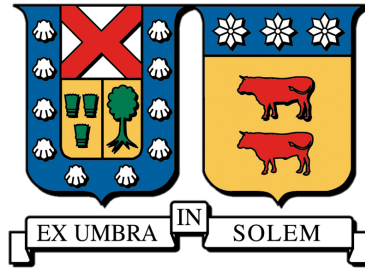


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



**“DISEÑO Y DESARROLLO DE UN SISTEMA DE
VIDEOVIGILANCIA OPTIMIZADO CON
INTELIGENCIA ARTIFICIAL Y PLATAFORMA WEB
INTEGRADA:
ENFOQUE EN PLATAFORMA WEB”**

PAULA BELÉN MALDONADO CASTILLO

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO
CIVIL TELEMÁTICO**

PROFESOR GUÍA:

NICOLÁS JARA

PROFESOR CO-REFERENTE:

FRANCISCO CABEZAS

May 27, 2024

Agradecimientos

A mis padres, por nunca soltar mi mano y siempre brindarme su apoyo incondicional. En mis momentos más oscuros, su amor y guía me han dado la fortaleza para seguir adelante. Gracias por enseñarme el verdadero significado de la perseverancia y el amor incondicional.

A mi hermano Diego, por ser mi compañero de aventuras y mi apoyo constante. Tu lealtad inquebrantable y tu espíritu aventurero han sido una fuente de inspiración y alegría en mi vida. Gracias por cada momento compartido y por ser siempre un refugio seguro.

A Andrés, por ser el compañero que necesitaba en el momento justo. Tu presencia y apoyo han sido fundamentales para superar desafíos y celebrar triunfos. Gracias por estar a mi lado cuando más lo necesitaba.

A Javiera, por estar conmigo desde el inicio de este viaje. Tu amistad y constancia han sido un pilar fundamental en mi vida. A Alan, por ser mi partner en esta travesía, tu compañía y apoyo han hecho de este camino una experiencia inolvidable. Gracias por cada momento compartido.

A mis amigos Alexis, Catalina, Mari, Eric y Piero, por acompañarme en mis últimos años de carrera. Su amistad, risas y apoyo han hecho de estos años una etapa memorable y enriquecedora. Gracias por estar siempre ahí, compartiendo sueños y desafíos.

Resumen

La seguridad en Chile representa una de las principales preocupaciones a nivel nacional. A pesar de los esfuerzos por implementar leyes y medidas punitivas, persisten inquietudes respecto a este tema. Esta sensación de inseguridad se manifiesta de manera notable, especialmente en el ámbito residencial.

En este escenario, se hace evidente la necesidad de adoptar enfoques preventivos específicos para los hogares, con el objetivo de brindar a las personas la tranquilidad necesaria en su entorno más íntimo. Esta problemática resalta la importancia de abordarla de manera integral y proactiva para mejorar la calidad de vida de los ciudadanos y fortalecer la seguridad en los hogares chilenos.

En este contexto, es fundamental la implementación de estrategias preventivas dirigidas específicamente a los entornos residenciales. El objetivo de este trabajo de titulación es desarrollar un MVP que incluya una interfaz web para recibir notificaciones de eventos anómalos en tiempo real, asegurando así una respuesta rápida y eficiente ante situaciones de riesgo.

Abstract

Security in Chile represents one of the main concerns at the national level. Despite efforts to implement laws and punitive measures, concerns persist regarding this issue. This sense of insecurity is particularly noticeable, especially in the residential sphere.

In this scenario, the need for adopting specific preventive approaches for households becomes evident, aiming to provide people with the necessary peace of mind in their most intimate environment. This issue underscores the importance of addressing it comprehensively and proactively to enhance the quality of life for citizens and strengthen security in Chilean homes.

Within this context, the implementation of preventive strategies targeted specifically at residential environments is fundamental. The aim of this thesis project is to develop an MVP (Minimum Viable Product) that includes a web interface for receiving notifications of anomalous events in real-time, thus ensuring a quick and efficient response to risky situations.

Glosario

API: es como un intermediario que permite que diferentes aplicaciones se comuniquen y compartan datos entre sí de manera estandarizada y segura.

Array: es una estructura de datos que permite almacenar múltiples valores del mismo tipo en una sola variable.

Back-end: se refiere a la parte de un sistema o aplicación que se encarga de procesar y gestionar los datos, así como de llevar a cabo la lógica y la funcionalidad de la aplicación.

CORS: es una política de seguridad en los navegadores web que controla cómo los recursos de una página pueden ser solicitados desde un dominio diferente.

Endpoints: es un punto de acceso o una dirección URL específica en un servicio web o una API donde se puede realizar una solicitud para interactuar con ese servicio y acceder a recursos o funcionalidades específicas.

.env: es un archivo de configuración utilizado para almacenar variables de entorno sensibles o específicas del entorno de desarrollo, como claves de API o credenciales de bases de datos.

Front-end: es la parte de un sistema informático o una aplicación que interactúa directamente con el usuario.

HTTP: es un protocolo de comunicación utilizado para la transferencia de datos en la World Wide Web.

JWT: es un estándar abierto (RFC 7519) que define un método compacto y seguro para transmitir información entre partes.

ORM: es una técnica de programación que permite a los desarrolladores manipular y trabajar con datos de una base de datos relacional utilizando objetos en lugar de consultas SQL directas.

Request: se refiere a una petición realizada por un cliente (como un navegador web o una aplicación) a un servidor para acceder a un recurso específico o realizar una operación determinada.

Token: es un elemento digital que representa la autorización para acceder a recursos específicos o realizar determinadas acciones en un sistema informático.

URL: es una dirección única que se utiliza para localizar recursos en la web, como páginas web, archivos, imágenes u otros recursos en Internet.

UUID: es un identificador único universalmente utilizado en la informática.

Acrónimos

API: Application Programming Interface

BD: Base de Datos

CORS: Cross-Origin Resource Sharing

HTML: HyperText Markup Language

HTTP: Hypertext Transfer Protocol

JSON: JavaScript Object Notation

JWT: JSON Web Token

ORM: Object Relation Mapper

REST: Representational State Transfer

RUT: Rol Único Tributario

TI: Tecnología de la Información

URL: Uniform Resource Locators

UUID: Universally Unique Identifier

XML: Extensible Markup Language

Índice de figuras

2.1. Interfaz Gráfica de MaVis	11
2.2. Interfaz Gráfica de Frigate	11
3.1. Diagrama de Contexto	29
3.2. Arquitectura Propuesta	30
3.3. Modelo de Dominio de Base de Datos API-Usuarios.	35
4.1. Respuesta Correcta de getAllUsers de API-Usuarios.	38
4.2. Respuesta Correcta de getEventsByDateAndUser de API-Usuarios. . .	39
4.3. Respuesta Correcta de createEvent de API-Usuarios.	40
4.4. Respuesta Correcta de createUser de API-Usuarios.	41
4.5. Respuesta Correcta de Login de API-Usuarios.	42
4.6. Respuesta Correcta de getAllNotifications de API-Intrusos.	43
4.7. Respuesta Correcta de getAllNotificationsByDate de API-Intrusos. . .	44
4.8. Respuesta Correcta de getLastNotification de API-Intrusos.	45
4.9. Respuesta Correcta de getLastNoPublishedNotification de API-Intrusos	46
4.10. Respuesta Correcta de createNotification de API-Intrusos	47
4.11. Respuesta Correcta de updatePublishedStateNotification de API-Intrusos.	48
4.12. Respuesta Correcta de updateNotificationData de API-Intrusos.	49
4.13. Respuesta Correcta de searchLicensePlate de API-Patentes.	51
4.14. Respuesta Correcta de getAllNotifications de API-Patentes.	52
4.15. Respuesta Correcta de getAllNotificationsByDate de API-Patentes. . .	53

4.16. Respuesta Correcta de getLastNotification de API-Patentes.	54
4.17. Respuesta Correcta de getLastNoPublishedNotification de API-Patentes.	55
4.18. Respuesta Correcta de createNotification de API-Patentes.	56
4.19. Respuesta Correcta de createVisitor de API-Patentes.	57
4.20. Respuesta Correcta de createResident de API-Patentes.	58
4.21. Respuesta Correcta de updatePublishedStateNotification de API-Patentes.	59
4.22. Respuesta Correcta de getAllVisitorsByDate de API-Patentes.	60
4.23. Respuesta Correcta de Login de API-Gateway	62
4.24. Respuesta Correcta de getAllNoPublishedNotifications de API-Gateway.	63
4.25. Respuesta Correcta de getAllNotificationsByDate de API-Gateway.	64
4.26. Respuesta Correcta de VisitorRegistration de API-Gateway.	65
4.27. Respuesta Correcta de createEventByUser de API-Gateway.	66
4.28. Respuesta Correcta de getLastVideoNotification de API-Gateway.	67
4.29. Respuesta Correcta de getLastPlateNotification de API-Gateway.	68
4.30. Respuesta Correcta de updateLastVideoNotification de API-Gateway.	69
4.31. Respuesta Correcta de updatePublishedState de API-Gateway.	70
4.32. Respuesta Correcta de updateResolutionNotification de API-Gateway.	71
4.33. Diagrama de Secuencia Inicio de Sesión.	73
4.34. Diagrama de Flujo de Inicio de Sesión.	74
4.35. Vista implementada de Login.	74
4.36. Vista implementada de Menú principal.	75
4.37. Diagrama de secuencia RF7.	76
4.38. Diagrama de secuencia RF6.	76
4.39. Diagrama de Flujo de Usuario para el Panel de Videovigilancia.	77
4.40. Vista de notificación de intrusos.	77
4.41. Vista de formulario para modificar notificación de intrusos.	78
4.42. Diagrama de secuencia RF8.	79
4.43. Diagrama de secuencia RF2.	80
4.44. Diagrama de Flujo RF2.	80

4.45. Vista de notificación de vehículo desconocido.	81
4.46. Vista de registro de visita.	81
4.47. Diagrama de secuencia RF3.	82
4.48. Diagrama de secuencia RF4.	83
4.49. Diagrama de secuencia RF5.	83
4.50. Diagrama de Flujo de Nuevo Evento.	84
4.51. Vista de resgistro de nueva visita.	84
4.52. Vista de Registro de evento anómalo.	85
5.1. Resultados de Code Coverage de API-Usuarios	88
5.2. Resultados de Code Coverage de API-Intrusos	88
5.3. Resultados de Code Coverage de API-Patentes	88
5.4. Resultados de Code Coverage de API-Gateway	89

Índice general

1. Introducción	5
1.1. Objetivos	6
1.1.1. Objetivo General	6
1.1.2. Objetivos Específicos	6
1.2. Contribuciones	7
2. Marco Teórico	9
2.1. Antecedentes	9
2.2. Estado del arte	10
2.2.1. Discusión	12
2.3. Marco Conceptual	12
3. Desarrollo de la solución	15
3.1. Análisis de requerimientos	15
3.1.1. Requisitos Funcionales del Sistema	15
3.1.2. Requisitos no Funcionales del Sistema	16
3.2. Stack de Tecnologías para la solución	18
3.2.1. Lenguaje de programación	18
3.2.1.1. Discusión	20
3.2.2. Framework de Back-end	21
3.2.2.1. Discusión	22
3.2.3. Framework de Front-end	23

3.2.3.1.	Discusión	24
3.2.4.	Bases de Datos	24
3.2.4.1.	Discusión	26
3.2.5.	Dependencias y Contenedores	27
3.3.	Diagrama de Contexto	29
3.4.	Arquitectura del sistema	30
3.4.1.	Servicios Propuestos	30
3.4.2.	Bases de Datos	31
3.4.2.1.	Base de datos de API-Intrusos	31
3.4.2.2.	Base de datos de API-Patentes	33
3.4.2.3.	Base de datos de API-Usuarios	34
4.	Resultados	37
4.1.	Back-end	37
4.1.1.	API-Usuarios	38
4.1.1.1.	Obtener todos los usuarios de la aplicación	38
4.1.1.2.	Obtener los eventos por Fecha y Usuario	39
4.1.1.3.	Registrar eventos del usuario	40
4.1.1.4.	Creación de nuevos usuarios	41
4.1.1.5.	Login de la aplicación	42
4.1.2.	API-Intrusos	43
4.1.2.1.	Obtención de todas las notificaciones	43
4.1.2.2.	Obtención de las notificaciones por fecha	44
4.1.2.3.	Obtención de la última notificación	45
4.1.2.4.	Obtener la última notificación no publicada	46
4.1.2.5.	Creación de nueva notificación	47
4.1.2.6.	Actualización de Estado Publicado	48
4.1.2.7.	Actualizar descripción y gravedad	49
4.1.3.	API-Patentes	50

4.1.3.1.	Buscar las patentes registradas	50
4.1.3.2.	Obtener todas las notificaciones	52
4.1.3.3.	Obtener todas las notificaciones de un determinado día	53
4.1.3.4.	Obtener última notificación	54
4.1.3.5.	Obtener la última publicación no publicada	55
4.1.3.6.	Creación de nueva notificación	56
4.1.3.7.	Creación de nuevo visitante	57
4.1.3.8.	Creación de nuevo Residente	58
4.1.3.9.	Actualización de Estado Publicado	59
4.1.3.10.	Obtener todas las visitas registradas en una fecha	60
4.1.4.	API-Gateway	61
4.1.4.1.	Login	61
4.1.4.2.	Obtener Todas las Notificaciones No Publicadas	63
4.1.4.3.	Obtener todos los eventos filtrados por fecha	64
4.1.4.4.	Registrar un visitante	65
4.1.4.5.	Registrar evento anómalo	66
4.1.4.6.	Obtener Última Notificación de API-Intrusos	67
4.1.4.7.	Obtener última notificación de API-Patentes	68
4.1.4.8.	Modificar campos de Notificaciones de API-Intrusos	69
4.1.4.9.	Modificar Estado Publicado de Notificación	70
4.1.4.10.	Modificar Resolución de Notificación de Api-Patentes	71
4.2.	Front-end	72
4.2.1.	Vista de Inicio de Sesión	73
4.2.1.1.	Implementación de Vista	74
4.2.2.	Vista de Menú Principal	75
4.2.2.1.	Panel de Alertas de Videovigilancia	75
4.2.2.2.	Panel de Entrada y Salida de Vehículos	79
4.2.2.3.	Historial de Eventos	82

5. Validaciones y Pruebas	87
5.1. Verificación y Pruebas	87
5.1.1. API-Usuarios	88
5.1.2. API-Intrusos	88
5.1.3. API-Patentes	88
5.1.4. API-Gateway	89
5.2. Validaciones	89
6. Conclusiones y Trabajos futuros	91
6.1. Trabajos futuros	92

Capítulo 1

Introducción

En Chile, la seguridad se funde como una de las problemáticas más apremiante a nivel nacional. Según los resultados de la Encuesta Nacional de Victimización y Percepción de Seguridad Ciudadana (ENUSC) 2022 [1], un preocupante 21,8 % de las personas encuestadas ha sufrido algún tipo de delito en su hogar. A pesar de los esfuerzos desplegados para promulgar legislaciones orientadas a sancionar a quienes infringen la ley, la sensación de inseguridad entre las víctimas persiste.

En este escenario, se hace evidente la urgencia de implementar enfoques preventivos específicos para los entornos residenciales. La finalidad de estas medidas es proporcionar a las personas la posibilidad de experimentar una mayor tranquilidad en el espacio que debería ser su refugio. Esta problemática adquiere una relevancia significativa, destacando la necesidad de abordarla de manera integral y proactiva con el fin de mejorar la calidad de vida de la población y fortalecer la seguridad en los hogares chilenos.

Por esta razón, Jaquard SPA nos encomendó la tarea de crear un sistema proactivo que simplifique las labores de los conserjes en condominios, permitiéndoles anticiparse a posibles incidentes delictivos mediante la detección de intrusiones en áreas restringidas, al mismo tiempo que garantiza un control de acceso vehicular optimizado. El propósito fundamental de este sistema es mejorar y complementar las medidas

de seguridad ya implementadas en los condominios. Este trabajo de titulación se enfocará en el desarrollo de la plataforma web, la cual estará integrada con el detector de patentes y el detector de intrusos. De esta manera, será capaz de recibir notificaciones en tiempo real de ambos sistemas.

1.1. Objetivos

1.1.1. Objetivo General

El objetivo general es desarrollar una aplicación web con una arquitectura modularizada que asegure la escalabilidad del producto, centrándose en la notificación en tiempo real de eventos significativos en el condominio.

1.1.2. Objetivos Específicos

- Notificar al usuario acerca de situaciones anómalas captadas por las cámaras de videovigilancia.
- Reconocer la matrícula de vehículos, tanto de residentes como de visitantes.
- Registrar eventos ocurridos dentro del condominio.
- Evaluar y seleccionar las herramientas más idóneas para el desarrollo de la plataforma.
- Analizar y determinar las bases de datos más apropiadas para abordar el problema.

