su función principal es mantener un repositorio digital actualizado, donde los administradores pueden cargar certificados asociados a cursos, vinculándolos directamente al perfil del usuario. Cada registro incluye información como tipo de certificación, fecha de obtención y archivo respaldado.

Los bomberos pueden acceder a su historial de certificaciones, visualizar sus logros y descargar los documentos cuando lo requieran, facilitando el seguimiento de su progreso formativo.

El módulo se organiza en una arquitectura por capas: la capa de presentación gestiona la carga y consulta de certificados; la lógica de negocio valida y administra los documentos; y la capa de datos mantiene la persistencia estructurada. Esto garantiza trazabilidad, confiabilidad y un manejo eficiente de la información académica.

3.1.1.7 Módulo de Dashboard Académico.

El módulo de dashboard proporciona a administradores e instructores una vista consolidada de los principales indicadores académicos del sistema. Su propósito es ofrecer métricas actualizadas que faciliten el monitoreo institucional y la toma de decisiones.

El dashboard para administradores presenta información global, como cantidad de bomberos inscritos, cursos activos, módulos en ejecución, distribución por compañías y especialidades, además de estadísticas sobre participación y avance académico.
En el caso de los instructores, el dashboard se enfoca en métricas pedagógicas específicas de los cursos y módulos asignados, incluyendo número de estudiantes inscritos por módulo, cursos activos, notificaciones emitidas y registros de participación.

En conjunto, este módulo entrega una visión clara y actualizada del estado académico de la institución, fortaleciendo la supervisión administrativa y docente.

3.1.1.8 Módulo de Inscripciones a Cursos y Módulos

El módulo de inscripciones administra el proceso mediante el cual los bomberos se registran en cursos y módulos, asegurando que cada asignación cumpla con los criterios

académicos definidos por la institución. Para ello, valida cupos disponibles, prerrequisitos, estado del alumno y coherencia con la estructura pedagógica del curso. Una vez confirmada la inscripción, el sistema asocia automáticamente al usuario con los módulos correspondientes, respetando dependencias y orden de progreso.

Asimismo, permite actualizar el estado académico de cada inscripción (iniciada, completada o cancelada), manteniendo una trazabilidad clara del recorrido formativo. Esta información se integra posteriormente con los procesos de evaluación y asistencia, permitiendo un seguimiento completo del desempeño académico.

Este módulo es esencial para garantizar una progresión educativa ordenada y consistente dentro de la plataforma.

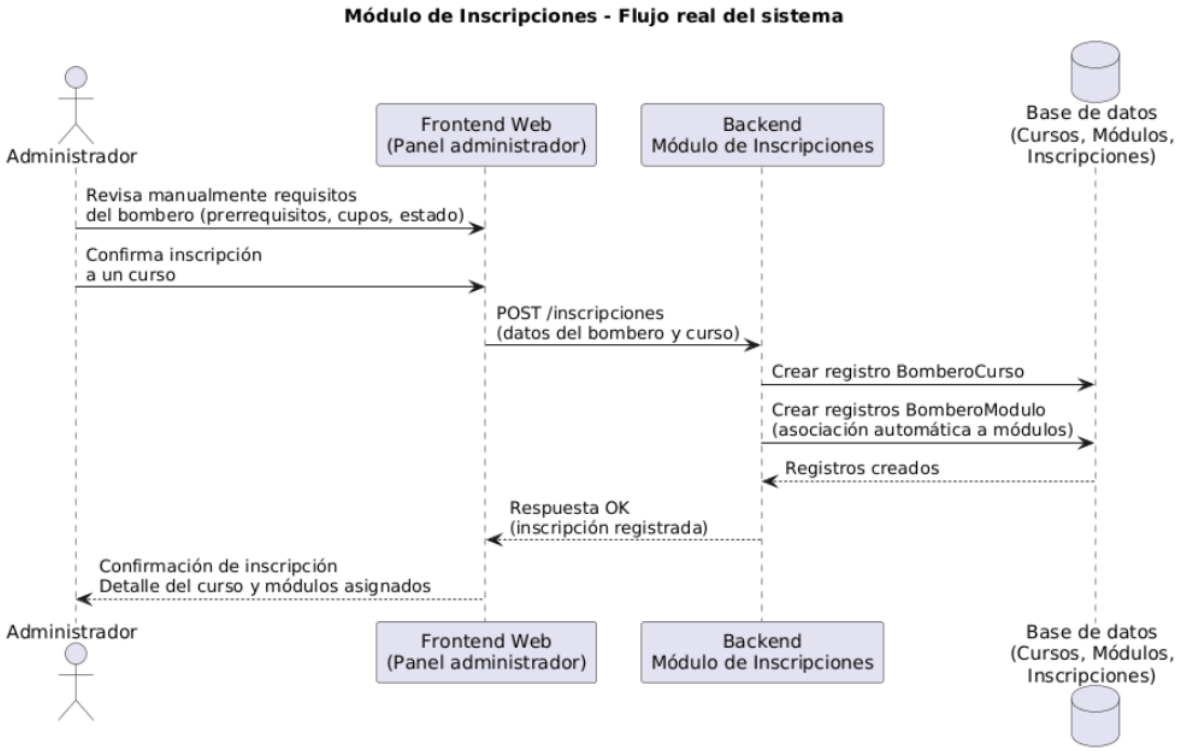


Figura 3-8. Diagrama de Flujo Inscripciones Cursos y Módulos.
Fuente: Elaboración propia.

El diagrama anterior ilustra el flujo real del proceso de inscripción, mostrando la interacción entre el administrador, la interfaz web y el backend. Desde el panel administrativo se envía una solicitud al servicio de inscripciones, el cual valida prerrequisitos, cupos y estado académico del bombero utilizando los módulos de cursos, usuarios y módulos. Una vez verificados estos criterios, el backend crea los registros correspondientes en la base de datos, asociando al bombero con el curso y con los módulos que lo componen. Este flujo asegura consistencia en las reglas académicas, manteniendo trazabilidad y un control centralizado del avance formativo dentro de la plataforma.

3.1.1.9 Módulo de Archivos Académicos.

El módulo de archivos académicos permite a los instructores administrar y publicar material educativo asociado a cada módulo de formación, centralizando recursos como guías, presentaciones y evaluaciones. El backend valida el tipo y tamaño de los archivos y los almacena de forma segura, manteniendo su asociación con el módulo correspondiente. Los estudiantes solo pueden acceder a los documentos de los módulos en los que están inscritos, garantizando un control adecuado de permisos y la protección de contenido sensible. Este módulo aporta organización, trazabilidad y un repositorio académico seguro para apoyar el proceso formativo.

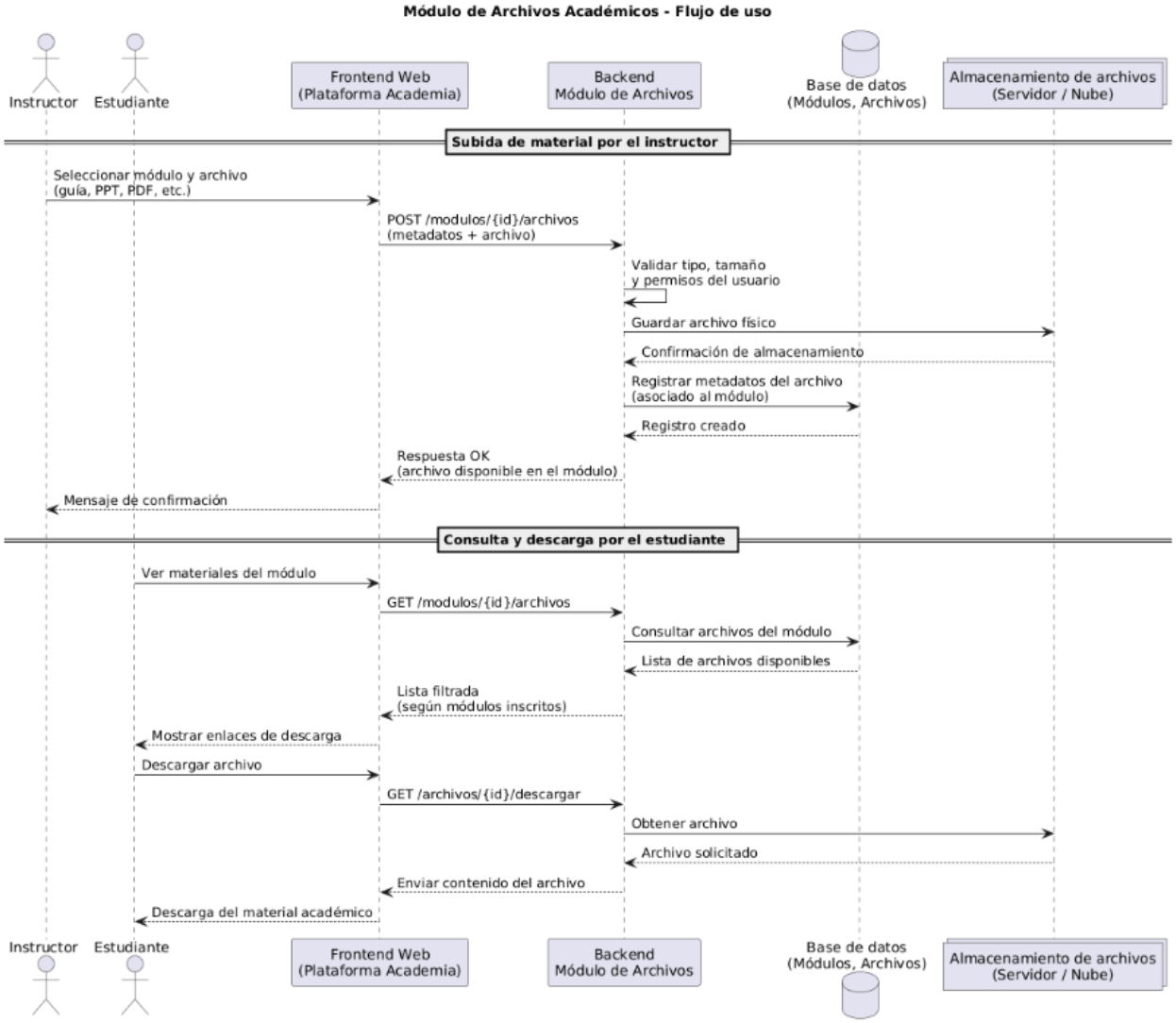


Figura 3-9. Diagrama de Flujo Archivos Académicos.
Fuente: Elaboración propia.

3.1.1.10 Módulo de Seguridad y Control de Acceso.

El módulo de seguridad constituye la base operativa del sistema, garantizando la protección de los datos y el acceso controlado a las funcionalidades. La plataforma utiliza autenticación mediante tokens JWT, permitiendo validar la identidad del usuario en cada solicitud y aplicar permisos según su rol.

El backend incorpora validaciones previas a cualquier operación sensible, asegurando que solo usuarios autorizados administradores, instructores o bomberos puedan ejecutar acciones como gestión de usuarios, edición de cursos o publicación de anuncios. Asimismo, se implementan medidas de protección frente a vulnerabilidades comunes, como inyección SQL o accesos indebidos a información restringida.

Este módulo integra servicios transversales del sistema, como administración de sesiones, refresco de tokens y manejo consistente de errores, permitiendo que toda la plataforma opere bajo estándares adecuados de seguridad y confiabilidad.

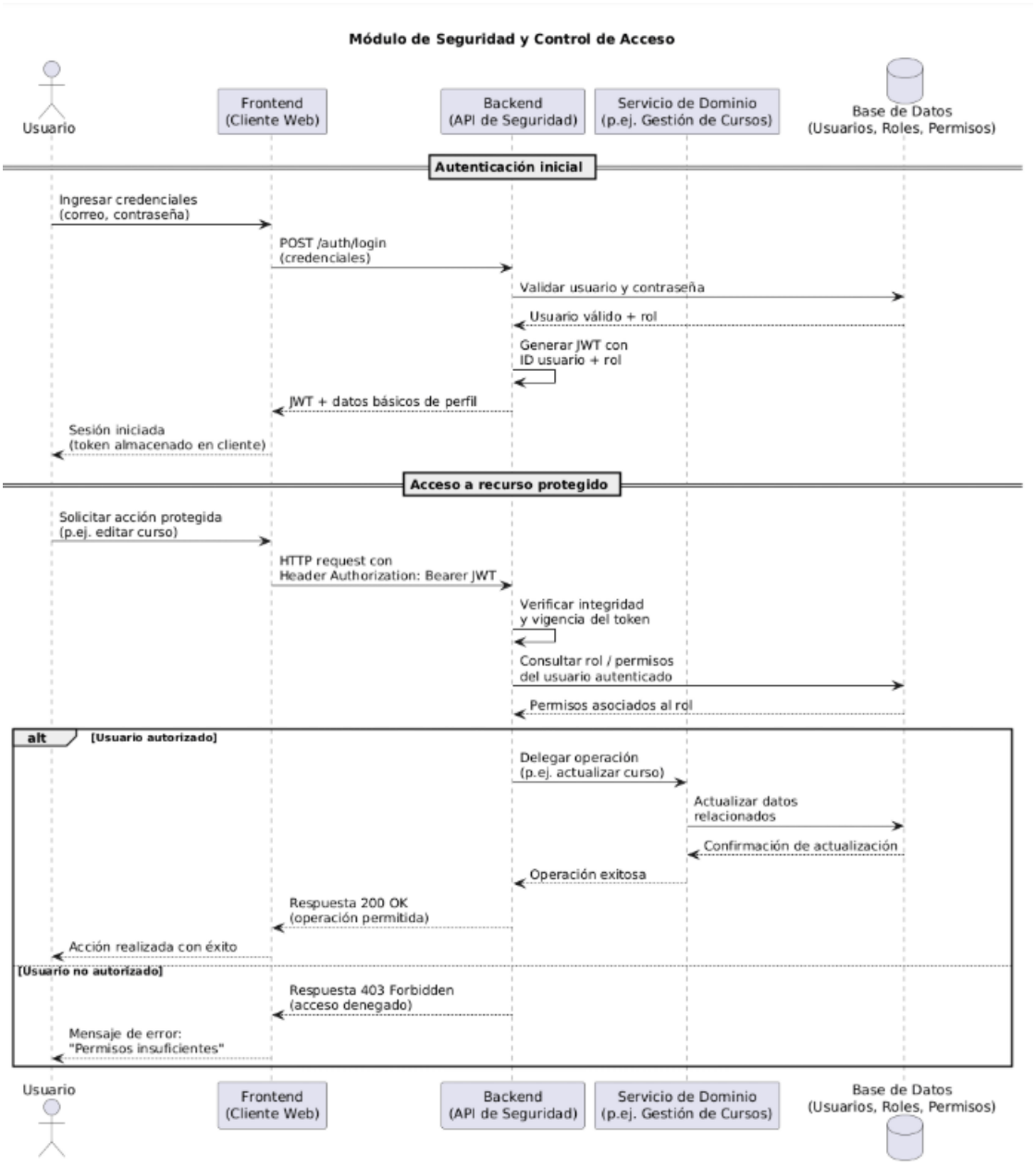


Figura 3-10. Diagrama de Flujo Seguridad y Control de acceso.
Fuente: Elaboración propia.

La figura muestra el flujo del módulo de seguridad y control de acceso. Primero, el usuario envía sus credenciales al backend, donde se validan y se genera un token JWT que contiene su identidad y rol. Posteriormente, en cada operación protegida, el cliente incluye este token en la cabecera de la solicitud. El backend verifica su validez, revisa los permisos del rol y, según corresponda, permite la ejecución del servicio solicitado o rechaza la acción por falta de autorización. Este proceso asegura que todas las operaciones sensibles estén protegidas mediante un mecanismo centralizado de autenticación y control de acceso.

3.1.1.11 Estrategia de Implementación.

La implementación del backend de la plataforma académica para Bomberos de Chile se realizó mediante una estrategia iterativa, modular y basada en APIs REST, orientada a asegurar que cada componente tecnológico del sistema fuera desarrollado, probado e integrado de forma progresiva.

Este enfoque permitió mantener la consistencia del código, garantizar el cumplimiento de los requerimientos funcionales y facilitar la integración con el frontend desarrollado en Angular, asegurando una arquitectura escalable, segura y alineada con las necesidades institucionales.

3.1.1.12 Desarrollo Iterativo.

El proyecto se construyó de manera incremental, validando cada módulo antes de su integración. Esto permitió detectar errores tempranamente, asegurar trazabilidad y adaptar el sistema a necesidades emergentes.

1) Planeación
En esta fase se levantaron las necesidades funcionales de la plataforma según los actores involucrados: administradores, instructores y bomberos estudiantes. A partir de ello se identificaron los módulos críticos: gestión de usuarios y roles, administración académica, inscripciones, certificados, notificaciones y panel de análisis administrativo.

2) Requerimientos
Se definieron los requisitos técnicos y funcionales de cada módulo, considerando aspectos como:

- Restricciones de acceso por rol.
- Flujos académicos: cursos, módulos, evaluaciones y certificaciones.
- Reglas de validación de datos.
- Interoperabilidad entre componentes.