1.2. Contribuciones

Este trabajo de título fue realizado en colaboración con Alexis Alva y Catalina Zelaya.

Se sugiere consultar el trabajo de título de Alexis Alva para obtener más detalles sobre la detección de intrusos, y el trabajo de título de Catalina Zelaya para más información sobre la detección y reconocimiento de placas vehiculares.

Capítulo 2

Marco Teórico

En esta sección se proporcionarán los antecedentes que justifican la necesidad de desarrollar la solución propuesta, se presentará el estado actual del arte y se establecerá el marco conceptual que sustenta el enfoque adoptado.

2.1. Antecedentes

Durante el año 2022, se observó un notable incremento en los casos policiales de Delitos de Mayor Connotación Social (DMCS) en Chile [2], los cuales comprenden tanto detenciones por flagrancia como denuncias formales realizadas por la ciudadanía ante Carabineros de Chile o la Policía de Investigaciones (PDI).

En comparación con el año anterior, la tasa de casos policiales de DMCS informados por ambas fuerzas policiales experimentó un significativo crecimiento del 44,6

Entre los delitos que más contribuyeron a este aumento destacan el robo con violencia o intimidación, con un incremento del 63,1 %, seguido por el robo por sorpresa con un 61,2 %. Además, se registraron notables aumentos en los robos en lugar no habitado (56,4 %) y los robos en lugar habitado (44,6 %).

A nivel nacional, las denuncias presentadas por la población durante el mismo periodo exhibieron un incremento del 46,1 %, mientras que las detenciones aumentaron en un 35,6 % en comparación con el año anterior.

Este incremento en la incidencia delictiva no fue exclusivo de una región en particular, ya que todas las regiones del país mostraron un alza en la tasa de DMCS respecto al año 2021. Las regiones que destacaron por su significativo crecimiento fueron la Región de Atacama, con un incremento del 63,9 %, seguida por la Región de Arica y Parinacota con un aumento del 59,3 %, y la Región de Valparaíso con un crecimiento del 58,1 %.

2.2. Estado del arte

En esta sección se presentan soluciones relacionadas que abordan aspectos relevantes para el desarrollo del presente trabajo de título.

- Watsor [3] se destaca en videovigilancia al emplear redes neuronales para una detección inteligente, reduciendo falsos positivos. Permite delimitar áreas de detección con máscaras alfa y es compatible con aceleradores como The Coral USB y Nvidia CUDA GPUs. Utiliza MQTT para informar objetos detectados e integrarse con HomeAssistant, controlando el decodificador de vídeo por MQTT. Aunque no presenta una interfaz gráfica, ofrece una sólida funcionalidad en la detección de objetos en flujos de video.
- MaVis (Machine Vision Security System) [4] es un software diseñado para identificar posibles intrusos y proporcionar registros visuales para una respuesta eficiente. Utiliza cámaras integradas en Jetson Nano para detectar en tiempo real la presencia de personas y enviar notificaciones por correo electrónico a los propietarios. El sistema utiliza servicios de Amazon Web Services para la entrega de notificaciones.



Figura 2.1: Interfaz Gráfica de MaVis

- Frigate [5] es un software que se centra en la detección de personas en cámaras de seguridad. Destaca por permitir la delimitación de zonas y notificar cuando estas son transgredidas. Su interfaz gráfica muestra notificaciones de transgresiones de zonas y proporciona vistas en tiempo real de las cámaras de video.



Figura 2.2: Interfaz Gráfica de Frigate

2.2.1. Discusión

Según la información recopilada de sistemas anteriores y centrándose en el enfoque de este trabajo de título, se han identificado diversas opciones. La primera alternativa ([3]) carece de una interfaz gráfica propiamente tal, lo que la hace poco factible para el contexto de este trabajo.

Por otro lado, la segunda alternativa implica el envío de notificaciones a los residentes de las viviendas. Sin embargo, el uso del correo electrónico como medio de entrega plantea la posibilidad de alertas tardías, ya que puede no revisarse de forma constante.

Finalmente, la tercera opción ([5]) presenta una interfaz gráfica, aunque su funcionamiento se restringe a un entorno local en lugar de basarse en la nube. Esta característica podría generar complicaciones durante la implementación del sistema.

La solución propuesta incorporará una interfaz gráfica que permitirá notificar al usuario sobre la presencia de intrusos, al mismo tiempo informará sobre el ingreso de vehículos a la zona residencial. Esto último hará que se diferencie de las soluciones anteriores.

2.3. Marco Conceptual

En esta sección se definirá conceptos esenciales para el desarrollo de este trabajo.

Detector de Intrusos: es un sistema de software que utiliza inteligencia artificial para detectar la presencia de personas en áreas previamente delimitadas.

Detector de Patentes: es un sistema de software que está diseñado para identificar la matrícula de vehículos motorizados.

Microservicios: son un enfoque arquitectónico y organizativo para el desarrollo de software, donde el software está compuesto por pequeños servicios indepen-

dientes que se comunican a través de API bien definidas. Los propietarios de estos servicios son equipos pequeños independientes. [6]

Base de Datos: es una recopilación de datos sistemática y almacenada electrónicamente.[7]

Back-end: es la parte del desarrollo web que se encarga de que toda la lógica de una página web funcione. [8]

Front-end: se refiere a la parte visible de un sitio web o aplicación con la que los usuarios pueden interactuar directamente. [9]

Plataforma web: es el software y la tecnología que se utilizan para unificar y optimizar las operaciones de negocio y los sistemas de TI. [10]

API Rest: es una interfaz de comunicación entre sistemas de información que usa el protocolo de transferencia de hipertexto (HTTP) para obtener datos o ejecutar operaciones sobre dichos datos en diversos formatos, como pueden ser XML o JSON. [11]

Capítulo 3

Desarrollo de la solución

En esta sección se presentará el análisis y diseño de la plataforma para su posterior implementación.

3.1. Análisis de requerimientos

En esta sección se presentará un análisis de requerimientos funcionales indispensables para obtener una primera versión de la aplicación.

3.1.1. Requisitos Funcionales del Sistema

Los requisitos funcionales delimitan las acciones que el sistema debe llevar a cabo para cumplir con las necesidades o expectativas del usuario. En este trabajo de título, se han identificado los siguientes requisitos funcionales tras sostener conversaciones con la empresa responsable del desafío.

- **RF1:** El sistema debe ser capaz de notificar al usuario al detectar a un intruso en alguna zona restringida.
- **RF2:** El sistema debe ser capaz de notificar al usuario al detectar una patente desconocida.

- **RF3:** El sistema debe ser capaz de mostrar los eventos que han ocurrido en el último día.
- **RF4:** El sistema debe ser capaz de registrar un visitante.
- **RF5:** El sistema debe permitir al usuario registrar eventos puntuales que ocurran en su turno laboral.
- **RF6:** El sistema debe permitir modificar la descripción y la gravedad de la alerta.
- **RF7:** El sistema debe permitir al usuario ver la última notificación enviada por el detector de intrusos.
- **RF8:** El sistema debe permitir al usuario ver el último vehículo que ingresó al condominio y cual fue la resolución de su entrada.
- **RF9:** El sistema debe solicitar la autenticación del usuario para ingresar al menú principal.

3.1.2. Requisitos no Funcionales del Sistema

Los requisitos no funcionales de un sistema (RNF) o atributos de calidad, son requisitos que especifican criterios que pueden usarse para juzgar la operación, cualidades o características de un sistema de software. A diferencia de los funcionales, no determinan la funcionalidad del sistema a desarrollar.

- **RNF1:** El sistema debe actualizar los datos en periodos de medio segundo para poder tener alertas en tiempo real.
- **RNF2:** El sistema debe contar con una interfaz intuitiva para no complicar el trabajo del conserje.
- **RNF3:** El sistema debe contar con un Uptime del 99,9 %, ya que no puede estar caído por tiempos prolongados.

- **RNF4:** El sistema debe ser implementado utilizando contenedores Docker.
- **RNF5:** El sistema debe emplear JWT para garantizar la seguridad de las rutas.
- **RNF6:** El sistema debe ofrecer una vista compacta que muestre los módulos de detección de patentes y detección de intrusos.
- **RNF7:** Las vistas del sistema deben estar modularizadas, permitiendo la remoción de los módulos no requeridos.

3.2. Stack de Tecnologías para la solución

En esta sección, se detallarán las tecnologías empleadas en el desarrollo de la solución, y se realizará una comparativa con otras tecnologías utilizadas para el mismo propósito, con el fin de justificar la elección de las tecnologías seleccionadas. Este análisis se dividirá en cuatro partes principales: el lenguaje de programación, el framework utilizado en el Back-end, el framework utilizado en el Front-end y las bases de datos.

3.2.1. Lenguaje de programación

Un lenguaje de programación, en términos simples, constituye el conjunto de instrucciones mediante el cual los seres humanos se comunican con las computadoras [12] . En la actualidad, existen numerosos lenguajes de programación, y la elección de uno debe basarse en diversos aspectos. Para los fines de este trabajo de memoria, el criterio principal que guiará dicha elección será la idoneidad del lenguaje para el desarrollo de aplicaciones web y posteriormente la idoneidad que tendrá con la solución.

A continuación, se presentará una tabla que muestra los lenguajes óptimos para el desarrollo web, junto con sus ventajas y desventajas principales.

Lenguaje	Ventajas	Desventajas
Javascript	■ De código abierto ■ JavaScript del lado del cliente es increíblemente rápido ya que no requiere compilación. ■ Es ideal para mejorar la experiencia del usuario y la interacción de un sitio web. ■ Amplia comunidad de desarrolladores.	■ Sintaxis a veces confusa. ■ Rendimiento inestable entre navegadores.
Python	■ De Código Abierto ■ Altamente Escalable ■ Apoyo de la Comunidad en constante crecimiento. ■ Sintaxis clara y legible.	■ Más lento por ser un lenguaje interpretado. ■ Consume mucha memoria. ■ No es ideal para el desarrollo móvil.
Ruby	■ Sintaxis simple y elegante. ■ De código abierto. ■ Comunidad grande y activa.	■ Menos flexible. ■ Problemas de multihilo.

C#	■ Rápidos tiempos de compilación y ejecución. ■ Perfecto para todo tipo de desarrollo en Windows.	■ La curva de aprendizaje es pronunciada. ■ No se traduce bien entre plataformas, ya que depende del entorno .NET.
PHP	■ De código abierto. ■ Las versiones iguales y superiores a la 7 son rápidas. ■ Gran compatibilidad con navegadores. ■ Gran apoyo de la comunidad.	■ Posee algunas brechas de seguridad. ■ La creación de sitios web es más lenta que en otros lenguajes. ■ Menos herramientas de depuración.

Cuadro 3.1: Tabla comparativa de Lenguajes de Programación.

3.2.1.1. Discusión

Para el lenguaje de programación, se decidió elegir JavaScript por su alta compatibilidad con la solución, su extensa gama de bibliotecas y frameworks que simplifican el desarrollo, su facilidad para integrarse con otras tecnologías y servicios web (especialmente relevante dado que trabajaremos con microservicios), y el respaldo de una comunidad activa y actualizaciones frecuentes.

3.2.2. Framework de Back-end

Un framework es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software [13]. En el ámbito de este trabajo de memoria, nos enfocaremos en buscar un framework de Javascript que se ajuste a la solución y al desarrollo.

A continuación, se describen una serie de frameworks utilizados para el desarrollo web y que utilizan de motor a Javascript.

Framework	Ventajas	Desventajas
Express.js	■ Flexible y Rápido. ■ Altamente Escalable. ■ Sistema de enrutamiento robusto. ■ Mayor control del flujo de la aplicación. ■ Gran comunidad.	■ Menos estructura y convenciones. ■ Al ser ligero, el desarrollador debe tener una mayor toma de decisiones.
Nest.js	■ Compatible con Typescript. ■ Documentación exhaustiva. ■ Buena arquitectura y desarrollo rápido.	■ Depuración ardua. ■ Curva de aprendizaje más pronunciada que otros frameworks.

Koa.js	■ Facilita el manejo de errores. ■ También se puede utilizar como servidor HTTP. ■ Permite el escalamiento de la aplicación de forma sencilla.	■ No proporciona funcionalidades adicionales.
Hapi.js	■ Fácil de entender. ■ Promete mantener una estructura. ■ Orientado para servidores de alto rendimiento.	■ Menor flexibilidad al mantener una estructura. ■ La curva de aprendizaje puede ser más elevada que las de otro framework.

Cuadro 3.2: Tabla comparativa de Frameworks de Backend

3.2.2.1. Discusión

Dentro del contexto de la solución propuesta, se decidió seleccionar Express.js [14] como framework debido a su capacidad de escalar de manera efectiva. Este marco cuenta con una amplia comunidad de usuarios y muchos otros frameworks están contruidos sobre la base de Express.js, lo que lo convierte en una opción altamente compatible con otras tecnologías. Además, Express.js es compatible con numerosas bibliotecas, lo que amplía aún más su versatilidad y funcionalidad. Como motor de servidor, se utilizará Node.js, que es ampliamente reconocido y utilizado a nivel mundial.

3.2.3. Framework de Front-end

Los framework de Front-end es un producto o herramienta de software que proporciona una base para el desarrollo de la interfaz de usuario de soluciones web [15]. En el ámbito de este trabajo de título, se buscará el framework que se ajuste más a la solución propuesta.

A continuación, se muestra una tabla con las ventajas y desventajas de los frameworks más utilizados.

Framework	Ventajas	Desventajas
React	■ Agiliza la creación de interfaces, dado que está basado en componentes reutilizables. ■ Es fácil de entender, leer y depurar. ■ Posee una comunidad activa.	■ Se enfoca solamente en la interfaz de usuario. ■ Al no ser un framework, necesita la integración de otras herramientas para tareas como el manejo de estado o el enrutamiento.
Vue.js	■ Adaptable a cualquier tipo de proyecto. ■ Su sintaxis es de fácil uso.	■ La comunidad no es tan grande en comparación a otras tecnologías. ■ Riesgos de exceso de flexibilidad.

Angular	■ Permite la creación total de la aplicación. ■ Mantiene una arquitectura MVC. ■ Permite la instalación de otras dependencias.	■ Curva de aprendizaje pronunciada. ■ Aplicaciones más pesadas.
---------	--	--

Cuadro 3.3: Tabla comparativa de Frameworks de Front-end

3.2.3.1. Discusión

El framework seleccionado es React [16]. Es importante señalar que React no es un framework, sino una biblioteca. Esta elección se basa en su enfoque centrado en componentes, lo cual es fundamental para la comercialización del servicio, dado que algunos clientes pueden no requerir ambos módulos. La modularización de las vistas se simplificará gracias a este enfoque basado en componentes. Además, se elige esta herramienta porque uno de los principales requisitos de la empresa es la simplicidad de la aplicación, es decir, se prefiere tener la menor cantidad posible de vistas, para estos casos React es muy útil.

3.2.4. Bases de Datos

Las bases de datos son sistemas que almacenan datos y los organizan en una estructura lógica, junto con los metadatos necesarios para su procesamiento. Son herramientas esenciales para gestionar grandes volúmenes de información y simplificar la búsqueda de datos. Estas bases de datos se clasifican principalmente en dos categorías: relacionales y no relacionales. Las bases de datos relacionales están compuestas por múltiples tablas que se conectan entre sí mediante claves foráneas, faci-

litando la gestión y consulta de datos estructurados. Por otro lado, las bases de datos no relacionales se centran en almacenar información que no requiere una estructura definida, ofreciendo flexibilidad y escalabilidad para datos no estructurados o semi-estructurados.

A continuación, se presentará en la siguiente tabla las tecnologías más utilizadas para el almacenamiento de datos.

Base de Datos	Tipo	Ventajas	Desventajas
MySQL	Relacional	■ Rápido para la ejecución. ■ Es de código abierto.	■ No maneja de buena forma bases de datos muy grandes.
PostgreSQL	Relacional	■ Está diseñado para bases de datos de alto volumen. ■ Es confiable. ■ Gran escalabilidad ■ Buena adaptación con el hardware.	■ Lento en inserciones y actualizaciones en bases de datos pequeñas

MongoDB	No Relacional	■ Es ideal para entornos con pocos recursos. ■ Tiene una gran documentación. ■ Es un complemento perfecto para JavaScript.	■ No es una base de datos adecuada para aplicaciones con transacciones complejas. ■ No tiene Joins para consultas.
CouchDB	No Relacional	■ Escalamiento horizontal. ■ Cuenta con una API RESTful para interactuar con los datos.	■ Puede requerir una infraestructura de hardware adicional por el escalamiento horizontal. ■ La comunidad no es tan grande como la de otras herramientas.

Cuadro 3.4: Tabla comparativa de Bases de Datos

3.2.4.1. Discusión

En relación con la solución, como se detalla en la sección de arquitectura, está basada en microservicios, cada uno con su propia base de datos para mantener modularizado el sistema. Esta elección se fundamenta en la necesidad de manejar diferentes

tipos de datos, como se detallará en la sección 3.4.2.

Para la base de datos relacional, se optó por PostgreSQL [17] debido a su capacidad para adaptarse de manera óptima al hardware. Esta característica es crucial en nuestra solución, ya que buscamos garantizar la adaptabilidad a diversos entornos. Además, este trabajo describe un producto mínimo viable, con expectativas de crecimiento futuro. Por lo tanto, la promesa de escalabilidad de PostgreSQL resulta relevante.

En cuanto a las bases de datos no relacionales, se ha seleccionado MongoDB [18]. Esta elección se debe, al igual que con la base de datos relacional, al bajo costo de recursos que requiere, así como a su fácil integración con JavaScript.

3.2.5. Dependencias y Contenedores

Para implementar la solución, se requiere emplear diversas dependencias o bibliotecas específicas de Node.js. A continuación, se detallarán todas las bibliotecas utilizadas en el desarrollo de la solución, junto con una breve descripción de cada una.

- **axios**[19]: es un Cliente HTTP basado en promesas para node.js y el navegador.
- **bcrypt**[20]: es una biblioteca que permite codificar contraseñas.
- **body-parser**[21]: es una biblioteca de middleware de Node.js utilizada para analizar los datos del cuerpo de las solicitudes HTTP en aplicaciones Express.js.
- **cors**[22]: es un mecanismo de seguridad en los navegadores web que controla qué recursos pueden ser solicitados por una página web desde otro dominio. Permite a los servidores especificar quién puede acceder a sus recursos a través de encabezados HTTP.
- **dotenv**[23]: es un módulo de dependencia cero que carga las variables de entorno desde un archivo `.env` en `process.env`.

- **jest**[24]: es un framework que se enfoca en el testeo de las aplicaciones.
- **jsonwebtoken**[25]: es un estándar abierto que permite transmitir información de forma segura entre partes como un objeto JSON
- **moment**[26]: analiza, valida, manipula y muestra fechas y horas en JavaScript.
- **mongoose**[27]: es una biblioteca de modelado de objetos de MongoDB para Node.js.
- **@mui/material**[28]: es una biblioteca de componentes de interfaz de usuario para React, basada en Material-UI.
- **nodemon**[29]: monitorea los cambios que se hacen en el código y reinicia el servidor.
- **pg**[30]: permite conectarse a una base de datos PostgreSQL, enviar consultas SQL.
- **react-dom**: proporciona métodos específicos del DOM que pueden ser utilizados en el nivel más alto de la aplicación.
- **sequelize**[31]: es un ORM que permite interactuar con bases de datos relacionales utilizando objetos JavaScript en lugar de escribir consultas SQL directamente.
- **sinon**[32]: es una biblioteca de pruebas para JavaScript que ofrece funciones de espionaje, simulación y stubbing para facilitar la escritura de pruebas automatizadas confiables.
- **supertest**[33]: es una biblioteca de pruebas para Node.js que facilita la realización de pruebas de integración para aplicaciones HTTP.

Además, uno de los requisitos no funcionales es mantener toda la aplicación en contenedores. Por esta razón, la aplicación utilizará Docker [34] para cumplir con

este requisito. Docker es una herramienta que facilita el despliegue de aplicaciones en contenedores. Entre los beneficios de Docker se incluyen la optimización de recursos, la portabilidad de la aplicación y la capacidad de escalabilidad.

3.3. Diagrama de Contexto

En esta sección, se expondrá el diagrama de contexto con el propósito de establecer claramente los límites y las interacciones entre los actores y el sistema.

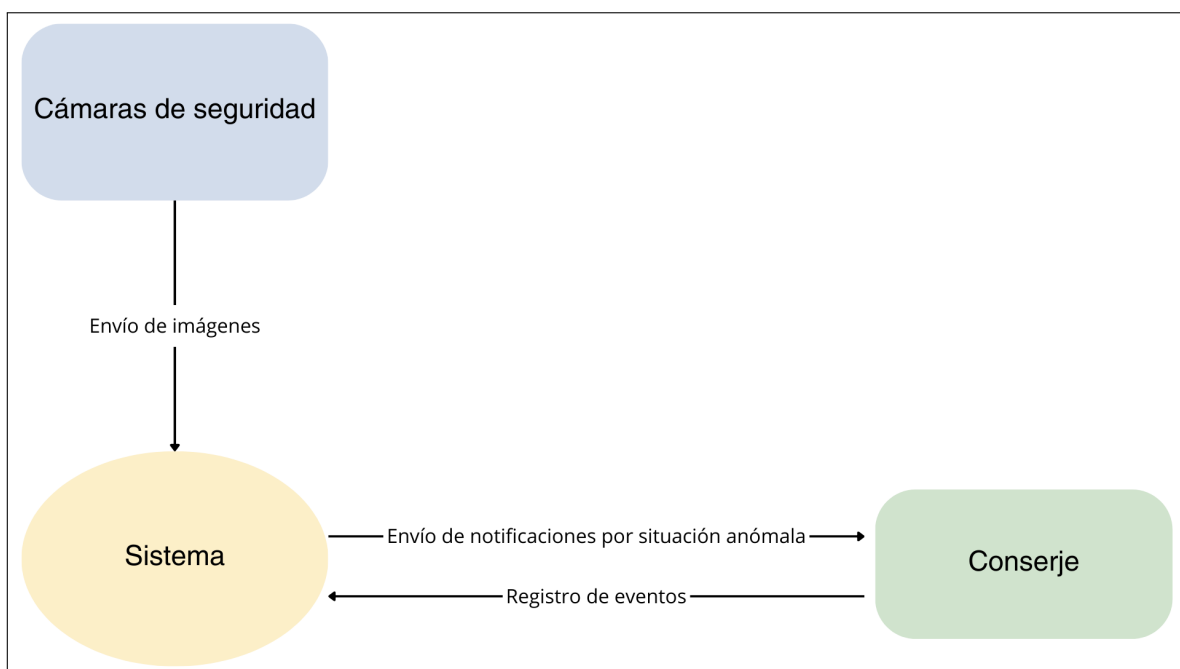


Figura 3.1: Diagrama de Contexto

Como se evidencia en la figura 3.1, el sistema debe contar con la capacidad de informar al conserje, mientras que, por otro lado, el conserje debe poseer la habilidad de registrar eventos, visitas y observaciones.

3.4. Arquitectura del sistema

En esta sección, procederemos a definir la arquitectura del sistema, empleando el modelo C4 [35], un tipo de gráfico que facilita la descomposición del sistema en contenedores y componentes.

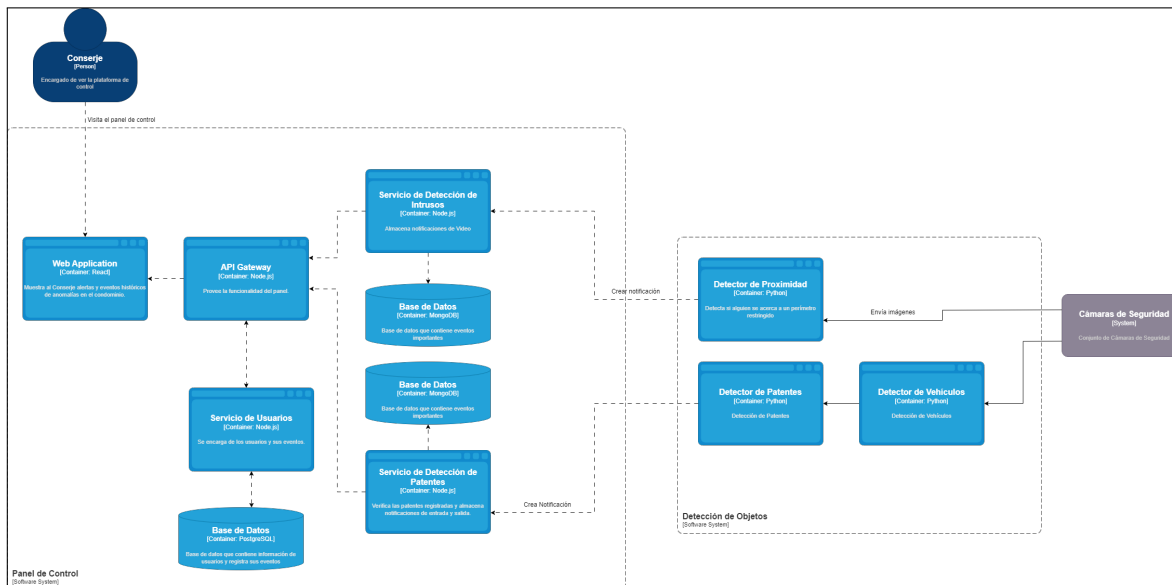


Figura 3.2: Arquitectura Propuesta

Como se puede apreciar en el diagrama de la figura 3.2, la solución propuesta se fundamenta en la adopción de microservicios, en cumplimiento con la solicitud expresa de la empresa, que prescinde de una aplicación monolítica. La comercialización del producto se adaptará a las preferencias del cliente, pudiendo optar por la detección de intrusos, la detección de patentes o ambos detectores.

3.4.1. Servicios Propuestos

A continuación, se explicará el alcance de cada servicio que conforma la arquitectura propuesta en el modelo C4.

- **API-Intrusos:** Este servicio se encarga de gestionar las notificaciones generadas por el detector de intrusos. Cuando se detecta alguna transgresión, el detector

registra la notificación a través de esta API.

- **API-Patentes:** Este servicio se ocupa de registrar los eventos relacionados con la detección de patentes y de mantener un registro de las visitas.
- **API-Usuario:** Este servicio se encarga de administrar a los usuarios que acceden a la plataforma. Además, posibilita la incorporación de eventos específicos que ocurran durante el turno del conserje.
- **API-Gateway:** Este servicio se encarga de comunicar y centralizar la información, preparándola para su posterior envío al frontend.
- **Web Application:** Esta aplicación es la parte visual de la plataforma e interactúa con el usuario.

3.4.2. Bases de Datos

En relación con las bases de datos, se pueden identificar tres en el diagrama, dado que cada microservicio posee la suya propia. Esta distribución se fundamenta en la necesidad de mantener la modularidad en la propuesta de solución. De esta manera, resultará más sencillo eliminar algún elemento en caso de requerirlo, sin afectar la integridad del sistema.

3.4.2.1. Base de datos de API-Intrusos

Para el servicio de detección de intrusos, se ha optado por una base de datos no relacional, dado que los eventos no mantienen relaciones entre sí, permitiendo su gestión de manera más eficiente como una colección. A continuación, se detalla los atributos del documento almacenado como **notification**.

Atributo	Tipo	Requerido	Descripción
_id	ObjectType	Si	Identificador del documento.
alert_type	String	Si	Tipo de alerta.
camera	String	Si	Cámara que gatilla la alerta.
severity	String	False	Gravedad de la alerta.
published	Boolean	Si	Si fue publicada la notificación o no.
image_url	String	Si	Dirección de almacenamiento de la imagen capturada.
date	Date	Si	Fecha y hora de cuando se registró el evento.
description	String	No	Descripción que entrega el conserje a lo que ve en el evento.
id_event	String	Si	UUID que entrega el detector para identificar el evento como único.

Cuadro 3.5: Documento de Colección Notifications

3.4.2.2. Base de datos de API-Patentes

Se ha seleccionado una base de datos no relacional para el servicio de detección de patentes, con el fin de gestionar las notificaciones, residentes y visitantes. A continuación, se describirán los atributos de los documentos de las tres colecciones.

Atributo	Tipo	Requerido	Descripción
_id	ObjectType	Si	Identificador del documento
alert_type	String	Si	Tipo de alerta.
license_plate	String	Si	Matrícula del vehículo indentificada.
published	Boolean	Si	Si fue publicada la notificación o no.
image_url	String	Si	Dirección de almacenamiento de la imagen capturada.
date	Date	Si	Fecha y hora de cuando se registró el ingreso del vehículo.
description	String	No	Descripción de la alerta.
resolution	String	Si	Resolución de la alerta, si fue autorizado para ingresar o es un vehículo desconocido.

Cuadro 3.6: Documento de Colección Notifications

Atributo	Tipo	Requerido	Descripción
_id	ObjectType	Si	Identificador del documento
license_plate	String	Si	Matrícula del vehículo.
id_house	String	Si	Número de casa.
owner	String	Si	Nombre del dueño del vehículo.

Cuadro 3.7: Documento de Colección Residents

Atributo	Tipo	Requerido	Descripción
_id	ObjectType	Si	Identificador del documento
license_plate	String	Si	Matrícula del vehículo.
id_house	String	Si	Número de casa.
driver_name	String	Si	Nombre del conductor del vehículo.
driver_dni	String	Si	RUT del conductor del vehículo
start_date	Date	Si	Fecha y hora en el que se registra la visita
end_date	Date	Si	Fecha y hora límite para el acceso libre al condominio.

Cuadro 3.8: Documento de Colección Visitors

3.4.2.3. Base de datos de API-Usuarios

Se ha elegido una base de datos relacional para este servicio debido a las estrechas relaciones entre las entidades. La figura 3.3 ilustra las dos entidades relacionadas mediante un id_usuario. La tabla de usuarios guarda todos los datos del trabajador con acceso a la plataforma, mientras que en la otra tabla se registran observaciones o eventos específicos que tienen lugar durante su turno laboral.

En cuanto a los objetivos de este trabajo, el modelo actual depende únicamente de dos entidades para su funcionamiento óptimo, pero la empresa planea expandirlo con el desarrollo de una aplicación de administración que incluirá nuevas entidades para este modelo.

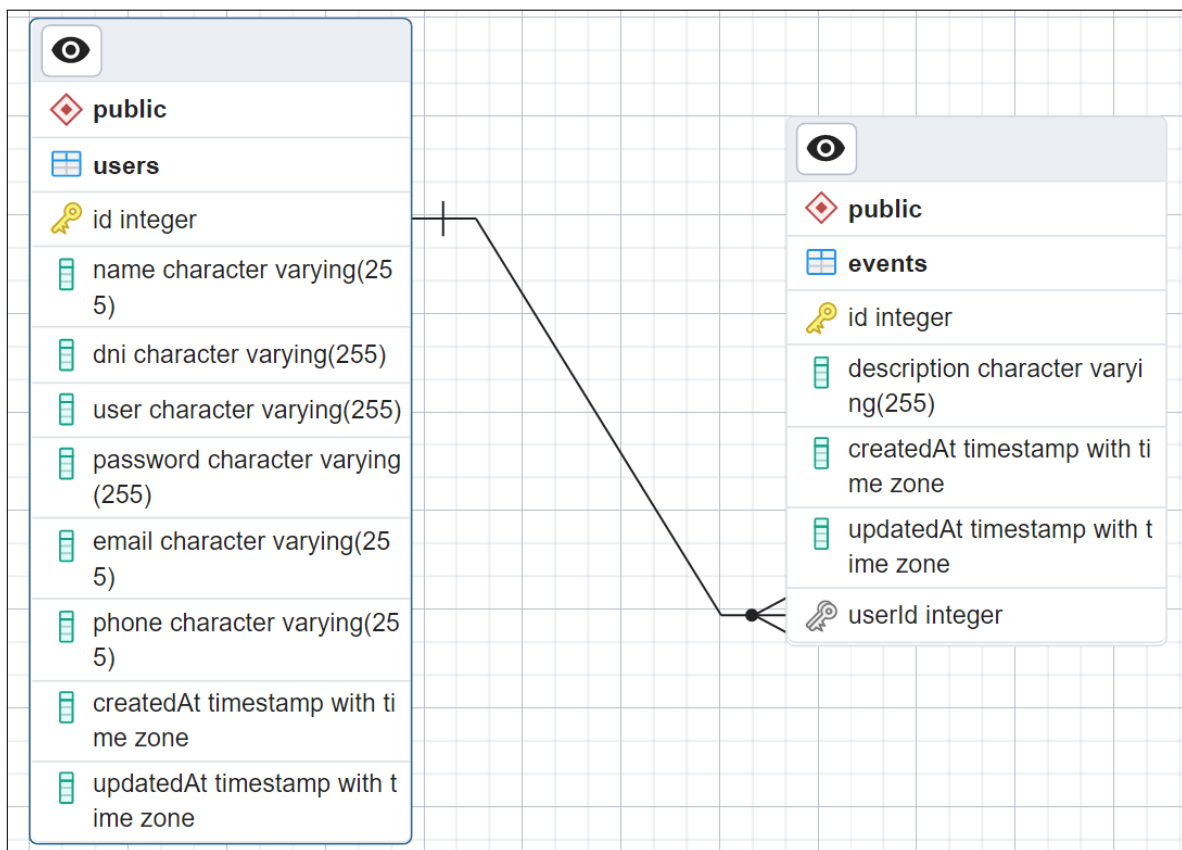


Figura 3.3: Modelo de Dominio de Base de Datos API-Usuarios.