3) Diseño
Se estableció la arquitectura en capas del backend y se diseñaron:

- Modelos y entidades de la base de datos.
- Relaciones entre tablas.
- Endpoints REST organizados.
- Diagramas UML necesarios para representar la estructura lógica y operativa del sistema.

4) Implementación
Cada componente fue desarrollado utilizando Django REST Framework, aplicando prácticas de modularidad, separación de capas y reutilización de código. Los módulos se

implementaron de manera independiente, permitiendo su incorporación progresiva según las prioridades establecidas.

5) Pruebas
En cada iteración se llevaron a cabo:

- Pruebas unitarias de modelos, lógica interna y validaciones.
- Pruebas de integración mediante herramientas como Postman para validar la interacción completa de las API.
- Pruebas funcionales realizadas desde el frontend Angular para verificar el comportamiento real del sistema.

6) Evaluación
Finalizada cada iteración, se analizaron avances, ajustes necesarios y nuevos requerimientos surgidos durante el desarrollo, permitiendo mejorar continuamente la calidad del sistema.

Este método permitió construir un backend coherente, estable y adaptable. Al validar cada módulo antes de integrarlo, se redujeron riesgos de incompatibilidades, errores acumulados o fallos en cascada. En consecuencia, el sistema final obtuvo mayor solidez, confiabilidad y alineación con las necesidades operativas de la plataforma académica.

3.1.1.13 Uso de API RESTs.

El backend fue diseñado mediante un enfoque completamente desacoplado, basando su comunicación con el frontend Angular en APIs RESTful, lo que permite escalabilidad, interoperabilidad y facilidad de mantenimiento.

Principios REST aplicados:
La implementación siguió los siguientes pilares técnicos:

- Cliente-servidor desacoplado: frontend y backend operan y evolucionan de manera independiente.
- Comunicación sin estado: cada solicitud contiene la información necesaria para ser procesada.
- Transferencia en formato JSON: liviano, estándar y ampliamente compatible.
- Uso correcto de métodos HTTP: GET, POST, PUT y DELETE con semántica específica para cada operación.

Organización de endpoints:
Los endpoints se estructuraron en grupos según los módulos principales del sistema:

- Autenticación y roles: Inicio de sesión, actualización de token, permisos.
- Usuarios: Creación, edición, eliminación, detalles por rol.
- Cursos y módulos académicos: Creación, edición, consulta detallada, vinculación entre módulos y cursos.
- Inscripciones: Registro de participantes, revisión de inscritos, actualización de estado.
- Certificados: Carga de certificados, validación, consulta por usuario.
- Notificaciones y anuncios: Publicación y lectura de notificaciones internas.
- Dashboard académico: Métricas generales del sistema, estadísticas y reportes.

Seguridad:
La capa de seguridad incluye:

- Autenticación basada en tokens JWT.
- Permisos por rol en cada vista.
- Validación estricta de datos de entrada.
- Control de errores y respuestas estructuradas siguiendo códigos HTTP estándar.

Pruebas de API
Las API fueron validadas utilizando:

- Pruebas de integración (Postman).
- Pruebas funcionales desde el frontend.
- Secuencias de prueba para rutas críticas: cursos, módulos, inscripción y login.

El uso de APIs REST permitió construir un backend modular y mantenible, donde cada componente cumple un rol claramente definido y puede evolucionar sin afectar al resto del sistema. Este desacoplamiento facilita la reutilización de lógica en distintos contextos, habilita futuras integraciones —como una aplicación móvil o servicios institucionales externos— y garantiza que el crecimiento del sistema pueda realizarse de manera ordenada, escalable y segura.

3.1.1.14 Despliegue y Entorno.

El entorno de despliegue se concibió bajo principios de escalabilidad, consistencia y facilidad de mantenimiento, utilizando contenedores y servicios de infraestructura en la nube.

Contenedores Docker:
El backend y la base de datos fueron empaquetados mediante contenedores Docker, lo cual permite:

- Asegurar entornos homogéneos entre desarrollo y producción.
- Reducir errores por diferencias de versiones.
- Facilitar la migración y despliegue del sistema.

Infraestructura en la nube:
La arquitectura está diseñada para ejecutarse en una instancia de servidor virtual ofreciendo:

- Escalabilidad de recursos bajo demanda.
- Mayor estabilidad ante cargas variables.
- Flexibilidad para incorporar nuevos servicios complementarios.

Base de datos:
El proyecto emplea dos configuraciones según el entorno:

- SQLite3 para desarrollo local (simple y sin necesidad de configuración adicional).
- PostgreSQL para producción, garantizando transacciones seguras y rendimiento en consultas concurrentes.

Almacenamiento de archivos:
El almacenamiento de documentos y certificados está diseñado para utilizar un servicio dedicado de almacenamiento externo, lo que permite:

- Alta disponibilidad de los archivos.
- Seguridad mediante claves y políticas de acceso.
- Escalabilidad sin depender del servidor principal.

El diseño del entorno permite operar con consistencia y estabilidad en todas las etapas del proyecto, evitando discrepancias entre ambientes y facilitando el mantenimiento. Además, deja preparado el sistema para incorporar en el futuro pipelines de integración y despliegue continuo, asegurando una evolución ordenada y sostenible sin necesidad de rediseños mayores.

3.1.2 Uso de buenas prácticas y patrones de diseño.

En el desarrollo del backend de la plataforma académica se aplicaron patrones de diseño y buenas prácticas orientadas a mantener un sistema modular, escalable y fácil de mantener. Entre los lineamientos principales destacan el uso del patrón MVC y la adopción del estilo arquitectónico REST, además de prácticas generales de desarrollo seguro.

3.1.2.1 Patrones de Diseño aplicados.

Patrón MVC (Model–View–Controller)
El backend se organizó siguiendo el patrón MVC, adaptado al uso de Django como API:

- **Modelos:** definen la estructura de datos y reglas de negocio (usuarios, roles, cursos, módulos, inscripciones, certificados).
- **Vistas:** Encargadas de retornar las respuestas en formato JSON consumidas por el frontend, recibiendo y procesando solicitudes HTTP.
- **Controladores:** Administran la lógica que coordina la interacción entre las vistas y los modelos, interpretando solicitudes, validando datos y ejecutando operaciones sobre la base de datos.

Aunque Django utiliza el patrón MVT, al prescindir de plantillas y operar como API, su comportamiento es equivalente a MVC en términos de separación de responsabilidades.

Razones para adoptar MVC en este proyecto

- **Frontend independiente:** Angular opera como cliente externo, sin depender del motor de plantillas de Django.
- **Modularidad y claridad arquitectónica:** La separación entre modelos, vistas y controladores facilita el mantenimiento, la organización del código y la comprensión del flujo interno del sistema.
- **Escalabilidad:** Permite que el backend evolucione sin afectar al frontend, y viceversa, promoviendo una arquitectura flexible y preparada para futuras ampliaciones.

3.1.2.2 Estilo Arquitectónico REST.

La comunicación entre el backend y el frontend se implementó utilizando el estilo arquitectónico REST, lo cual aporta eficiencia y estandarización a las interacciones del sistema.

Principios REST aplicados:

La adopción del estilo arquitectónico REST permite establecer una comunicación clara y estandarizada entre el backend y el frontend, manteniendo una separación total entre cliente y servidor. Al operar bajo un enfoque *stateless*, cada solicitud incluye toda la información necesaria para ser procesada, evitando dependencias entre peticiones y favoreciendo la escalabilidad. Asimismo, el uso apropiado de los métodos HTTP —GET para consultas, POST para creación, PUT para actualizaciones y DELETE para eliminaciones— junto con el empleo de JSON como formato principal de intercambio, garantiza interoperabilidad con cualquier cliente externo. Este enfoque facilita la integración futura de nuevas interfaces, simplifica las tareas de prueba y mantenimiento y proporciona una capa de comunicación bien definida y documentada entre los componentes del sistema.

Ventajas técnicas para la plataforma

- Facilita la escalabilidad del backend.
- Permite la integración de otros clientes futuros (app móvil, panel institucional).
- Simplifica las pruebas, mantenimiento y depuración.
- Establece una capa clara y documentada de comunicación entre sistemas.

3.1.2.3 Buenas prácticas de desarrollo.

Estructura modular del código

El backend está organizado en módulos independientes para usuarios, roles, cursos, módulos académicos, inscripciones, certificados, anuncios y dashboard. Esto permite trabajar en cada componente sin afectar otros y favorece la mantenibilidad.