Capítulo 4

Resultados

En este capítulo, se expondrán las respuestas y documentación definitiva de cada microservicio, junto con las vistas que componen el frontend de la propuesta de solución.

Con el fin de facilitar la lectura, se dividirá esta sección en dos partes: el back-end, que contendrá las documentaciones de los microservicios, y la sección front-end, que abordará las vistas de la plataforma.

4.1. Back-end

El backend de la solución propuesta se compone principalmente de cuatro aplicaciones. A continuación, se detallará cada servicio y se demostrarán las funcionalidades a nivel local mediante la herramienta Postman [36]. Además, se incluirán los códigos HTTP [37] de respuesta correspondientes en cada endpoint.

4.1.1. API-Usuarios

4.1.1.1. Obtener todos los usuarios de la aplicación

Este endpoint tiene como objetivo recuperar la información de todos los usuarios registrados para su autenticación. Utiliza el método HTTP **GET** y se identifica con la ruta *api-users/users*. A continuación, se detallarán las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Devuelve el resultado y un array con todos los usuarios registrados.
204	No Content	Devuelve el resultado y un array vacío
500	Internal Server Error	Error al obtener usuarios.

Cuadro 4.1: Respuestas del sistema getAllUsers de API-Usuarios.

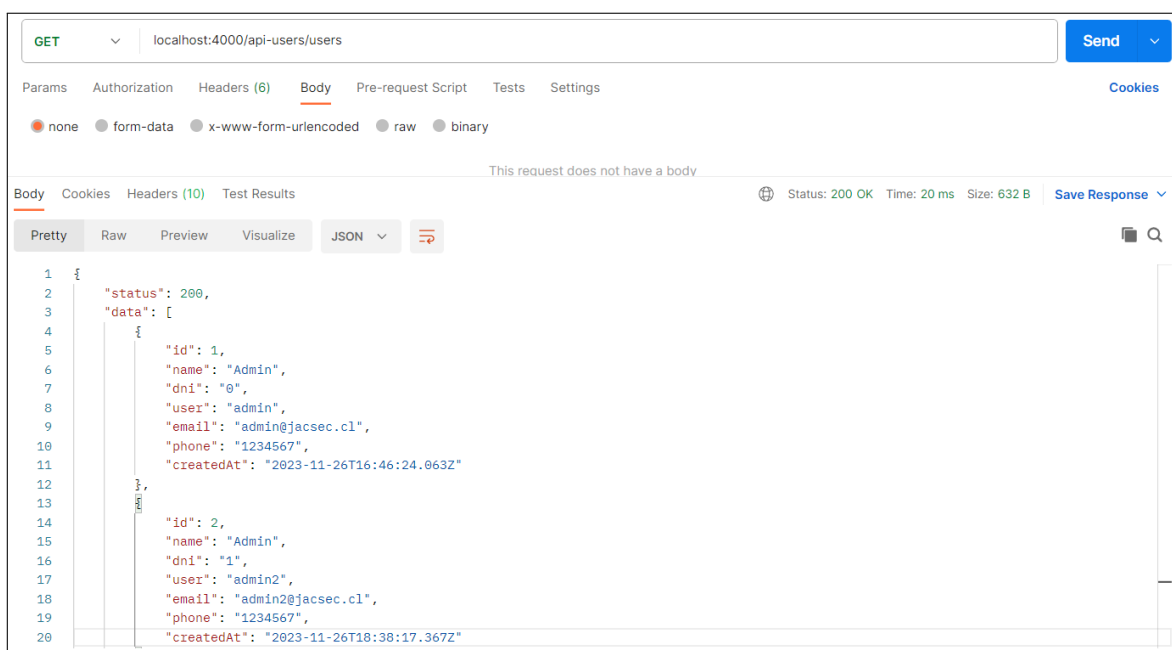


Figura 4.1: Respuesta Correcta de getAllUsers de API-Usuarios.

4.1.1.2. Obtener los eventos por Fecha y Usuario

El propósito de este endpoint es recuperar todos los eventos asociados a un usuario en una fecha específica. Se emplea el método HTTP **GET**, y su ruta se define como *api-user/user/{id_user}/events?date={date}*, donde los parámetros **id_user** y **date** representan el identificador del usuario y la fecha de filtrado, respectivamente. A continuación, se describen las posibles respuestas del sistema, cada una con su correspondiente código.

Código HTTP	Respuesta	Descripción
200	Ok	Devuelve el resultado y un array con todos los eventos registrados.
204	No Content	Devuelve el resultado y un array vacío.
400	Bad Request	Error al realizar solicitud. Verifique los parámetros.
500	Internal Server Error	Error al obtener los eventos.

Cuadro 4.2: Respuestas del sistema `getEventsByDateAndUser` de API-Usuarios.

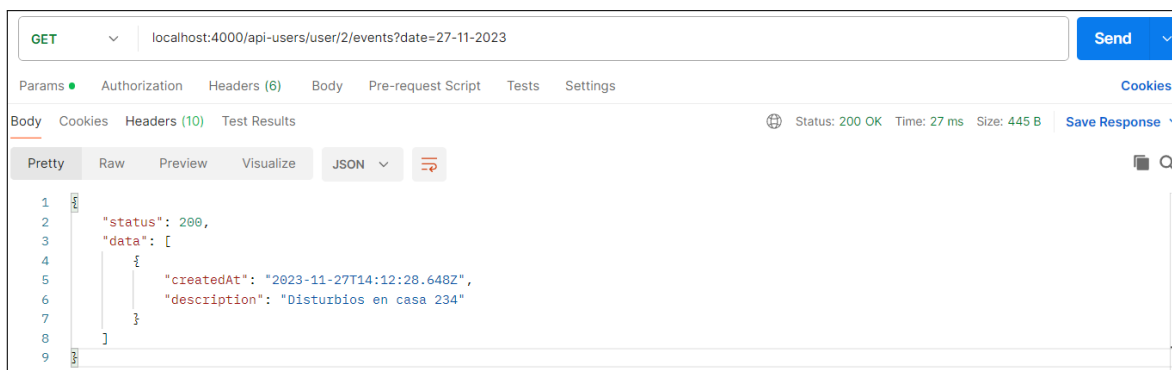


Figura 4.2: Respuesta Correcta de `getEventsByDateAndUser` de API-Usuarios.

4.1.1.3. Registrar eventos del usuario

Este endpoint tiene como objetivo facilitar el registro por parte del usuario de cualquier evento anómalo que pueda ocurrir durante su turno. Utiliza el método HTTP **POST** y su ruta se especifica como *api-user/user/{id_user}/event*, donde el parámetro **id_user** corresponde al identificador del usuario. Además, se debe enviar el campo **description** con la descripción del evento a registrar. A continuación, se describen las posibles respuestas del sistema, cada una con su correspondiente código.

Código HTTP	Respuesta	Descripción
201	Created	Devuelve el resultado y un objeto con los datos del evento agregado.
400	Bad Request	Error al realizar solicitud. Verifique los parámetros.
500	Internal Server Error	Error al crear el evento.

Cuadro 4.3: Respuestas del sistema createEvent de API-Usuarios.

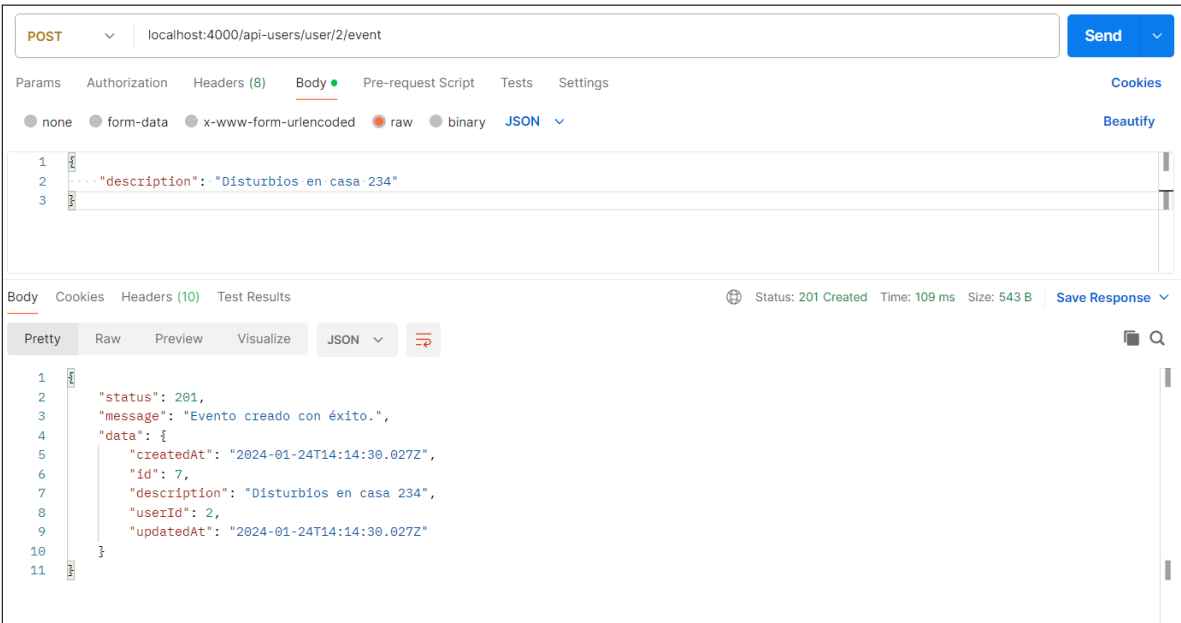


Figura 4.3: Respuesta Correcta de createEvent de API-Usuarios.

4.1.1.4. Creación de nuevos usuarios

El propósito de este endpoint es simplificar el proceso de registro de nuevos usuarios. Se utiliza el método HTTP **POST**, y su ruta se especifica como *api-user/user*. Para este método, se deben enviar los siguientes datos: **name**, **dni**, **user**, **password**, **email**, **phone**. A continuación, se detallan las posibles respuestas del sistema, cada una con su correspondiente código.

Código HTTP	Respuesta	Descripción
201	Created	Usuario creado con éxito.
400	Bad Request	Error al realizar solicitud. Verifique parámetros.
409	Conflict	El usuario ya está registrado.
500	Internal Server Error	Error al crear el usuario.

Cuadro 4.4: Respuestas del sistema createUser de API-Usuarios.

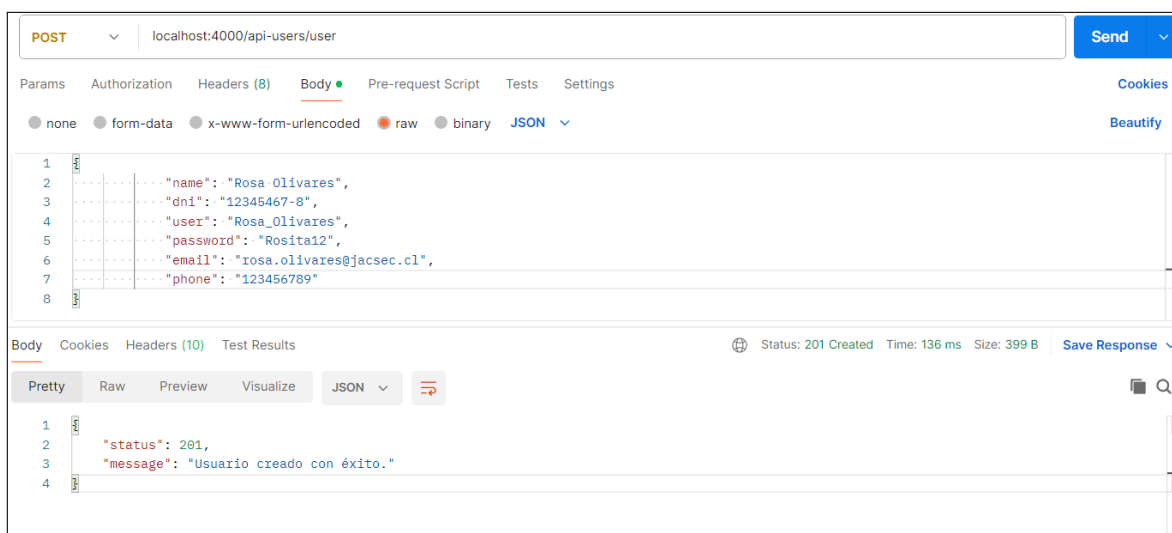


Figura 4.4: Respuesta Correcta de createUser de API-Usuarios.

4.1.1.5. Login de la aplicación

Este endpoint tiene como finalidad autenticar al usuario en el sistema. Se emplea el método HTTP **POST**, y su ruta se define como *api-user/login*. Para utilizar este método, es necesario enviar el nombre de usuario (username) y la contraseña (password). A continuación, se describen las posibles respuestas del sistema, cada una con su código correspondiente.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica un exitoso inicio de sesión mediante un mensaje, acompañado de un objeto que contiene la identificación del usuario.
400	Bad Request	Error al realizar solicitud. Verifique parámetros.
401	Unauthorized	Credenciales inválidas.
500	Internal Server Error	Error al iniciar sesión.

Cuadro 4.5: Respuestas del sistema Login de API-Usuarios.

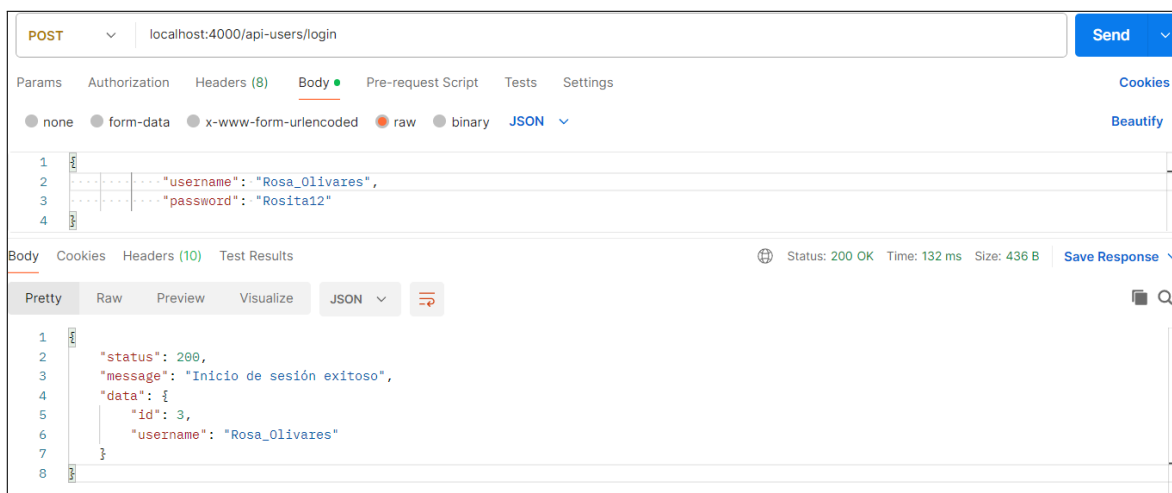


Figura 4.5: Respuesta Correcta de Login de API-Usuarios.

4.1.2. API-Intrusos

4.1.2.1. Obtención de todas las notificaciones

El propósito de este endpoint es recuperar todas las notificaciones almacenadas en la base de datos. Se emplea el método HTTP **GET**, y su ruta se define como *api-intrusos/notifications-no-filtered*. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con todas las notificaciones.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
500	Internal Server Error	Error al obtener todas las notificaciones.

Cuadro 4.6: Respuestas del sistema getAllNotifications de API-Intrusos.

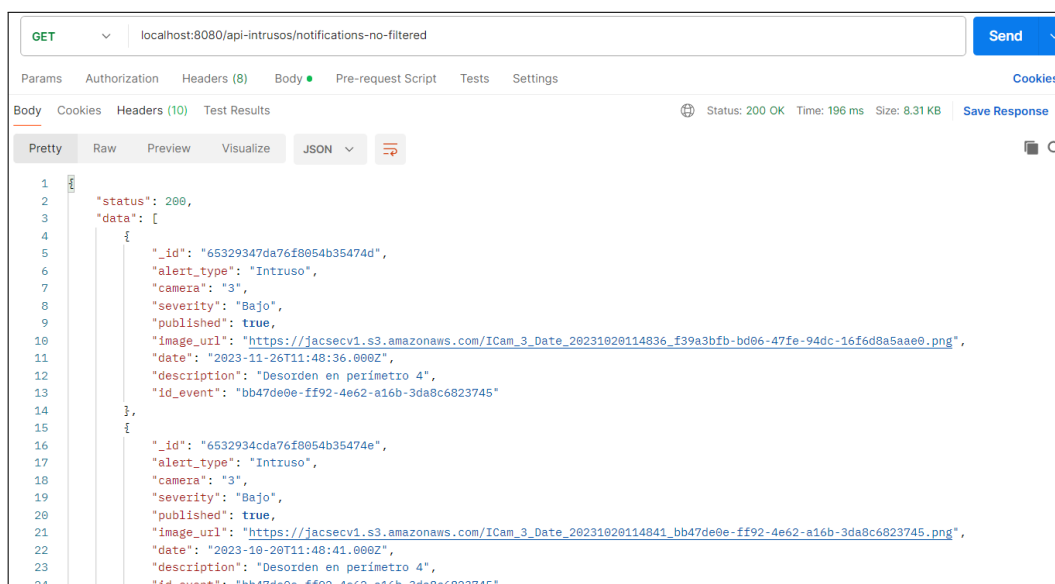


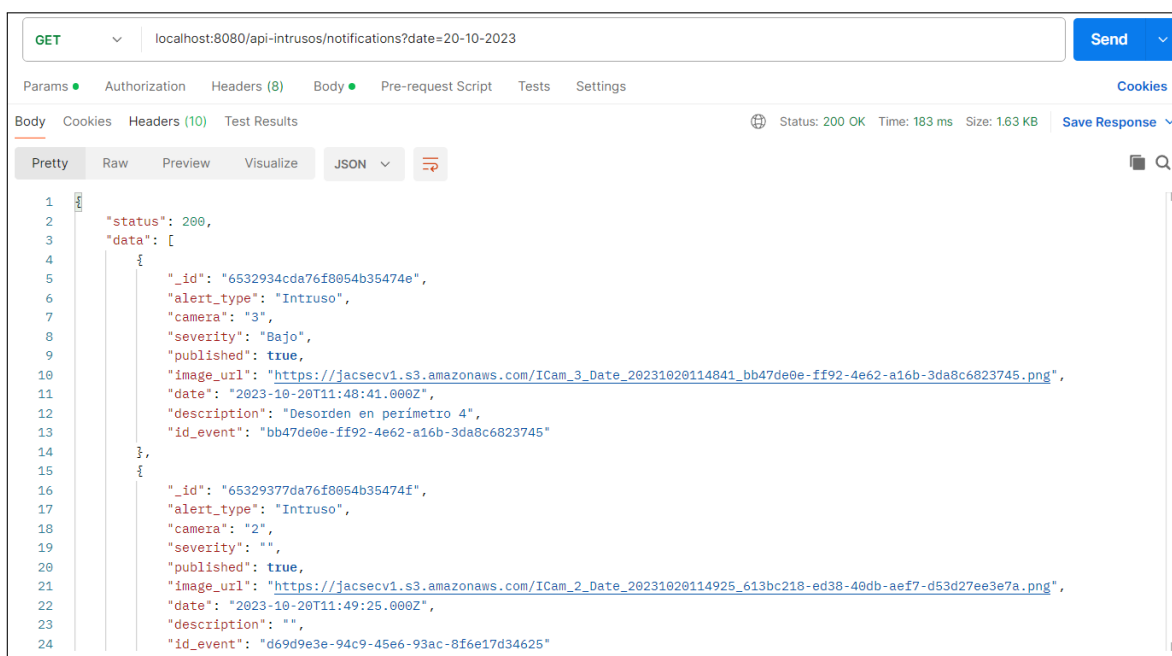
Figura 4.6: Respuesta Correcta de getAllNotifications de API-Intrusos.

4.1.2.2. Obtención de las notificaciones por fecha

El objetivo de este endpoint es recuperar todas las notificaciones filtradas por una fecha específica. Se utiliza el método HTTP **GET**, y su ruta se define como *api-intrusos/notificaciones?date={date}*. A continuación, se detallan las posibles respuestas del sistema, cada una con su correspondiente código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con todas las notificaciones.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
400	Bad Request	Error al realizar solicitud. Verifique parámetros.
500	Internal Server Error	Error al obtener todas las notificaciones.

Cuadro 4.7: Respuestas del sistema getAllNotificationsByDate de API-Intrusos.



```
GET localhost:8080/api-intrusos/notificaciones?date=20-10-2023
Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies
Body Cookies Headers (10) Test Results
Status: 200 OK Time: 183 ms Size: 1.63 KB Save Response

Pretty Raw Preview Visualize JSON

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

{
  "status": 200,
  "data": [
    {
      "_id": "6532934cda76f8054b35474e",
      "alert_type": "Intruso",
      "camera": "3",
      "severity": "Bajo",
      "published": true,
      "image_url": "https://jacsecv1.s3.amazonaws.com/ICam_3_Date_20231020114841_bb47de0e-ff92-4e62-a16b-3da8c6823745.png",
      "date": "2023-10-20T11:48:41.000Z",
      "description": "Desorden en perimetro 4",
      "id_event": "bb47de0e-ff92-4e62-a16b-3da8c6823745"
    },
    {
      "_id": "65329377da76f8054b35474f",
      "alert_type": "Intruso",
      "camera": "2",
      "severity": "",
      "published": true,
      "image_url": "https://jacsecv1.s3.amazonaws.com/ICam_2_Date_20231020114925_613bc218-ed38-40db-aef7-d53d27ee3e7a.png",
      "date": "2023-10-20T11:49:25.000Z",
      "description": "",
      "id_event": "d69d9e3e-94c9-45e6-93ac-8f6e17d34625"
    }
  ]
}
```

Figura 4.7: Respuesta Correcta de getAllNotificationsByDate de API-Intrusos.

4.1.2.3. Obtención de la última notificación

El propósito de este endpoint es recuperar la última notificación registrada en la base de datos. Se emplea el método HTTP **GET**, y su ruta se especifica como *api-intrusos/notification/last*. A continuación, se describen las posibles respuestas del sistema, cada una con su correspondiente código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con la última notificación.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
500	Internal Server Error	Error al obtener la última notificación.

Cuadro 4.8: Respuestas del sistema getLastNotification de API-Intrusos.

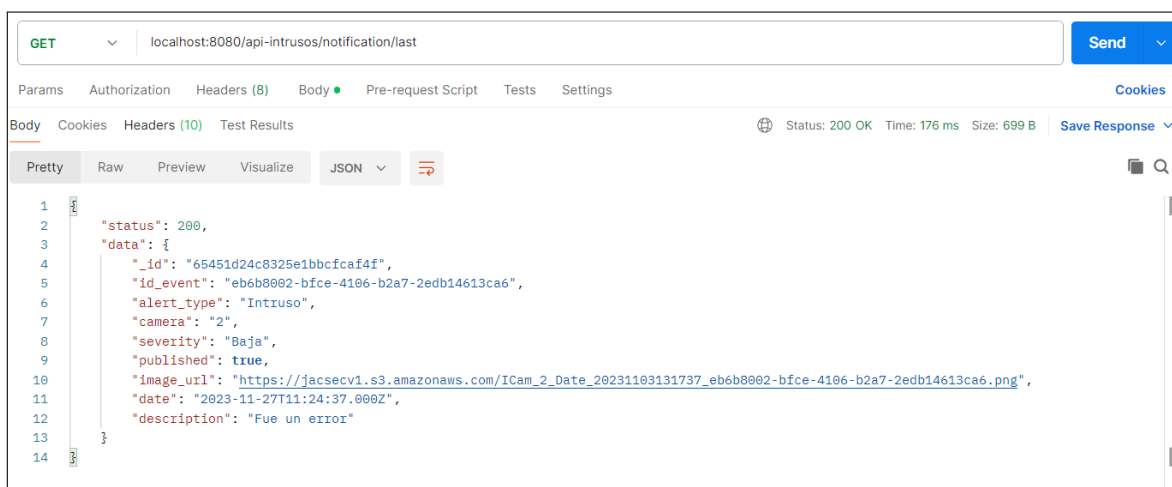


Figura 4.8: Respuesta Correcta de getLastNotification de API-Intrusos.

4.1.2.4. Obtener la última notificación no publicada

El objetivo de este endpoint es recuperar la última notificación no publicada registrada en la base de datos. Se utiliza el método HTTP **GET**, y su ruta se define como *api-intrusos/notification/last-no-published*. A continuación, se detallan las posibles respuestas del sistema, cada una con su correspondiente código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con las notificaciones no publicadas.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
500	Internal Server Error	Error al obtener las notificaciones no publicadas.

Cuadro 4.9: Respuestas del sistema getLastNoPublishedNotification de API-Intrusos.

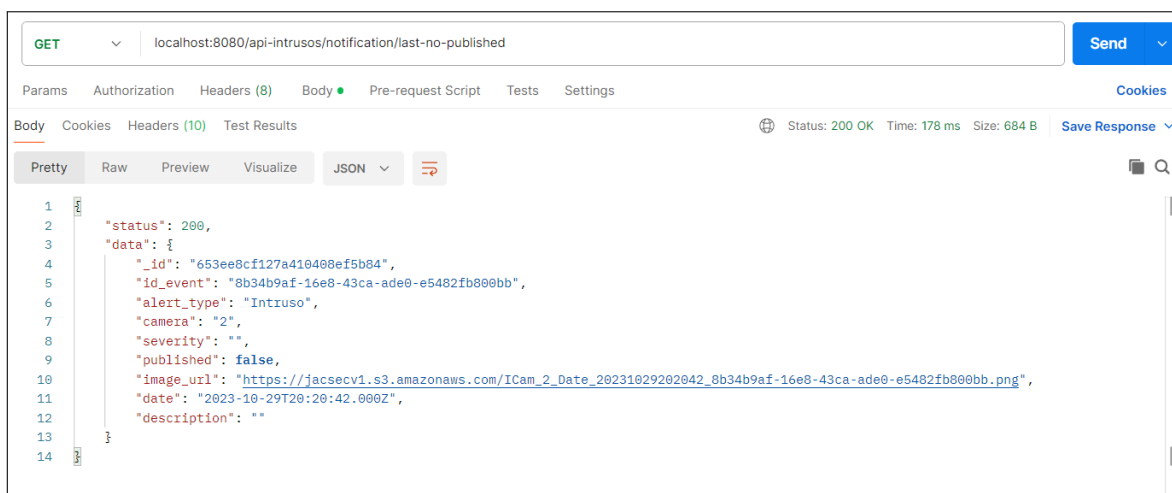


Figura 4.9: Respuesta Correcta de getLastNoPublishedNotification de API-Intrusos

4.1.2.5. Creación de nueva notificación

Este endpoint tiene como objetivo generar una notificación a partir de un evento, siendo utilizado por el detector para crear la notificación correspondiente. Se utiliza el método HTTP **POST**, y su ruta se especifica como *api-intrusos/event*. Para utilizar este método, es necesario enviar los siguientes datos: **id_event**, **imagen_url**, **date**, **camera**. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
201	Created	Se notifica que la creación de la notificación fue exitosa.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros.
500	Internal Server Error	Error al registrar la notificación.

Cuadro 4.10: Respuestas del sistema createNotification de API-Intrusos.

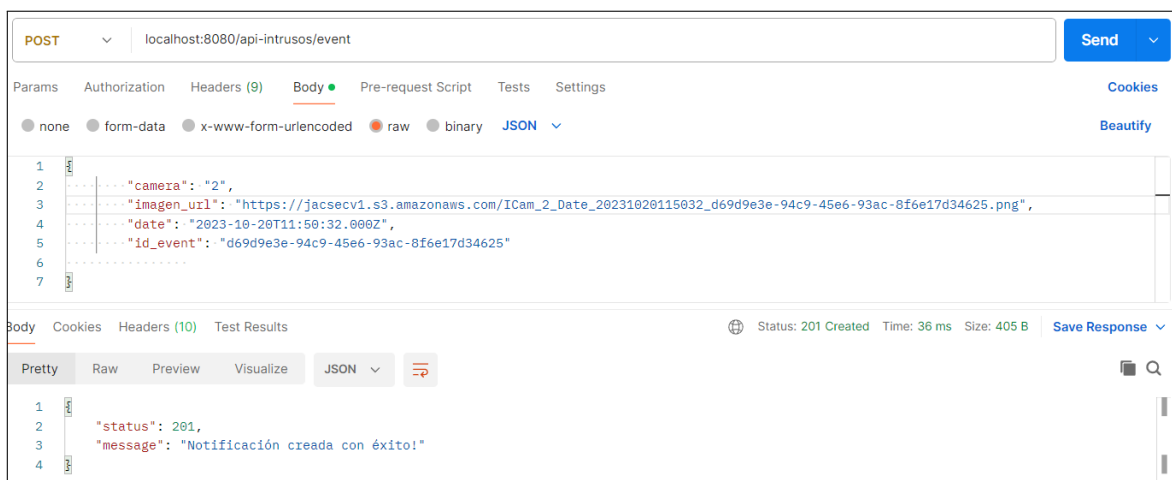


Figura 4.10: Respuesta Correcta de createNotification de API-Intrusos

4.1.2.6. Actualización de Estado Publicado

El propósito de este endpoint es actualizar el estado de publicación de la notificación. Se utiliza el método HTTP **PUT**, y su ruta se especifica como *api-intrusos/events/{id_event}/notification*, donde **id_event** representa el identificador registrado por el detector del evento. A continuación, se detallan las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	La notificación ha sido actualizada con éxito.
400	Bad Request	Error al realizar solicitud. Verifique parámetros.
500	Internal Server Error	Error al actualizar la notificación.

Cuadro 4.11: Respuestas del sistema `updatePublishedStateNotification` de API-Intrusos.

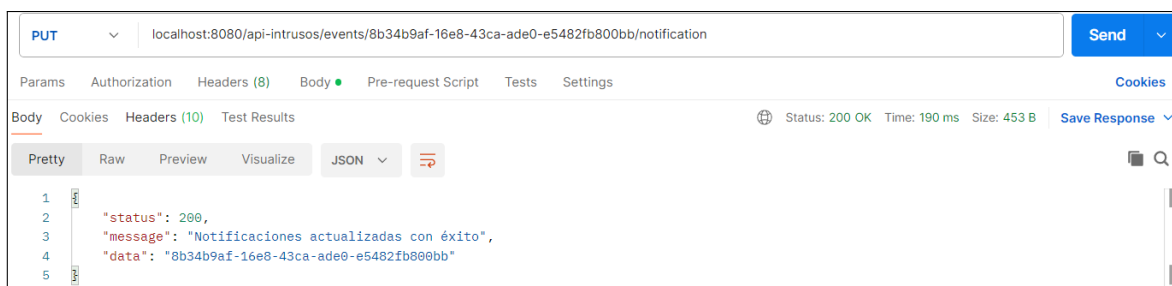


Figura 4.11: Respuesta Correcta de `updatePublishedStateNotification` de API-Intrusos.

4.1.2.7. Actualizar descripción y gravedad

El objetivo de este endpoint es actualizar la descripción y gravedad de la notificación. Se emplea el método HTTP **PUT**, y su ruta se define como *api-intrusos/events/{id_event}/description*, donde **id_event** corresponde al identificador registrado por el detector del evento. Además, se deben incluir los campos **description** y **severity**. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	La notificación ha sido actualizada con éxito.
400	Bad Request	Error al realizar solicitud. Verifique parámetros.
500	Internal Server Error	Error al actualizar la notificación.

Cuadro 4.12: Respuestas del sistema updateNotificationData de API-Intrusos.