Control de versiones con Git

Se utilizó un flujo de trabajo basado en ramas específicas para funcionalidades, manteniendo:

- Historial limpio y trazable
- Separación entre desarrollo y versiones estables
- Integración controlada mediante revisiones de código

Pruebas unitarias e integración

Se implementaron pruebas para validar:

- Reglas de negocio
- Autenticación y manejo de permisos
- Endpoints críticos del sistema

El uso de herramientas como Postman permitió verificar la correcta interacción del frontend con las API.

Gestión de errores

- Manejo explícito de excepciones mediante bloques try-except.
- Respuestas JSON estandarizadas indicando códigos de estado HTTP.
- Registro de errores para facilitar su análisis y corrección.

Buenas prácticas de seguridad

- Cifrado de contraseñas mediante algoritmos seguros incluidos en Django.
- Validación y sanitización exhaustiva de entradas del usuario.
- Uso de tokens JWT para proteger rutas sensibles.
- Restricciones de acceso basadas en roles.
- Configuración de protección frente a ataques comunes (inyecciones SQL, XSS, CSRF para operaciones web).

Automatización y despliegue

- Uso de Docker para asegurar que el backend funcione de manera idéntica en todos los entornos.
- Preparación para CI/CD, facilitando la automatización de pruebas y despliegues futuros.
- Estructura de contenedores que permite ampliar o migrar el sistema sin afectar la arquitectura interna.

3.1.2.4 Integración del sistema en General.

La integración del backend con el resto del sistema —principalmente con el frontend desarrollado en Angular— se realiza mediante solicitudes HTTP y respuestas en formato JSON, lo que permite una comunicación clara, estandarizada y completamente desacoplada entre los componentes. El backend actúa como capa lógica central, exponiendo un conjunto de APIs REST que encapsulan la lógica de negocio, gestionan las operaciones con la base de datos y controlan los permisos según los roles definidos, mientras que el frontend consume estas API para mostrar información, ejecutar acciones académicas y mantener sincronizado el estado de la plataforma.

Método de integración.

APIs REST como mecanismo principal de interacción

Cada funcionalidad relevante del sistema cuenta con uno o más endpoints diseñados específicamente para cubrir su flujo operativo. Las respuestas se entregan exclusivamente en formato JSON, lo que garantiza compatibilidad con distintos tipos de clientes y facilita tanto el consumo como el procesamiento de la información por parte del frontend.

Mensajes intercambiados entre frontend y backend

El frontend envía solicitudes HTTP utilizando los métodos adecuados según la operación requerida:

- **GET:** consultas de información (usuarios, cursos, módulos, inscripciones, certificados).
- **POST:** creación de registros (inscripciones, cursos, usuarios, certificados).
- **PUT:** actualización de información (roles, estados, datos académicos).
- **DELETE:** eliminación de registros en módulos específicos.

Cada respuesta emitida por el backend incluye:

- Código de estado HTTP (200, 201, 400, 401, 404, 500), indicando el resultado de la operación.
- Mensaje descriptivo señalando éxito o error.
- Datos devueltos cuando corresponde (información de usuario, detalles de cursos, inscripciones activas, documentos asociados, etc.).

Seguridad en la comunicación

Para proteger el acceso a las funcionalidades críticas, se aplican varias medidas de seguridad:

- Todas las solicitudes a rutas protegidas deben incluir un token JWT válido en los encabezados:
  `Authorization: Bearer <token>`

- El backend valida el rol del usuario antes de ejecutar cualquier acción (administrador, instructor o bombero estudiante).

- La comunicación entre frontend y servidor está pensada para ejecutarse mediante HTTPS, asegurando la integridad y confidencialidad de los datos transmitidos.

3.1.2.5 Documentación de los EndPoints principales.

A continuación, se presenta una síntesis de los endpoints más relevantes de la plataforma, organizados según los módulos funcionales del sistema. La versión detallada y completa se incluye en los anexos correspondientes.

Módulo de Usuarios

Descripción	**Endpoint**	**Método**	**Parámetros**	**¿Requiere Administrador?**
Iniciar sesión	`/api/user/login_access/`	POST	email, contraseña	No
Crear usuario	`/api/user/bomberos`	POST	nombre, email, rol, contraseña	Sí
Actualizar usuario	`/api/user/bomberos/{id}/`	PUT	datos por actualizar	Sí
Obtener usuarios por rol	`/api/user/bomberos/rangos/{id}`	GET	n/a	Sí

Tabla 3.1 – Endpoints del Módulo de Usuarios.
Fuente: Elaboración propia.

Módulo Académico (Cursos y Módulos)

Descripción	Endpoint	Método	Parámetros	¿Requiere Administrador?
Crear curso	`/api/cursos/`	POST	nombre, descripción, instructor	Sí
Obtener detalle de curso	`/api/cursos/{id}/`	GET	id de curso	No
Crear módulo académico	`/api/modulos/`	POST	nombre, contenido, id_curso	Sí
Editar módulo	`/api/modulos/{id}/`	PUT	campos por modificar	Sí

Tabla 3.2 – Endpoint del Módulo Académico.
Fuente: Elaboración propia.

En conjunto, esta estrategia de integración permite que el backend funcione como un componente centralizado, seguro y desacoplado, capaz de proveer información de manera consistente al frontend y otros posibles clientes. Gracias al uso de APIs REST bien definidas, un intercambio claro mediante JSON y mecanismos de seguridad basados en tokens, el sistema garantiza una operación estable, escalable y alineada con las necesidades académicas de la institución.

3.2 Detalles de la codificación y desarrollo.

La codificación y el desarrollo del backend se llevaron a cabo siguiendo una planificación basada en metodologías ágiles, priorizando la modularidad, la escalabilidad y la claridad estructural. El proyecto utilizó Scrum como marco de trabajo, dividiendo el desarrollo en sprints iterativos con entregables definidos, considerando la complejidad del sistema académico y la necesidad de mantener un flujo de trabajo ordenado y adaptable.
El backend fue implementado utilizando Django REST Framework, complementado con herramientas de integración continua, contenedores Docker y un control de versiones estructurado en ramas, lo que permitió asegurar consistencia entre entornos y calidad en las funcionalidades entregadas.

3.2.1 Metodología de Desarrollo.

El desarrollo se organizó en tres sprints principales, donde cada iteración incorporó entregables incrementales, pruebas y revisiones. Esta estrategia permitió asegurar que cada módulo del backend fuera desarrollado de forma completa antes de integrarse con el resto del sistema.

Sprint 1: Configuración base y diseño del sistema

Objetivos principales

- Definir la arquitectura del backend.
- Configurar el entorno de desarrollo.
- Establecer los cimientos técnicos del sistema académico.

Actividades técnicas realizadas

- **Diseño de arquitectura:**
    1) Definición del patrón MVC aplicado al backend para mantener una separación clara entre modelos, vistas y controladores.
    2) Identificación de los módulos esenciales: autenticación, usuarios, roles, cursos, módulos, inscripciones y certificados.

- **Configuración inicial del entorno:**

    1) Configuración del proyecto en Django REST Framework.
    2) Implementación inicial de la base de datos (SQLite3 para desarrollo).
    3) Preparación del entorno Docker para garantizar consistencia entre desarrollo y despliegue futuro.

- **Estructuración del repositorio:**
    1) Configuración de ramas main, dev y ramas específicas por funcionalidad.

Sprint 2: Desarrollo de funcionalidades principales

Objetivos principales

- Implementar los módulos funcionales base del backend.
- Construir las primeras APIs REST para su consumo desde el frontend.

Actividades técnicas realizadas

- **Modelamiento de datos:**
    1) Creación de modelos para usuarios, roles, cursos, módulos académicos e inscripciones.
    2) Definición de relaciones jerárquicas entre curso–módulo y alumno–inscripción.

- **Implementación de endpoints iniciales:**
    1) Autenticación con JWT.
    2) Registro y consulta de usuarios.
    3) Creación y consulta de cursos y módulos.
    4) Gestión de inscripciones y validaciones iniciales.

- **Inicio del sistema de notificaciones internas:**
    1) Endpoints básicos para creación y visualización de anuncios.

- **Pruebas iniciales:**
    1) Pruebas unitarias en Django.
    2) Pruebas de integración utilizadas desde Angular.

Sprint 3: Funcionalidades avanzadas, optimización y preparación para despliegue

Objetivos principales

- Completar funcionalidades avanzadas relacionadas con la gestión académica.
- Optimizar rendimiento y seguridad del backend.

- Preparar el sistema para despliegue en producción.

Actividades técnicas realizadas

- **Implementación de módulos avanzados:**
    1) Panel administrativo (dashboard) con métricas: cursos activos, inscripciones, progreso académico y estadísticas por módulo.
    2) Sistema de historial académico del bombero estudiante.
    3) Asignación de roles y gestión de permisos.
    4) Gestión de materiales y recursos didácticos.
    5) Carga y consulta de certificados.