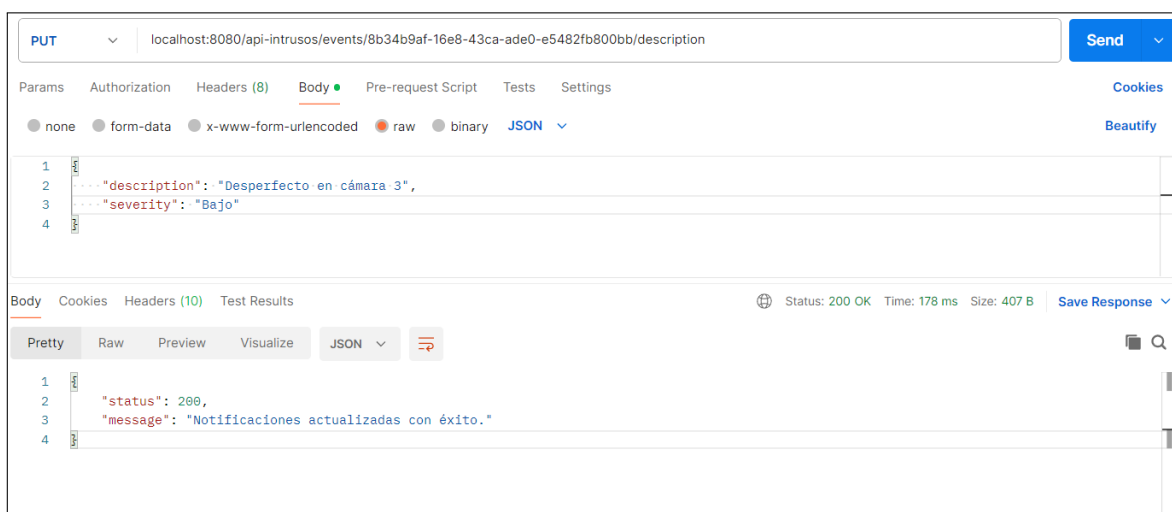


Figura 4.12: Respuesta Correcta de updateNotificationData de API-Intrusos.

4.1.3. API-Patentes

4.1.3.1. Buscar las patentes registradas

El propósito de este endpoint es verificar la existencia de un registro del vehículo detectado dentro de las colecciones Residents y Visitors. Este endpoint es fundamental para el detector de patentes, ya que determina la resolución de la notificación del vehículo registrado. Si el vehículo está registrado, se generará una notificación con una resolución positiva. En caso contrario, se activará una notificación como alerta de vehículo desconocido. En este endpoint se emplea el método HTTP **POST**, y su ruta se define como *api-patentes/license-plate*, se debe enviar los siguientes parámetros para que la respuesta sea exitosa: **license_plate** y **date**. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con el registro encontrado.
204	No Content	Se notifica que la solicitud fue exitosa, pero no existen registros.
400	Bad Request	Faltan parámetros para hacer la solicitud.
500	Internal Server Error	Hubo un error al hacer la consulta.

Cuadro 4.13: Respuestas del sistema searchLicensePlate de API-Patentes.

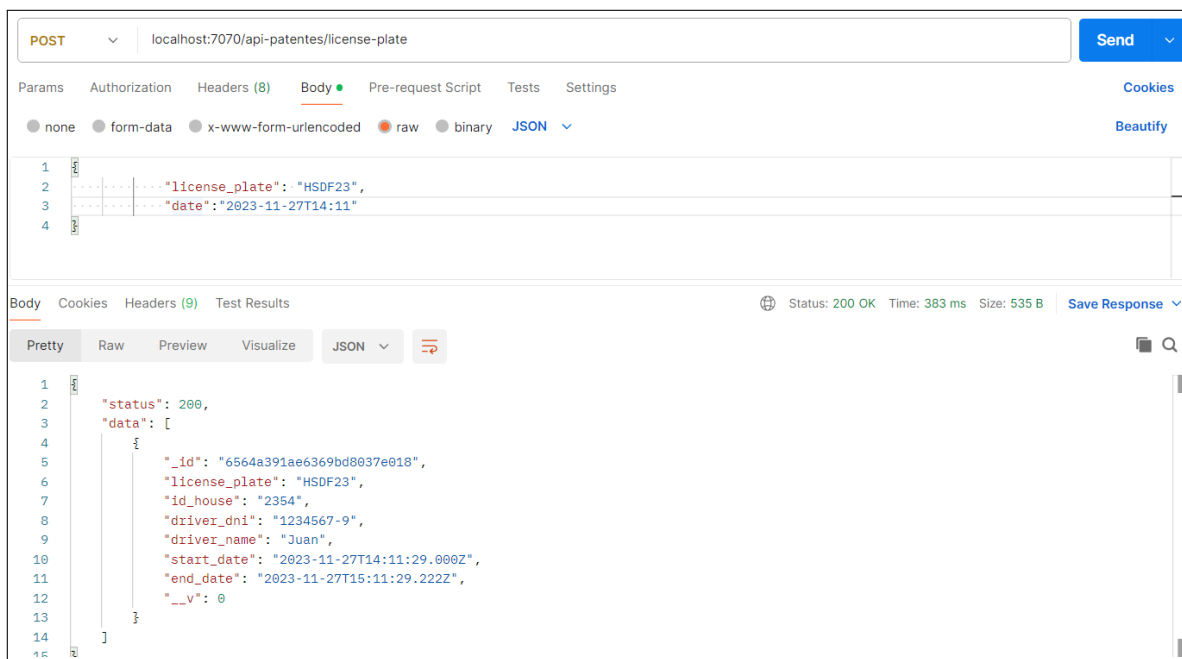


Figura 4.13: Respuesta Correcta de searchLicensePlate de API-Patentes.

4.1.3.2. Obtener todas las notificaciones

El objetivo de este endpoint es obtener todas las notificaciones. Se emplea el método HTTP **GET**, y su ruta se define como *api-patentes/notifications-no-filtered*". A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con todas las notificaciones.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
500	Internal Server Error	Error al obtener todas las notificaciones.

Cuadro 4.14: Respuestas del sistema getAllNotifications de API-Patentes.

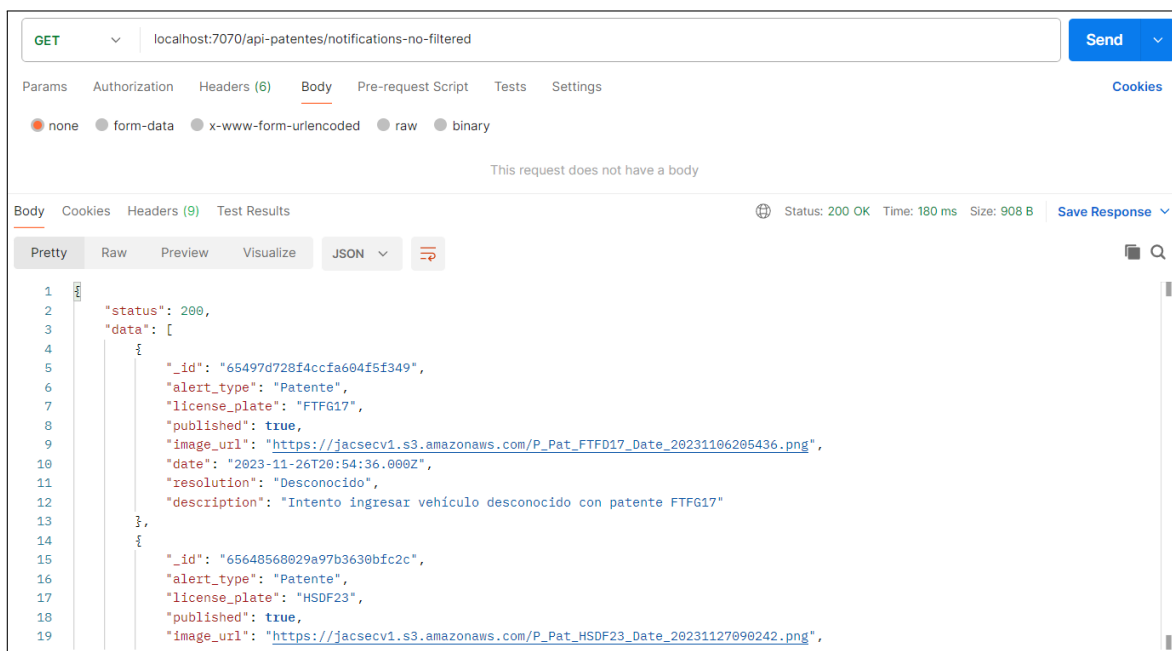


Figura 4.14: Respuesta Correcta de getAllNotifications de API-Patentes.

4.1.3.3. Obtener todas las notificaciones de un determinado día

El propósito de este endpoint es recuperar todas las notificaciones correspondientes a un día específico. Se utiliza el método HTTP **GET**, y su ruta se especifica como *api-patentes/notifications?date={date}*, donde el parámetro **date** indica la fecha de filtrado. A continuación, se detallan las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con todas las notificaciones.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
500	Internal Server Error	Error al obtener todas las notificaciones.

Cuadro 4.15: Respuestas del sistema getAllNotificationsByDate de API-Patentes.

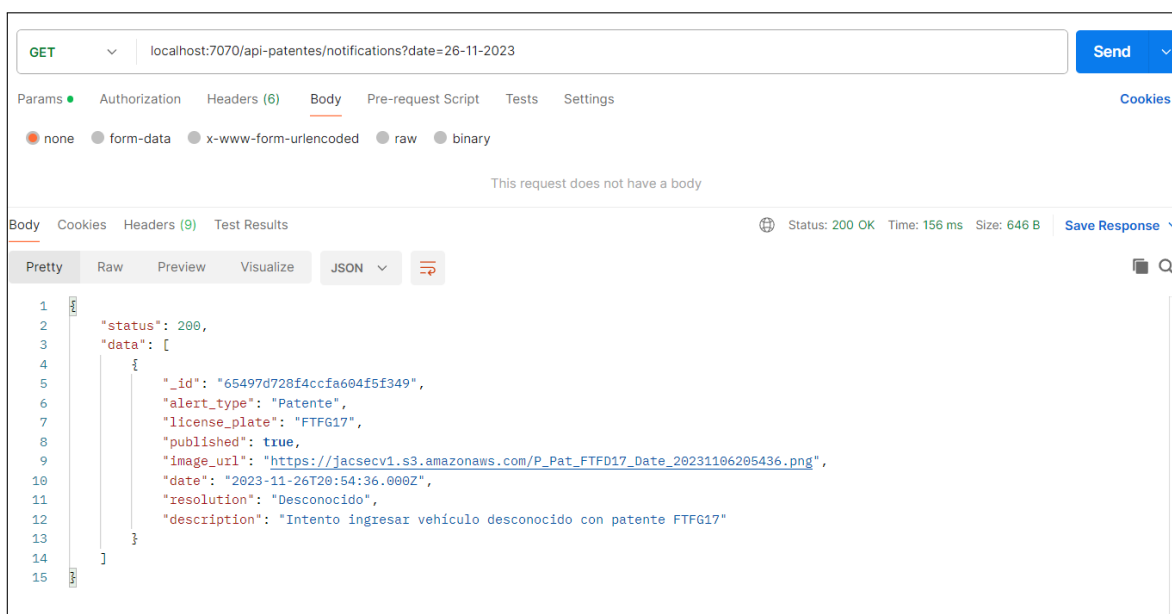


Figura 4.15: Respuesta Correcta de getAllNotificationsByDate de API-Patentes.

4.1.3.4. Obtener última notificación

El objetivo de este endpoint es recuperar la última notificación. Se utiliza el método HTTP **GET**, y su ruta se especifica como *api-patentes/notification/last*. A continuación, se detallan las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con la última notificación.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
500	Internal Server Error	Error al obtener la última notificación.

Cuadro 4.16: Respuestas del sistema getLastNotification de API-Patentes.

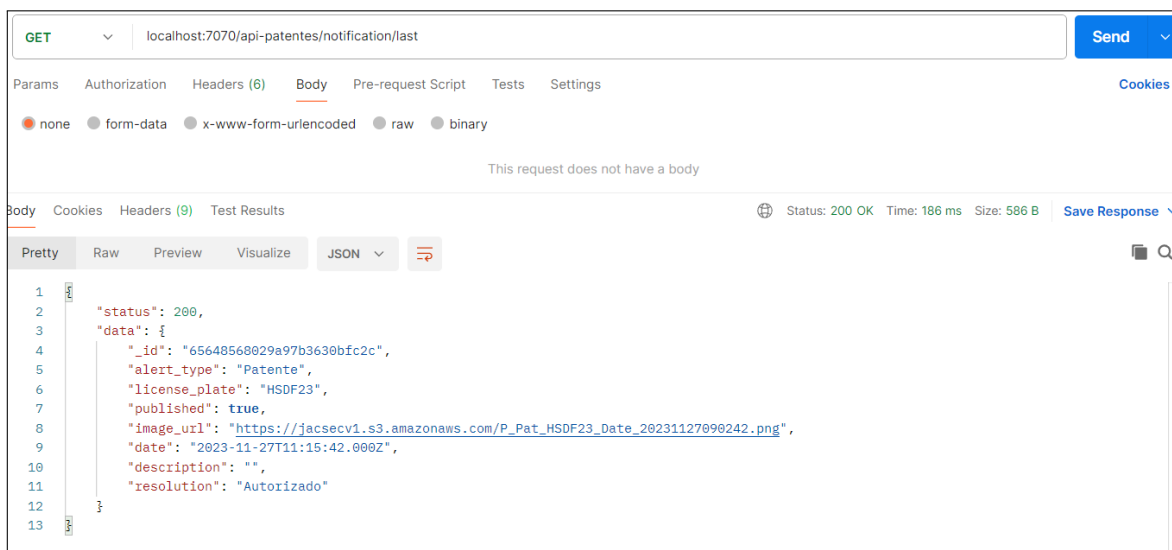


Figura 4.16: Respuesta Correcta de getLastNotification de API-Patentes.

4.1.3.5. Obtener la última publicación no publicada

El propósito de este endpoint es recuperar la última notificación no publicada. Se emplea el método HTTP **GET**, y su ruta se define como *api-patentes/notification/last-no-published*. A continuación, se detallan las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con las notificaciones no publicadas.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
500	Internal Server Error	Error al obtener la última notificación.

Cuadro 4.17: Respuestas del sistema `getLastNoPublishedNotification` de API-Patentes.

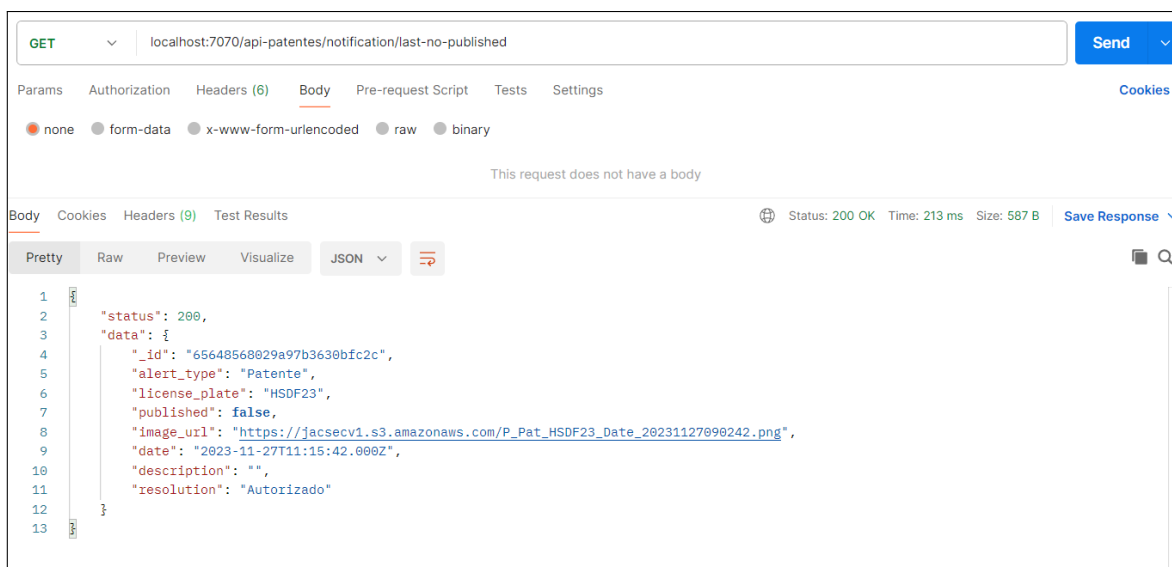


Figura 4.17: Respuesta Correcta de `getLastNoPublishedNotification` de API-Patentes.

4.1.3.6. Creación de nueva notificación

El propósito de este endpoint es registrar una nueva notificación en el sistema. Se utiliza el método HTTP **POST**, y su ruta se especifica como *api-patentes/notification*, se deben enviar como parámetros **image_url**, **license_plate**, **resolution**, **description** y **date** para que la respuesta sea exitosa. A continuación, se presentan las posibles respuestas del sistema.

Código HTTP	Respuesta	Descripción
201	Created	Se notifica que la creación de la notificación fue exitosa.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros.
500	Internal Server Error	Error al registrar la notificación.

Cuadro 4.18: Respuestas del sistema createNotification de API-Patentes.