- **Optimización del backend:**
    1) Validaciones estrictas en endpoints críticos.
    2) Manejo centralizado de errores.
    3) Optimización de consultas para disminuir tiempos de respuesta.

- **Pruebas finales:**
    1) Pruebas de estrés e integración con el frontend.
    2) Corrección de errores detectados en escenarios académicos reales.

- **Preparación para despliegue:**
    1) Contenedorización completa con Docker.
    2) Configuración para ejecutar en entorno de servidor (AWS o VPS).
    3) Ajustes finales para garantizar estabilidad en producción.

3.2.2 Historias de Usuario.

Las funcionalidades del backend se construyeron a partir de un conjunto de historias de usuario, organizadas por prioridad e iteraciones. Estas historias responden a las necesidades reales del sistema académico del Cuerpo de Bomberos, considerando perfiles como: estudiante, instructor, administrador académico y rector institucional.

Ejemplos representativos (de los más relevantes para el backend):

- Como usuario registrado, quiero iniciar sesión con mi correo y contraseña para acceder al sistema.
- Como instructor, quiero enviar notificaciones a mis estudiantes según curso o módulo.
- Como administrador, quiero asignar módulos y cursos dentro de una jerarquía estructurada.
- Como rector, quiero visualizar estadísticas del progreso académico.
- Como alumno, quiero ver mis módulos inscritos y mi avance dentro de la malla.

Estas historias guiaron el modelado de datos, los endpoints REST y la distribución del trabajo dentro de los sprints.

Casos de uso:

Los casos de uso permiten detallar el comportamiento esperado del sistema en escenarios clave. Para cada uno se desarrollaron diagramas de secuencia que muestran la interacción entre usuarios, backend, base de datos y servicios externos (por ejemplo, carga de certificados o notificaciones internas).

Casos de uso relevantes del proyecto incluyen:

- Inscripción académica en cursos y módulos.
- Revisión y descarga de certificados.
- Gestión administrativa de cursos.
- Visualización del progreso académico y métricas institucionales.
- Sistema de notificaciones internas.

Cada caso de uso está documentado con su flujo principal, excepciones, precondiciones y postcondiciones, lo que permitió diseñar la API de manera consistente y robusta.

Caso de uso 1: Inscribir bombero en módulo académico

CAMPO	DESCRIPCIÓN
ACTOR	Administrador académico
PROPÓSITO	Permitir al administrador académico inscribir a un bombero en un módulo específico de la malla curricular, con el fin de gestionar su avance académico y mantener un registro formal de las inscripciones realizadas.
FLUJO PRINCIPAL	1. El administrador académico ingresa a la plataforma web e inicia sesión. 2. El sistema valida sus credenciales y otorga acceso al panel administrativo. 3. El administrador accede a la sección de gestión de cursos y módulos. 4. Selecciona el curso y módulo donde desea inscribir al bombero. 5. Busca y selecciona al bombero desde el listado de usuarios disponibles. 6. Confirma la inscripción del bombero en el módulo seleccionado. 7. El sistema valida que exista disponibilidad y que el bombero no esté inscrito previamente. 8. El sistema registra la inscripción en la base de datos. 9. El sistema muestra un mensaje de confirmación.
EXCEPCIONES	2ª. Si las credenciales son incorrectas, el sistema rechaza el inicio de sesión. 7ª. Si el bombero ya está inscrito en el módulo, el sistema informa que no es posible duplicar la inscripción. 7ᵇ. Si el módulo está cerrado o inactivo, se informa que no puede inscribirse. 8ª. Si ocurre un error en el registro, se cancela la operación y se informa al usuario.
PRECONDICIONES	• El administrador académico debe tener una cuenta activa con permisos. • El bombero debe existir como usuario registrado. • Deben existir cursos y módulos configurados. • La conexión a internet debe estar activa.
POSTCONDICIONES	• La inscripción queda registrada en la base de datos. • El módulo aparece asociado al bombero en su historial académico. • La información de inscripciones se actualiza para reportes, estadísticas y paneles administrativos.

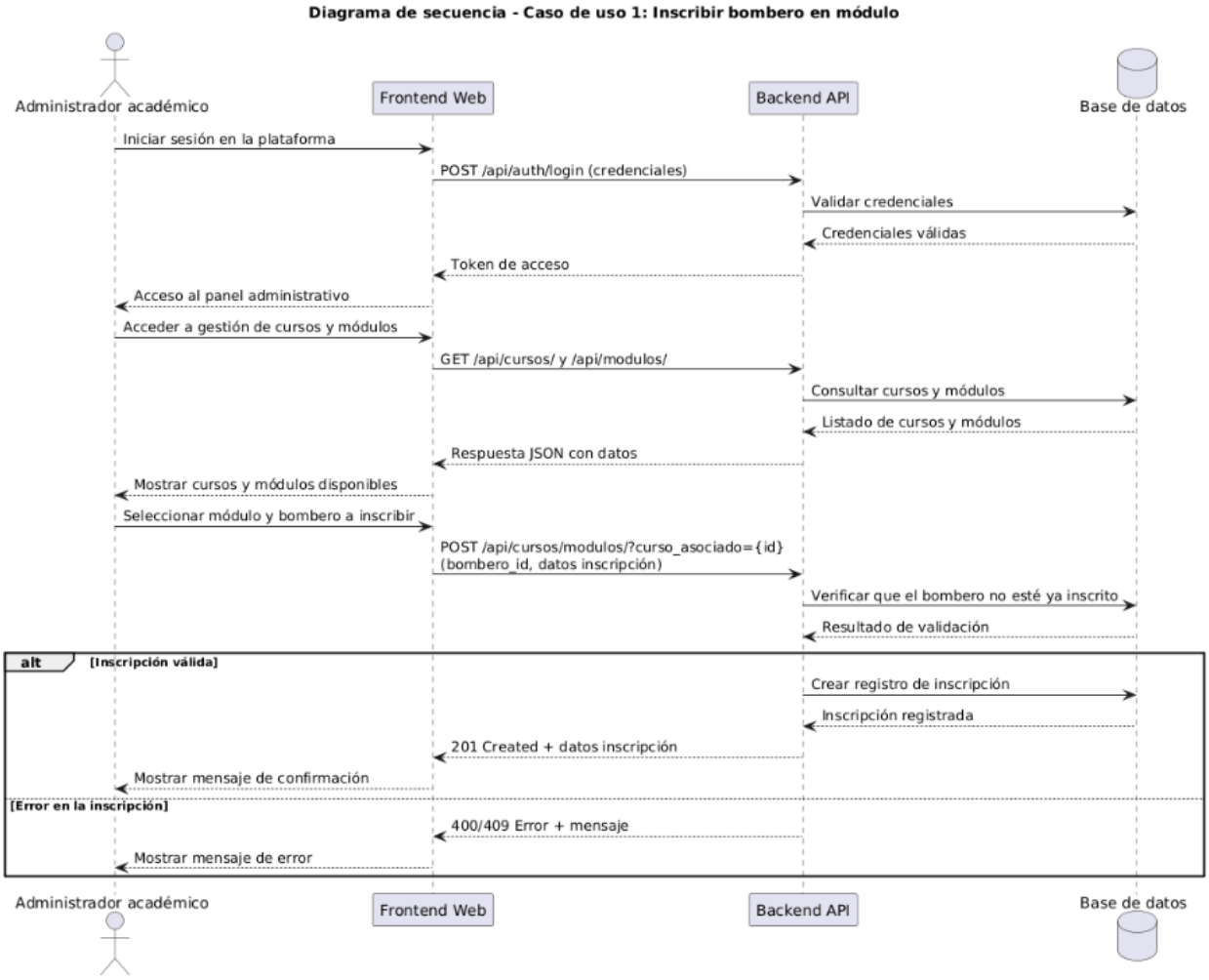


Figura 3-11. Diagrama de Secuencia Caso de uso 1.
Fuente: Elaboración propia.

Caso de uso 2: Enviar notificaciones a alumnos por curso o módulo

CAMPO	DESCRIPCIÓN
ACTOR	Instructor
PROPÓSITO	Permitir al instructor enviar notificaciones a los alumnos inscritos en un curso o módulo específico, con el fin de informar sobre actividades, clases, recordatorios o avisos importantes relacionados con el proceso académico.
FLUJO PRINCIPAL	1. El instructor inicia sesión en la plataforma. 2. El sistema valida sus credenciales y muestra el panel del instructor. 3. El instructor accede a la sección de "Notificaciones". 4. Selecciona el curso o módulo al cual desea enviar una notificación. 5. Ingresa el título y el mensaje de la notificación. 6. Selecciona el público destinatario (curso completo o un módulo específico). 7. Confirma el envío. 8. El sistema registra la notificación en la base de datos. 9. El sistema distribuye la notificación a todos los bomberos inscritos en el curso o módulo seleccionado. 10. El sistema muestra un mensaje indicando que la notificación fue enviada correctamente.
EXCEPCIONES	2ª. Si las credenciales son incorrectas, el sistema rechaza el acceso y muestra un mensaje de error. 4ª. Si el instructor no tiene cursos o módulos asignados, el sistema informa que no es posible enviar notificaciones. 5ª. Si el mensaje o título está vacío, el sistema solicita completar los campos obligatorios. 8ª. Si ocurre un error al registrar la notificación, el sistema cancela el envío e informa del error.
PRECONDICIONES	• El instructor debe tener una cuenta activa y módulos/cursos asignados. • Debe existir al menos un grupo de destinatarios asociado al instructor. • La conexión a internet y la API deben estar operativas. • El sistema debe tener configurado el módulo de notificaciones.
POSTCONDICIONES	• La notificación queda registrada en la base de datos. • Los alumnos inscritos reciben la notificación en su sección correspondiente. • La acción de envío puede quedar registrada para auditoría o historial administrativo.

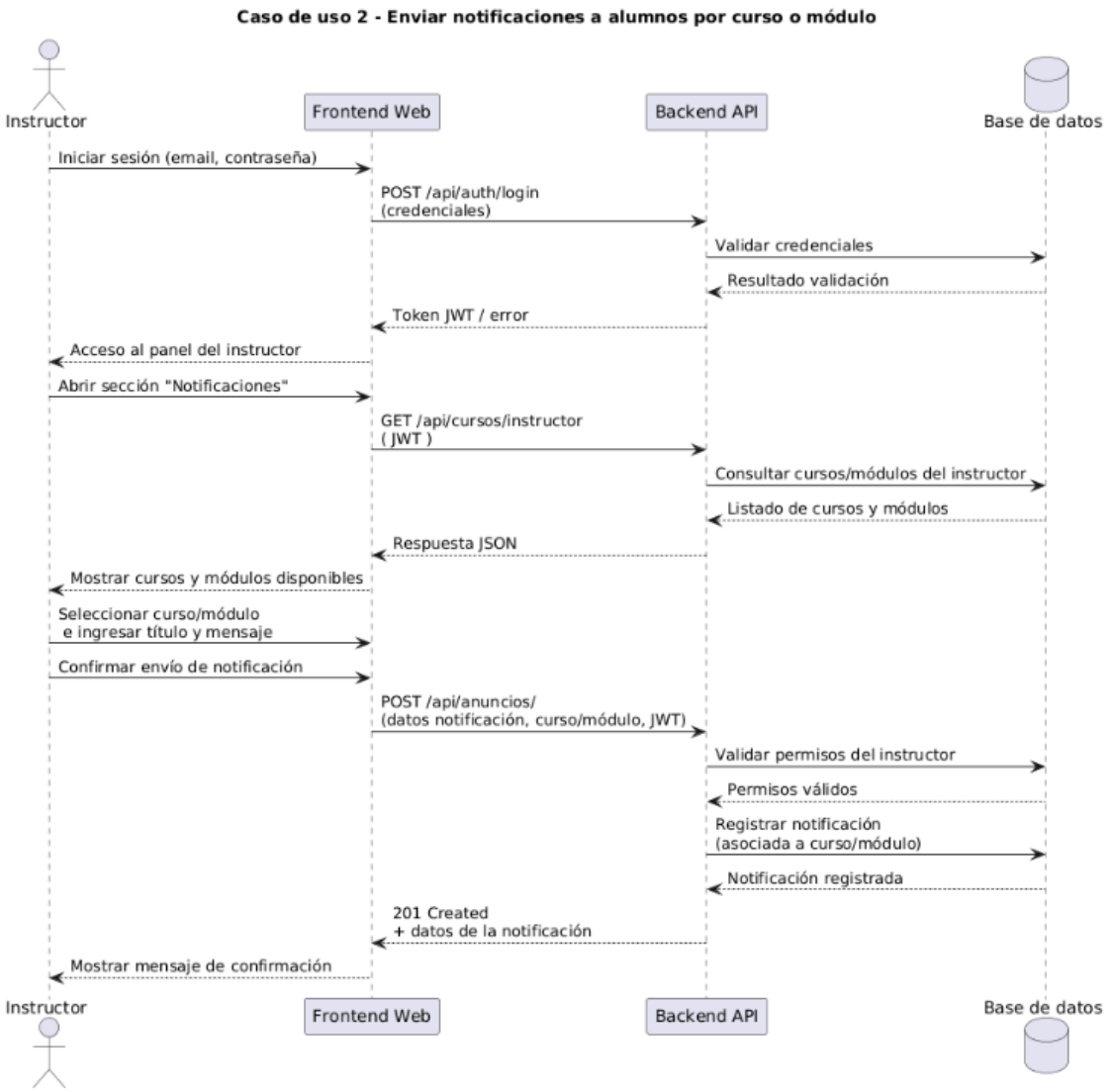


Figura 3-12. Diagrama de Secuencia Caso de uso 2.
Fuente: Elaboración propia.

3.2.3 Control de Versiones.

Se utilizó Git como sistema de control de versiones con la siguiente estructura:

- **main:** versión estable del proyecto.
- **dev:** rama de integración de nuevas funcionalidades.
- **Ramas específicas por módulo:** usuarios, cursos, certificaciones, dashboard, etc.

Este esquema permitió mantener un desarrollo ordenado, facilitar la revisión del código y evitar conflictos entre funcionalidades paralelas.

3.2.4 Integración continua y despliegue.

Para asegurar la calidad del código y la preparación del entorno productivo, se implementaron mecanismos básicos de CI/CD.

Integración continua

- Automatización de pruebas unitarias e integración.
- Validación de cambios antes de fusionar ramas.
- Análisis preventivo para evitar fallos en producción.

Despliegue

- Despliegue manual controlado hacia un servidor en la nube.
- Uso de Docker para asegurar consistencia del entorno.
- Organización de versiones para evitar interrupciones en el entorno académico.

3.3 Pruebas y validación.

Con el propósito de asegurar que el backend desarrollado cumpliera correctamente con los requerimientos funcionales y no funcionales definidos, se aplicó un proceso integral de pruebas y validación. Este proceso abordó tanto la verificación de los componentes individuales del sistema como la validación de flujos académicos completos dentro de entornos controlados, garantizando confiabilidad, rendimiento y consistencia en todas las funcionalidades.

3.3.1 Estrategias de testing aplicadas a los componentes.

El proceso de testing del backend se diseñó para garantizar que cada módulo funcionara correctamente de manera individual y en conjunto. Para ello, se aplicaron diferentes tipos de pruebas orientadas a validar la lógica de negocio, los endpoints expuestos mediante APIs REST y la interacción con la base de datos.

Las estrategias implementadas incluyen las siguientes:

3.3.1.1 Pruebas Unitarias.

Se desarrollaron pruebas unitarias utilizando el framework de testing nativo de Django. Estas pruebas permiten verificar que funciones, validaciones y modelos operen correctamente de manera aislada.

Aspectos validados:

- Creación y actualización de usuarios, cursos y módulos.
- Validación de roles y permisos.
- Lógica de inscripción y restricciones asociadas.
- Integridad de modelos y relaciones en la base de datos.

Ventajas:

- Detectan errores temprano.
- Garantizan estabilidad antes del despliegue.
- Mantienen la calidad del código a largo plazo.

3.3.1.2 Pruebas de Integración.

Objetivo:

Validar la interacción entre los distintos componentes del sistema académico para Bomberos (backend, base de datos y capa de seguridad) asegurando que los flujos completos funcionen correctamente a través de las APIs REST expuestas.

Componentes probados:

- Interacción entre el backend desarrollado en Django REST Framework y la base de datos, verificando la correcta creación y consulta de registros (usuarios, cursos y módulos).
- Flujo de autenticación y autorización mediante login de distintos tipos de usuario (usuario bombero y usuario administrador), confirmando la emisión y uso correcto de tokens JWT.
- Comportamiento de los endpoints REST y su lógica de negocio asociada, tales como:
    1) Inicio de sesión: `/api/user/login_access/`
    2) Listado de cursos: `/api/cursos/`
    3) Consulta de curso por ID: `/api/cursos/{id}/`
    4) Listado de módulos: `/api/modulos/`
    5) Consulta de módulo por ID: `/api/modulos/{id}/`

Herramientas utilizadas:

- **Postman:** herramienta utilizada para ejecutar manualmente solicitudes HTTP (GET y POST) hacia los endpoints del backend y verificar que las respuestas devueltas fueran coherentes con los escenarios definidos. La colección de pruebas incluye peticiones de login para distintos usuarios, así como consultas de cursos y módulos, tal como se observa en la captura de resultados donde todos los llamados retornan códigos HTTP 200.