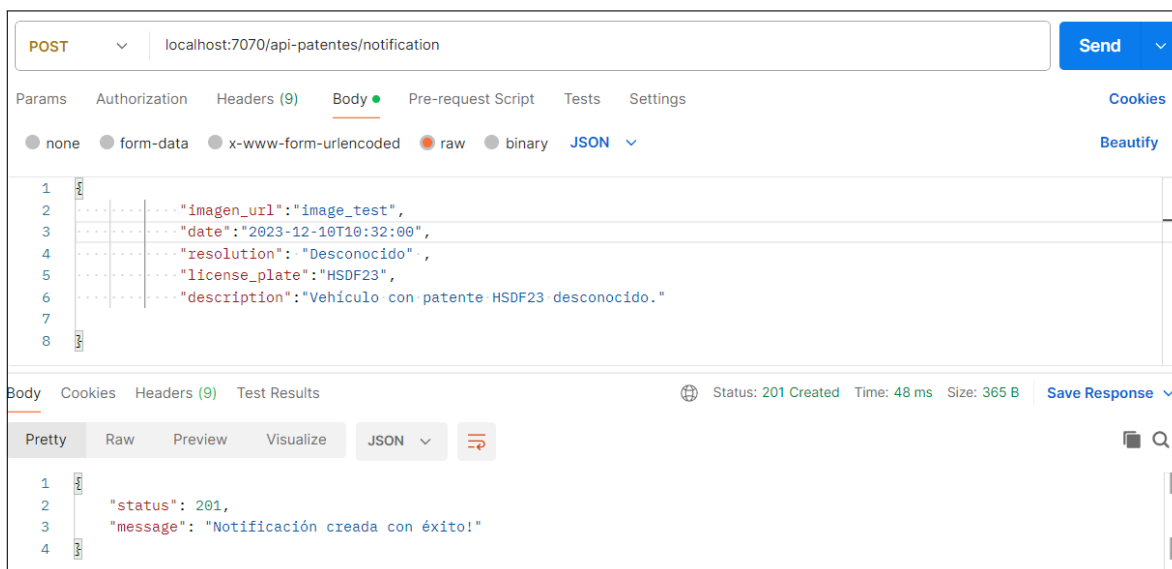


Figura 4.18: Respuesta Correcta de createNotification de API-Patentes.

4.1.3.7. Creación de nuevo visitante

El objetivo de este endpoint es registrar una nueva visita. Se utiliza el método HTTP **POST**, y su ruta se especifica como *api-patentes/visitor*, se debe enviar los siguientes parámetros para que la respuesta sea exitosa: **license_plate**, **id_house**, **driver_dni** y **driver_name**. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
201	Created	Se notifica que la creación de visita fue exitosa.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros.
500	Internal Server Error	Error al registrar una visita.

Cuadro 4.19: Respuestas del sistema createVisitor de API-Patentes.

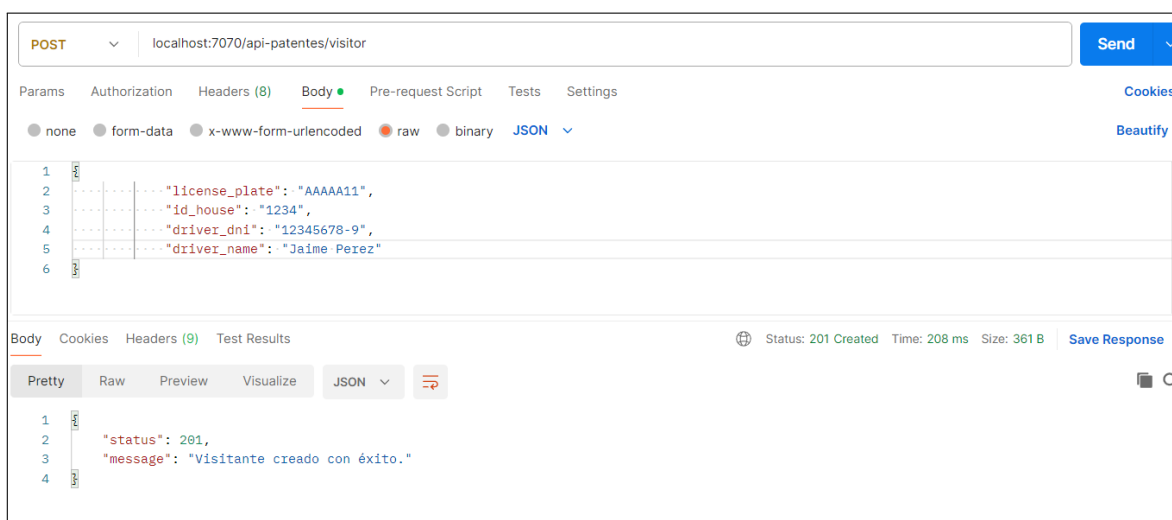


Figura 4.19: Respuesta Correcta de createVisitor de API-Patentes.

4.1.3.8. Creación de nuevo Residente

El objetivo de este endpoint es registrar un nuevo residente. Se utiliza el método HTTP **POST**, y su ruta se especifica como *api-patentes/resident* se debe enviar los siguientes parámetros para que la respuesta sea exitosa: **license_plate**, **id_house** y **owner**. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
201	Created	Se notifica que la creación de residente fue exitosa.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros.
500	Internal Server Error	Error al registrar un residente.

Cuadro 4.20: Respuestas del sistema createResident de API-Patentes.

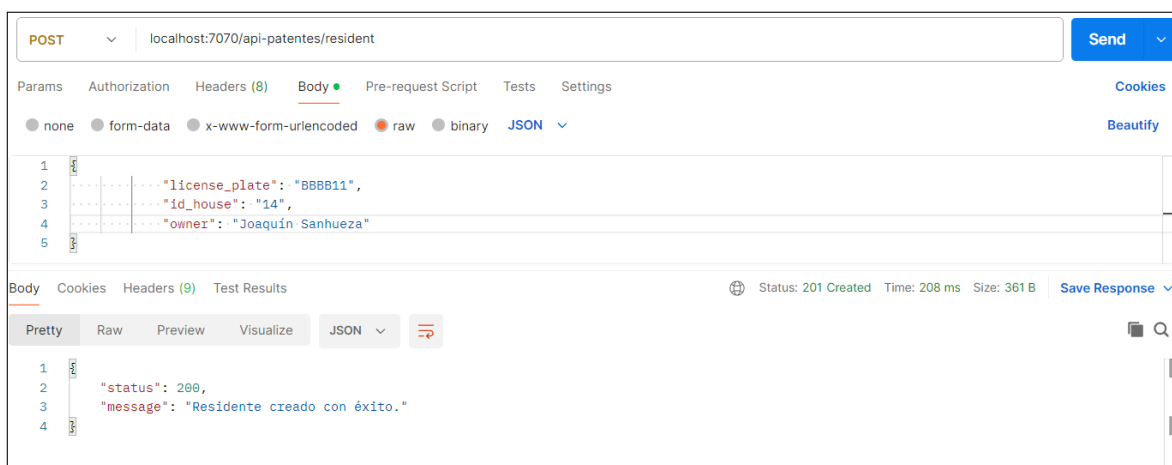


Figura 4.20: Respuesta Correcta de createResident de API-Patentes.

4.1.3.9. Actualización de Estado Publicado

El propósito de este endpoint es actualizar el estado de publicación (*published*) de una notificación. Se utiliza el método HTTP **PUT**, y su ruta se especifica como *api-patentes/notification/{id_}*. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	La notificación ha sido actualizada con éxito.
400	Bad Request	Error al realizar solicitud. Verifique parámetros.
500	Internal Server Error	Error al actualizar la notificación.

Cuadro 4.21: Respuestas del sistema updatePublishedStateNotification de API-Patentes.

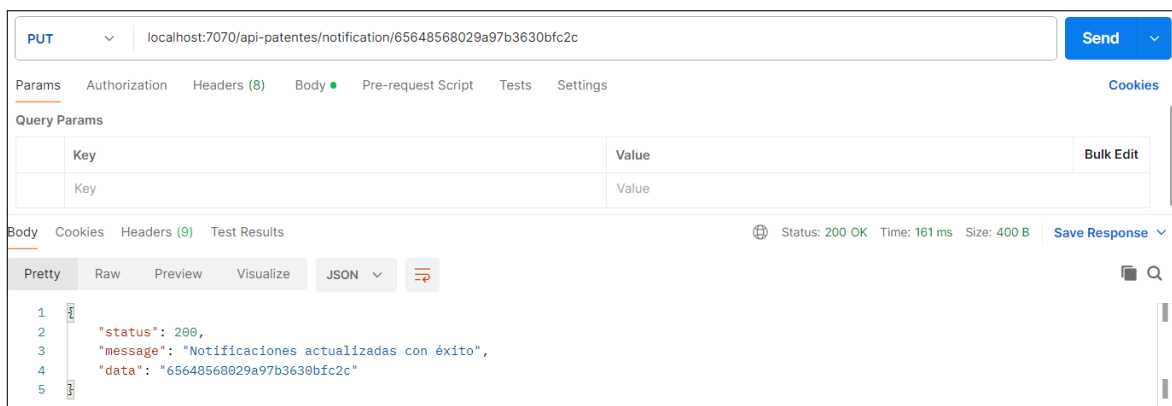


Figura 4.21: Respuesta Correcta de updatePublishedStateNotification de API-Patentes.

4.1.3.10. Obtener todas las visitas registradas en una fecha

El objetivo de este endpoint es recuperar todas las visitas registradas en una fecha específica. Se utiliza el método HTTP **GET**, y su ruta se especifica como *api-patentes/visitors?date={date}*, donde el campo **date** representa la fecha para la cual se desean obtener las visitas. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a un array con todas las visitas del día.
204	No Content	Se notifica que la solicitud fue exitosa, pero no hay contenido.
500	Internal Server Error	Error al obtener todas las visitas.

Cuadro 4.22: Respuestas del sistema getAllVisitorsByDate de API-Patentes.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:7070/api-patentes/visitors?date=26-11-2023
- Query Params:** date=26-11-2023
- Status:** 200 OK
- Time:** 157 ms
- Size:** 1.79 KB
- Response Body (JSON):**

```
{
  "status": 200,
  "data": [
    {
      "_id": "65635036dfa662e30834071a",
      "license_plate": "Ksd",
      "id_house": "1234",
      "driver_dni": "19230422-9",
      "driver_name": "Raúl Perez",
      "start_date": "2023-11-26T14:03:34.000Z",
      "end_date": "2023-11-26T15:03:34.957Z",
      "__v": 0
    },
    {
      "_id": "65635273f5417ee6b2089c70",
      "license_plate": "Ksd",
      "id_house": "1234",
      "driver_dni": "19230422-9",
      "driver_name": "Raúl Perez",
      "start_date": "2023-11-26T14:03:34.000Z",
      "end_date": "2023-11-26T15:03:34.957Z",
      "__v": 0
    }
  ]
}
```

Figura 4.22: Respuesta Correcta de getAllVisitorsByDate de API-Patentes.

4.1.4. API-Gateway

4.1.4.1. Login

Este endpoint tiene como objetivo autenticar al usuario a través de la **API-Usuarios**. En caso de que el usuario esté registrado, se genera un token mediante JWT, el cual será requerido en todas las futuras solicitudes a la **API-Gateway** a través del encabezado (header), proporcionando así una primera capa de seguridad en la plataforma.

La solicitud se realiza mediante el método HTTP **POST**, y su ruta se especifica como *api-gateway/login*. Se deben enviar las credenciales (username y password) para llevar a cabo la validación. A continuación, se describen las posibles respuestas del sistema, cada una con su correspondiente código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a los datos del usuario y el token válido.
400	Ok	Error al realizar la solicitud. Verifique los parámetros.
401	Unauthorized	Se notifica que las credenciales son inválidas.
500	Internal Server Error	Error al iniciar sesión.

Cuadro 4.23: Respuestas del sistema Login de API-Gateway

4.1.4.2. Obtener Todas las Notificaciones No Publicadas

El propósito de este endpoint es recuperar todas las notificaciones, tanto de la *API-Intrusos* como de la *api-patentes*. Es relevante destacar que estas notificaciones se ordenan por prioridad y fecha. Se aplica inicialmente un filtro de fecha y hora, seguido de la priorización de las notificaciones enviadas por el detector de intrusos. Se emplea el método HTTP GET, y su ruta se define como *api-gateway/notifications-no-published*. A continuación, se detallan las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a las notificaciones.
204	No Content	No hay notificaciones pendientes por publicar.
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al obtener notificaciones.

Cuadro 4.24: Respuestas del sistema getAllNoPublishedNotifications de API-Gateway

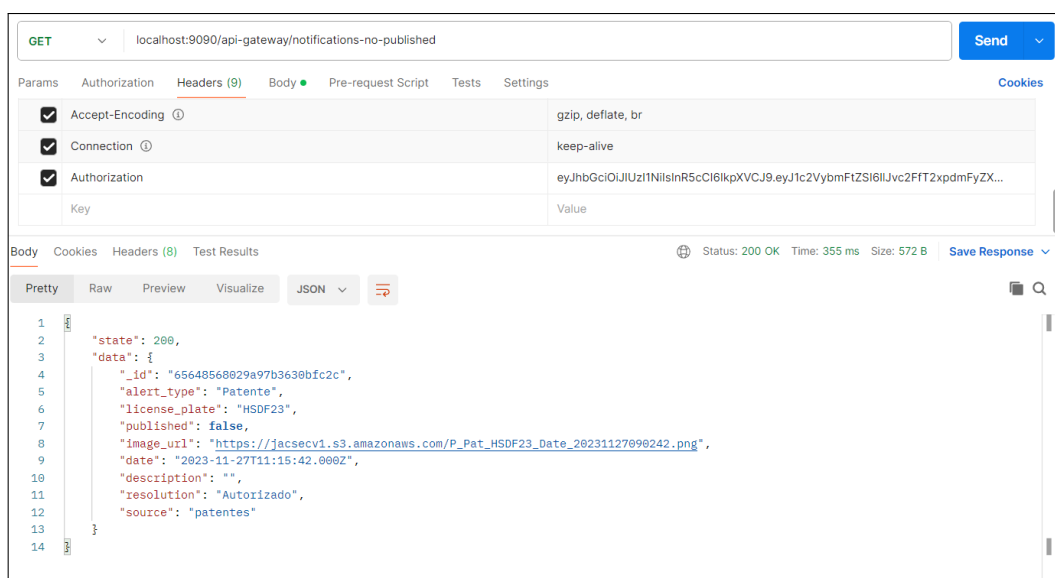


Figura 4.24: Respuesta Correcta de getAllNoPublishedNotifications de API-Gateway.

4.1.4.3. Obtener todos los eventos filtrados por fecha

El objetivo de este endpoint es obtener todas las notificaciones y eventos notificados por el usuario en una fecha específica. Para lograrlo, se realiza una solicitud a las **API-Intrusos**, **API-Patentes** y **API-Usuarios**. Una vez recopiladas todas las notificaciones y eventos, se ordenan por hora.

Se utiliza el método HTTP **GET**, y su ruta se especifica como *api-gateway/notifications/id_user?date={date}*, donde **date** representa la fecha de filtración. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a las notificaciones ordenadas por hora.
204	No Content	No hay notificaciones.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros.
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al obtener notificaciones.

Cuadro 4.25: Respuestas del sistema getAllNotificationsByDate de API-Gateway

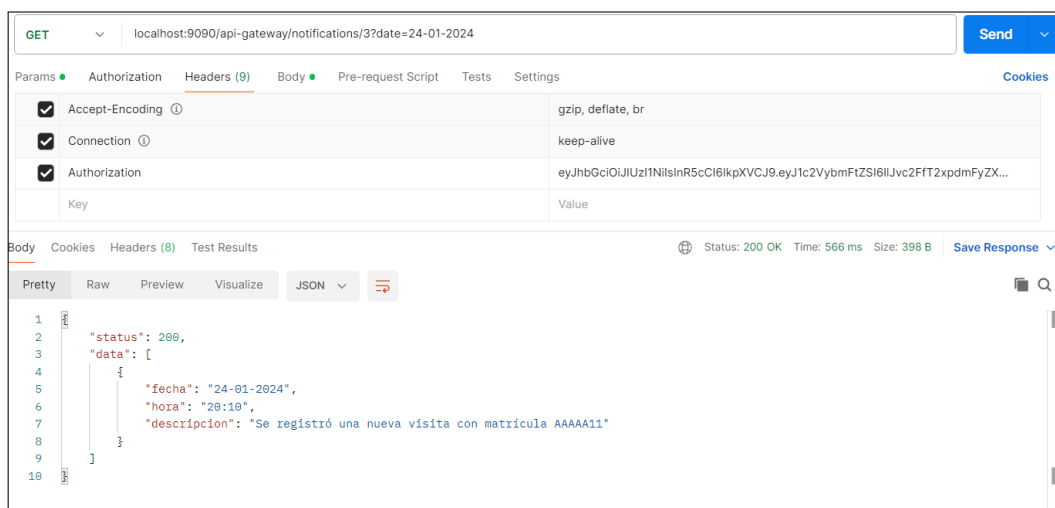


Figura 4.25: Respuesta Correcta de getAllNotificationsByDate de API-Gateway.