Metodología:

- Para cada endpoint de la API se definieron y ejecutaron escenarios de prueba, documentando:
    1) Códigos de estado HTTP esperados y obtenidos (por ejemplo, 200 OK para respuestas exitosas, 400 Bad Request para datos inválidos, 401 Unauthorized para solicitudes sin credenciales válidas, entre otros).
    2) Formato de la respuesta, describiendo la estructura de los objetos JSON retornados (atributos como id, nombre, descripcion, etc.) y comparando los datos obtenidos con los registros existentes en la base de datos.
    3) Pruebas con distintos perfiles de usuario, validando que solo los roles autorizados puedan acceder a ciertas operaciones.

3.3.1.3 Pruebas Funcionales.

Validar que las funcionalidades completas del sistema académico cumplen con los requerimientos definidos, asegurando que los flujos reales de interacción entre usuarios, cursos, módulos y roles operen correctamente de principio a fin.

Componentes probados:

- Flujo de creación de usuarios y asignación de roles académicos (Alumno, Instructor, Administrador).
- Inscripción de bomberos en cursos y módulos, incluyendo validación de prerequisitos y estados académicos.
- Visualización del progreso académico y consulta de módulos inscritos.

- Gestión de notificaciones enviadas por instructores a alumnos según curso o módulo.
- Gestión de cursos, módulos y jerarquías académicas por parte del administrador.
- Acceso y descarga de certificados emitidos para los alumnos.
- Control de permisos según rol (acceso restringido a paneles, estadísticas y funcionalidades avanzadas).

Métodos utilizados:

- Pruebas manuales, realizadas desde el frontend Angular, simulando casos de uso reales según cada tipo de usuario.
- Validación de restricciones de acceso, verificando que cada perfil solo pueda ejecutar las acciones permitidas por su rol.
- Comprobación del comportamiento completo del flujo, desde la creación del usuario hasta la interacción con cursos, módulos y certificados.

Ejemplos de pruebas funcionales ejecutadas:

- Un alumno bombero inicia sesión y verifica los módulos en los que está inscrito, revisa su progreso académico y accede a su historial.
- Un instructor envía una notificación masiva a los alumnos de un módulo, y los alumnos pueden visualizarla correctamente en su panel.
- Un administrador académico crea un nuevo curso, define sus módulos asociados y asigna instructores.
- Un alumno descarga sus certificados emitidos desde la sección correspondiente.
- Un usuario sin permisos (por ejemplo, un alumno) intenta acceder al panel de estadísticas del rector, siendo correctamente bloqueado por el sistema.

3.3.1.4 Pruebas de Regresión.

Garantizar que los cambios realizados en el backend no introduzcan fallas en funcionalidades previamente implementadas.

Componentes probados:

- Funcionalidades críticas del sistema académico:
    1) Autenticación.
    2) gestión de usuarios y roles.
    3) Cursos.
    4) Módulos.
    5) Inscripciones.
    6) Notificaciones.
    7) Certificados.
- Flujos completos que abarcan varias partes del backend, tales como:
    1) Creación de un usuario.
    2) Asignación de rol.
    3) Inscripción en un curso.
    4) Asignación de módulo.
    5) Envío y recepción de notificaciones.
    6) Consulta del progreso y certificados.

Métodos utilizados:

- Ejecutar pruebas unitarias mediante Django Test Client después de cada actualización del código.
- Repetir pruebas manuales de integración con Postman [19] y desde el frontend Angular.
- Validaciones periódicas tras cada cambio importante en el backend para asegurar que las funcionalidades previas sigan operando correctamente.

3.3.1.5 Pruebas de Rendimiento.

Evaluar la capacidad del backend para manejar múltiples peticiones simultáneas en escenarios similares al uso real del sistema académico, midiendo su estabilidad, tiempos de respuesta y comportamiento bajo carga.

Componentes probados:

- Autenticación de usuarios mediante JWT.
- Obtención de datos de usuarios (alumnos y profesores).
- Listado de cursos y módulos disponibles.
- Consultas masivas sobre estructuras académicas.
- Operaciones de inscripción y acciones asociadas a cursos o módulos.

Herramientas utilizadas:

- Locust [20], utilizado para simular alto tráfico y medir métricas de rendimiento bajo diferentes condiciones de carga concurrente.

Métricas analizadas:

- Tiempos de respuesta promedio, mediana y percentiles por endpoint.
- Cantidad de errores bajo carga.
- Capacidad del servidor antes de mostrar señales de degradación.
- Máximo número de usuarios simultáneos soportados manteniendo estabilidad operacional.

Metodología utilizada:

- Se configuraron escenarios de carga escalonados, incrementando progresivamente la cantidad de usuarios virtuales.
- Cada usuario virtual ejecutó solicitudes repetidas hacia los endpoints principales realmente utilizados en las pruebas, los cuales fueron:
    1) GET `/api/cursos/` — Listado general de cursos
    2) GET `/api/cursos/modulos/` — Listado de módulos por curso
    3) GET `/api/user/bomberos/estudiantes/` — Listado de alumnos inscritos
    4) GET `/api/user/bomberos/profesores/` — Listado de instructores
    5) GET `/api/user/rangos/` — Catálogo de rangos institucionales
    6) POST `/api/user/login_access/` — Proceso de autenticación JWT

Se generaron reportes detallados que permitieron identificar puntos críticos en los tiempos de respuesta, como el endpoint de autenticación, que presenta mayor carga computacional, así como validar la estabilidad general del sistema bajo escenarios de concurrencia intensiva.

LOCUST | Host: http://localhost:8000 | Status: STOPPED | RPS: 32.5 | Failures: 0% | NEW | RESET

STATISTICS | CHARTS | FAILURES | EXCEPTIONS | CURRENT RATIO | DOWNLOAD DATA | LOGS | LOCUST CLOUD

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	GET /api/cursos/	1345	0	7	18	25	8.13	4	68	1522	11.5	0
GET	GET /api/cursos/modulos/	898	0	12	34	50	15.23	8	71	702	7.1	0
GET	GET /api/user/bomberos/estudiantes/	434	0	39	85	120	46.05	28	164	16961	4.3	0
GET	GET /api/user/bomberos/profesores/	452	0	21	54	67	26.72	15	124	8193	5.2	0
GET	GET /api/user/rangos/	457	0	6	17	25	7.29	3	50	377	4.4	0
POST	POST /api/user/login_access/	100	0	2900	3100	3500	2949.96	2887	3490	993	0	0
	Aggregated	3686	0	11	62	2900	96.31	3	3490	3801.79	32.5	0

Figura 3-13. Resultados de las pruebas de rendimiento 100 usuarios concurrentes.
Fuente: Elaboración propia.

Figura 3-14. Gráfico de rendimiento para 100 usuarios concurrentes.
Fuente: Elaboración propia.

Los resultados obtenidos demostraron que el backend mantiene un comportamiento estable incluso con 100 usuarios concurrentes, registrando tiempos de respuesta entre 7 ms y 120 ms en los endpoints de consulta, sin fallos y con degradación controlada.
Sin embargo, el endpoint de autenticación presentó tiempos considerablemente más altos (aproximadamente 2900 ms), lo que constituye una limitación relevante del

sistema. Este comportamiento evidencia que el proceso de validación de credenciales y generación de tokens JWT requiere optimización, ya que dicho tiempo excede los valores recomendables para garantizar una experiencia de usuario fluida.

En conjunto, las métricas respaldan que el sistema puede operar de forma eficiente bajo cargas típicas en la mayoría de los endpoints, pero también identifican un punto crítico de mejora en el proceso de autenticación que deberá abordarse en trabajos futuros para reducir la latencia inicial del sistema.

3.3.2 Resolución de Bugs y optimización.

Durante el desarrollo del backend se realizaron pruebas continuas orientadas a detectar fallos funcionales, inconsistencias en la lógica de negocio y problemas de rendimiento. Estas pruebas permitieron identificar errores en etapas tempranas y optimizar componentes críticos del sistema. Las principales fuentes de detección fueron:

3.3.2.1 Pruebas Unitarias.

Las pruebas unitarias se aplicaron sobre los modelos, validaciones y funciones internas del sistema. Esto permitió aislar comportamientos incorrectos antes de integrar las funcionalidades con el resto del backend.

Ejemplo detectado:

- Se detectó que, en ciertas condiciones, la autenticación no invalidaba correctamente el token anterior al iniciar una nueva sesión, generando conflictos en la administración de credenciales.
- Asimismo, se identificaron inconsistencias en validaciones de datos para la creación de usuarios e instructores.

3.3.2.2 Pruebas de Integracion.

Estas pruebas permitieron verificar el comportamiento conjunto de controladores, modelos y servicios, especialmente en los flujos académicos más complejos.

Ejemplos detectados:

- En el flujo de inscripción de alumnos a módulos, se evidenció un error provocado por la ausencia de verificación previa de estado académico y prerrequisitos.
- Durante la obtención de listados masivos (cursos, módulos, profesores, alumnos), algunos endpoints presentaban aumentos en el tiempo de respuesta debido a consultas no optimizadas.

3.3.2.3 Pruebas de Rendimiento.

A través de Locust se simularon cargas de usuarios concurrentes para medir el comportamiento del sistema bajo demanda realista y estrés controlado.

Ejemplo detectado:

- Durante las pruebas con 80–100 usuarios simultáneos, se observó que el endpoint de inicio de sesión (`/api/user/login_access/`) aumentaba significativamente su tiempo de respuesta debido al proceso de validación y generación de tokens JWT.

- También se detectó que ciertas consultas a estructuras académicas (como `/api/user/bomberos/estudiantes/`) aumentaban su latencia bajo carga alta debido a la cantidad de datos retornados.

Durante el desarrollo del sistema se aplicaron diversas metodologías para identificar y corregir errores, combinando pruebas automatizadas y validaciones manuales. Para reproducir fallos se utilizaron pruebas unitarias del framework de Django, las cuales permitieron evaluar cambios en modelos y lógica de negocio de forma controlada. Adicionalmente, se empleó Postman para ejecutar solicitudes repetidas sobre los endpoints y confirmar comportamientos anómalos. La depuración se apoyó en la incorporación de logs detallados dentro de las vistas y servicios, facilitando el rastreo del flujo interno de ejecución y la localización precisa de fallos. También se analizaron las consultas SQL generadas por Django, lo cual permitió detectar operaciones ineficientes que afectaban el desempeño del sistema en escenarios de mayor carga.

A partir de esta revisión se aplicaron optimizaciones dirigidas a mejorar el rendimiento y la estabilidad del backend. Entre ellas se incluyó la normalización y reorganización de relaciones entre modelos, el uso de select_related y prefetch_related para reducir el número de consultas, la optimización del proceso de autenticación mediante un manejo más eficiente de tokens y la mejora de la gestión de notificaciones para evitar cargas innecesarias. Estas acciones, sumadas a la integración de pipelines CI/CD para ejecutar pruebas en cada actualización, permitieron disminuir errores funcionales, mejorar los tiempos de respuesta en endpoints críticos y asegurar un comportamiento estable incluso bajo alta concurrencia durante las pruebas realizadas con Locust.

3.4 Integración con Otros Componentes.

Dentro del proyecto, la principal integración externa corresponde al chatbot con Inteligencia Artificial, el cual permite realizar consultas en lenguaje natural sobre la información académica almacenada en el sistema. Esta funcionalidad fue implementada utilizando la API de Google Gemini, que actúa como motor de interpretación semántica y procesamiento del lenguaje natural (NLP).

El chatbot permite a los usuarios formular preguntas del tipo *"¿Qué bomberos tienen aprobado el curso de Materiales Peligrosos?"*, *"¿Qué instructores dictan módulos de rescate?"* o *"¿Cuántos alumnos están inscritos en el Curso Básico Operativo?"*, sin necesidad de navegar manualmente por los distintos módulos del sistema. Para ello, el backend recibe la consulta del usuario, la envía al modelo de Gemini junto con el contexto necesario y, posteriormente, traduce la respuesta del modelo en una consulta estructurada hacia la base de datos del sistema académico. De esta forma, el chatbot retorna información precisa y actualizada, directamente obtenida desde los registros oficiales del backend.

Esta integración mejora significativamente la usabilidad del sistema, ya que permite a bomberos, instructores y administradores obtener información compleja a partir de preguntas simples, reduciendo tiempos de búsqueda y facilitando el acceso a datos académicos. Además, la arquitectura desacoplada entre el backend y el servicio de IA permite ampliar las capacidades del chatbot en el futuro, incorporando nuevas funciones o tipos de consultas sin modificar la estructura central del sistema.

4 Conclusiones

El desarrollo del sistema académico para Bomberos permitió aplicar de manera integrada conceptos fundamentales de ingeniería de software, especialmente en el diseño e implementación de un backend escalable y seguro. El uso de metodologías ágiles como SCRUM facilitó la planificación iterativa y el cumplimiento progresivo de las funcionalidades establecidas, mientras que la adopción de una arquitectura basada en APIs REST permitió construir un sistema modular y adaptable a distintos tipos de clientes.

A lo largo del proyecto se fortalecieron competencias técnicas relacionadas con la gestión de datos, el diseño de modelos, la administración de roles y permisos, y la optimización del rendimiento del servidor. Las pruebas unitarias, de integración y de carga permitieron validar la estabilidad del sistema y corregir oportunamente errores críticos. Esto aseguró que el backend cumpliera con los requisitos institucionales de disponibilidad, seguridad y consistencia.

El proceso también permitió identificar decisiones que, en una nueva iteración, se abordarían de forma distinta, como la planificación temprana de ciertos componentes y la definición más precisa de algunos requerimientos funcionales. Entre los principales desafíos destacaron la gestión de permisos, la estructura del modelo académico y el rendimiento bajo alta demanda. Como trabajo futuro, se propone mejorar la automatización del despliegue, incorporar herramientas avanzadas de monitoreo e integrar nuevos módulos administrativos para ampliar la utilidad del sistema.

En términos personales, este proyecto permitió desarrollar habilidades en trabajo colaborativo, comunicación y resolución de problemas, además de reafirmar la importancia de un diseño backend bien estructurado para la operación de sistemas institucionales. Finalmente, el sistema construido aporta de manera significativa a la gestión académica dentro de Bomberos, mejorando la organización de la información y sentando una base sólida para futuras extensiones tecnológicas.

5 Agradecimientos

Agradezco profundamente a quienes hicieron posible la realización de este trabajo. A mis padres, abuelos y hermana, por su apoyo incondicional durante toda mi formación académica. A mi novia, por su compañía, paciencia y constante motivación. Extiendo también mi agradecimiento a mis profesores y compañeros, quienes contribuyeron con sus conocimientos, retroalimentación y colaboración en distintas etapas del proyecto. Su apoyo colectivo fue fundamental para culminar con éxito esta tesina.

6 Referencias

[1] Bomberos de Chile. "Información General de Bomberos". *bomberos.cl*. Disponible en: https://www.bomberos.cl/informacion-general-de-bomberos
[2] Moodle, *Moodle Documentation*, 2025. [En línea]. Disponible en: https://docs.moodle.org/
[3] Instructure, *Canvas LMS REST API Documentation*, 2025. [En línea]. Disponible en: https://canvas.instructure.com/doc/api/
[4] Open edX, *Open edX Developer Documentation*, 2025. [En línea]. Disponible en: https://docs.edx.org/
[5] Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide*. Scrum.org. Disponible en: https://scrumguides.org
[6] Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.
[7] Driessen, V. (2010). *A successful Git branching model*. Disponible en: https://nvie.com/posts/a-successful-git-branching-model/
[8] Roberts, M., & Chapin, J. (2017). *Serverless Architectures on AWS: With examples using AWS Lambda*. Manning Publications.
[9] Villamizar, M., et al. (2016). "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud." *2016 10th International Conference on Advanced Computing and Applications (ACOMP)*. IEEE.
[10] Gamma et al. – *Design Patterns* https://refactoring.guru/es/design-patterns
[11] The Scrum Guide - https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf
[12] Docker Docs – Overview https://docs.docker.com/get-started/overview/
[13] NIST CloudComputing Definition https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
[14] Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media. https://www.python.org/
[15] Django Software Foundation. *Django Documentation*. Disponible en: https://www.djangoproject.com/
[16] PostgreSQL Global Development Group. *PostgreSQL Documentation*. Disponible en: https://www.postgresql.org/docs/
[17] OWASP Foundation. *JSON Web Token (JWT) Cheat Sheet for Java*. Disponible en: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html
[18] GitHub, Inc. (2025). *About GitHub*. Disponible en: https://github.com/about
[19] Postman, Inc. (2024). *Postman API Platform* [Software]. https://www.postman.com/
[20] Locust.io Community. (2024). *Locust – Load Testing Tool* [Software]. https://locust.io/

7 Anexos

Repositorio de documentación del proyecto.