4.1.4.4. Registrar un visitante

El propósito de este endpoint es registrar una visita utilizando el servicio **API-Patentes**. Se emplea el método HTTP **POST**, y su ruta se especifica como *api-gateway/register-visitor*. Además, se deben incluir los siguientes campos: **license_plate**, **id_house**, **driver_dni** y **driver_name**. A continuación, se detallan las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
201	Created	Se ha registrado con éxito la visita.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al registrar visita.

Cuadro 4.26: Respuestas del sistema VisitorRegistration de API-Gateway

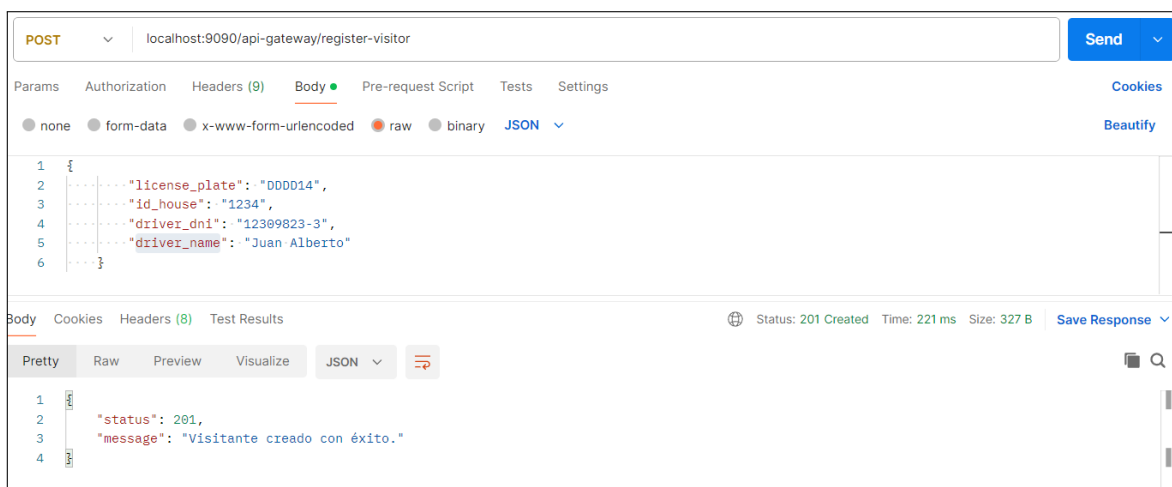


Figura 4.26: Respuesta Correcta de VisitorRegistration de API-Gateway.

4.1.4.5. Registrar evento anómalo

El objetivo de este endpoint es registrar un evento anómalo que ocurra durante el turno del conserje utilizando el servicio **API-Usuarios**. Se utiliza el método HTTP **POST**, y su ruta se especifica como *api-gateway/user/{id}/register-event*, donde **id** indica el identificador del usuario autenticado en el sistema. Además, se debe incluir el campo **description** en el cuerpo de la solicitud. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
201	Created	Se ha registrado con éxito el evento.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al registrar evento.

Cuadro 4.27: Respuestas del sistema createEventByUser de API-Gateway

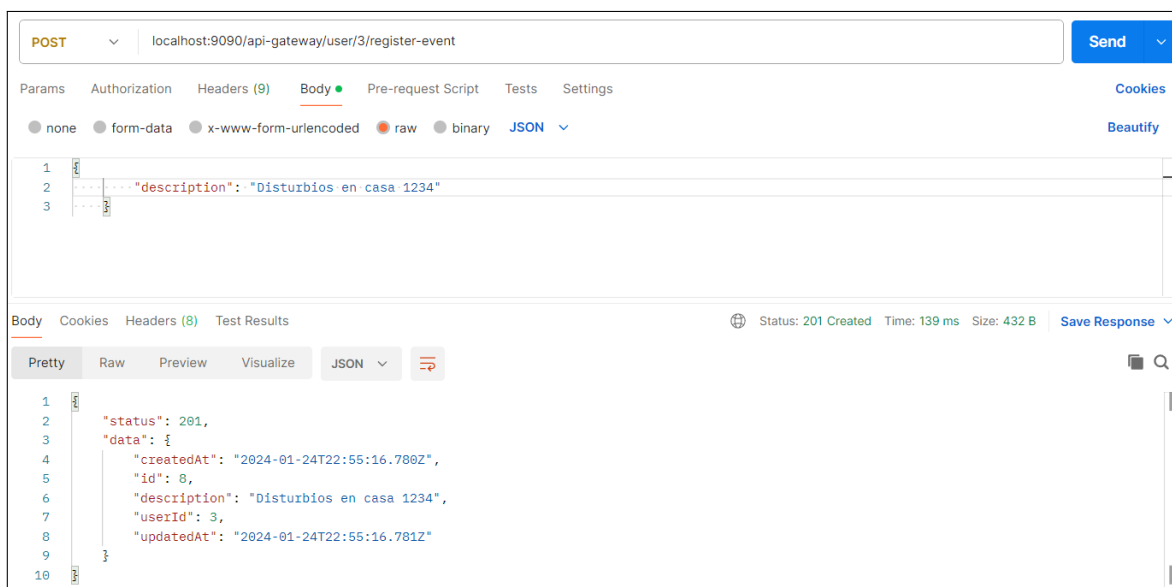


Figura 4.27: Respuesta Correcta de createEventByUser de API-Gateway.

4.1.4.6. Obtener Última Notificación de API-Intrusos

Este endpoint tiene como propósito obtener la última notificación enviada por el detector de intrusos. Consumiendo la *API-Intrusos*, se utiliza el método HTTP **GET**, y su ruta se define como *api-gateway/last-video-notification*. A continuación, se detallan las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a la última notificación que envió el detector.
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al obtener notificación.

Cuadro 4.28: Respuestas del sistema getLastVideoNotification de API-Gateway

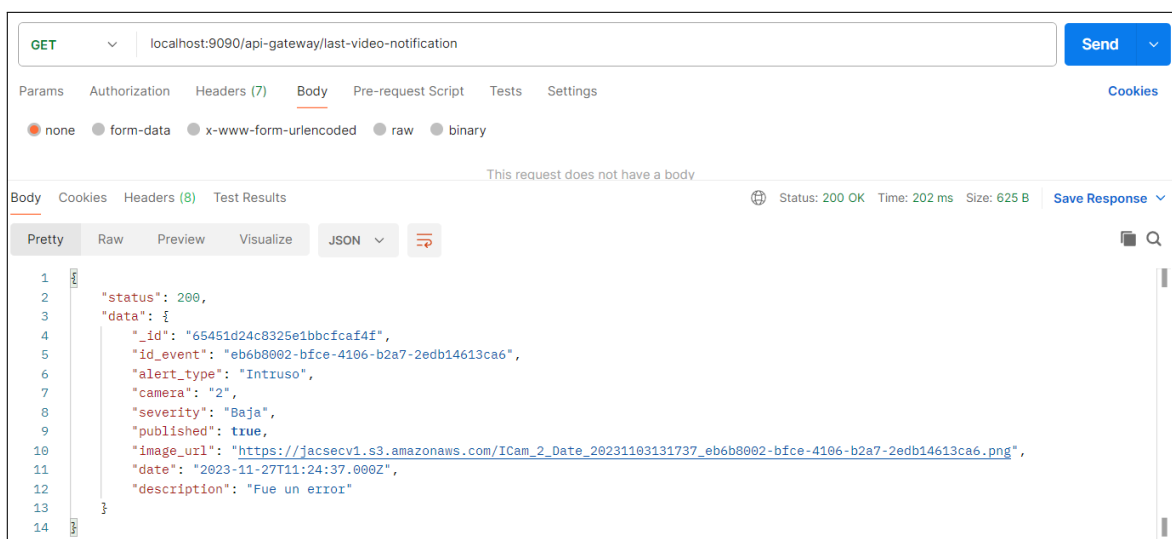


Figura 4.28: Respuesta Correcta de getLastVideoNotification de API-Gateway.

4.1.4.7. Obtener última notificación de API-Patentes

Este endpoint tiene como finalidad obtener la última notificación de la **API-Patentes**. Utiliza el método HTTP **GET**, y su ruta se define como *api-gateway/last-plate-notification*. A continuación, se describen las posibles respuestas del sistema, cada una con su respectivo código.

Código HTTP	Respuesta	Descripción
200	Ok	Se notifica que la solicitud fue exitosa junto a la última notificación que envió el detector.
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al obtener notificación.

Cuadro 4.29: Respuestas del sistema getLastPlateNotification de API-Gateway.

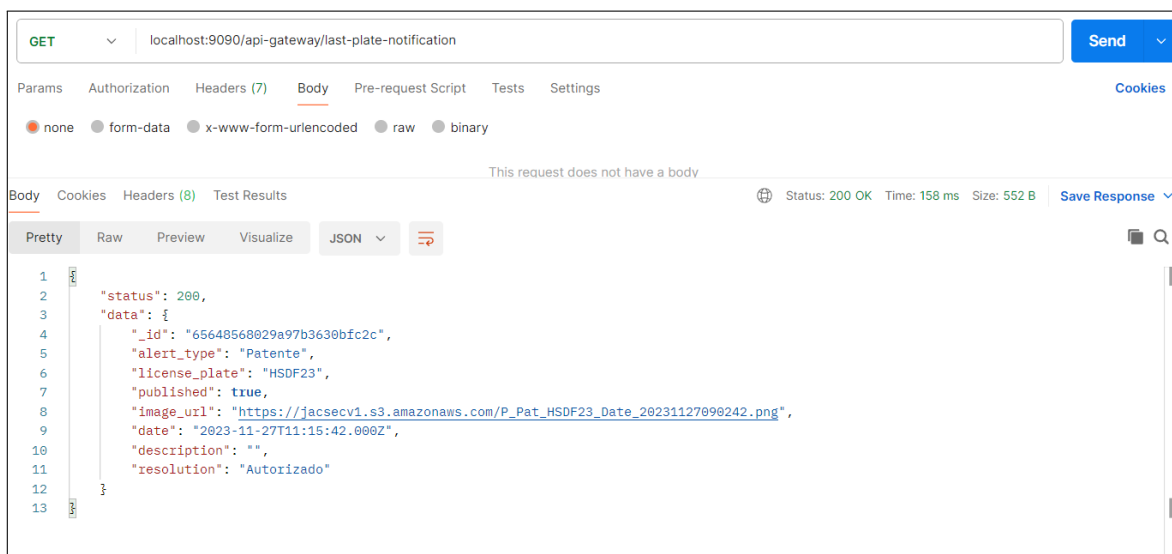


Figura 4.29: Respuesta Correcta de getLastPlateNotification de API-Gateway.

4.1.4.8. Modificar campos de Notificaciones de API-Intrusos

Este endpoint tiene como objetivo proporcionar una descripción y clasificar la gravedad de la alerta generada por el conserje. Se realiza una llamada a la *API-Intrusos*.

Se utiliza el método HTTP **PUT**, y su ruta se especifica como *api-gateway/last-video-notification?id_event={id_event}*, donde el campo **id_event** representa el identificador del evento. Además, se deben incluir la descripción y la gravedad en el cuerpo de la solicitud. A continuación, se presentan las posibles respuestas del sistema junto con sus respectivos códigos.

Código HTTP	Respuesta	Descripción
200	Ok	Se actualizó correctamente la notificación.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros.
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al actualizar la notificación.

Cuadro 4.30: Respuestas del sistema updateLastVideoNotification de API-Gateway

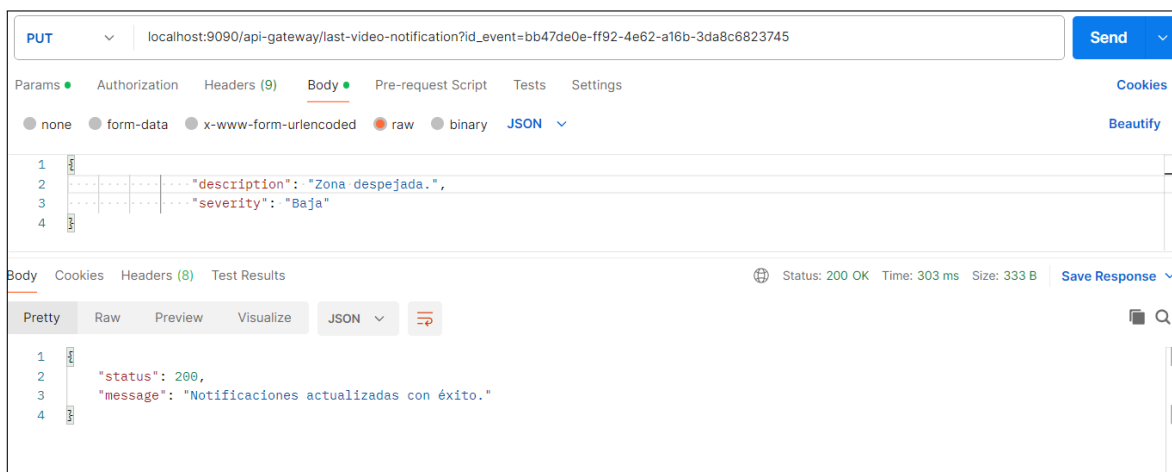


Figura 4.30: Respuesta Correcta de updateLastVideoNotification de API-Gateway.

4.1.4.9. Modificar Estado Publicado de Notificación

Este endpoint tiene como objetivo modificar el estado de publicación de las notificaciones mediante una llamada a las API **API-Intrusos** y **API-Patentes**. Se utiliza el método HTTP **PUT**, y la ruta se especifica como *api-gateway/notification/{id}*, donde **id** es el identificador del evento. Para la API-Patentes, se utiliza **id_**, y para la API-Intrusos, se utiliza **id_event**. A continuación, se detallan las posibles respuestas del sistema, junto con sus correspondientes códigos.

Código HTTP	Respuesta	Descripción
200	Ok	Se actualizó correctamente la notificación.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros.
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al actualizar el estado de la notificación.

Cuadro 4.31: Respuestas del sistema updatePublishedState de API-Gateway

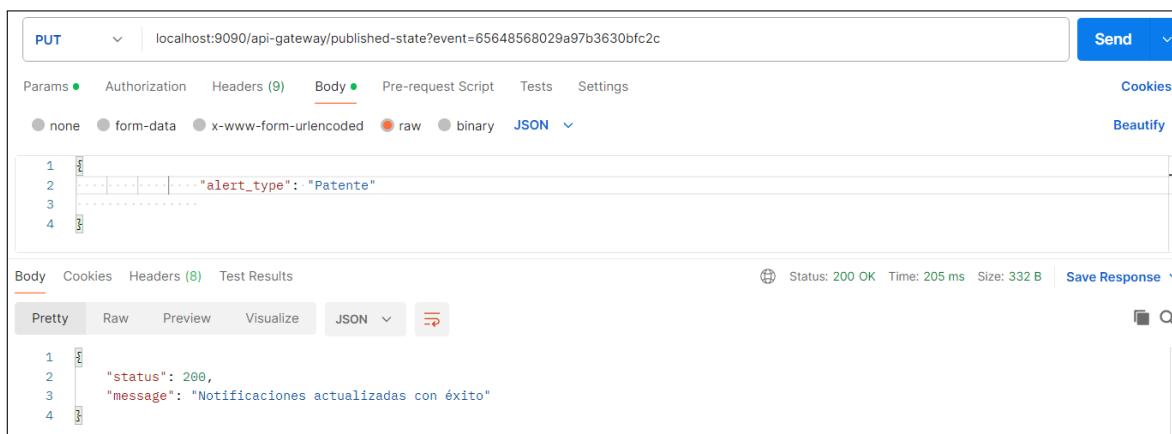


Figura 4.31: Respuesta Correcta de updatePublishedState de API-Gateway.

4.1.4.10. Modificar Resolución de Notificación de Api-Patentes

Este endpoint está diseñado para ajustar la resolución de las notificaciones emitidas por el detector de Patentes. Se emplea el método HTTP **PUT**, y la ruta se define como *api-gateway/notification/{id_}/resolution*, donde **id_** representa el identificador único de la notificación. A continuación, se describen las respuestas potenciales del sistema, acompañadas de sus respectivos códigos.

Código HTTP	Respuesta	Descripción
200	Ok	Se actualizó correctamente la notificación.
400	Bad Request	Error al realizar la solicitud. Verifique los parámetros.
401	Unauthorized	Se notifica que el Token es inválido.
500	Internal Server Error	Error al actualizar el estado de la notificación.

Cuadro 4.32: Respuestas del sistema updatePublishedState de API-Gateway

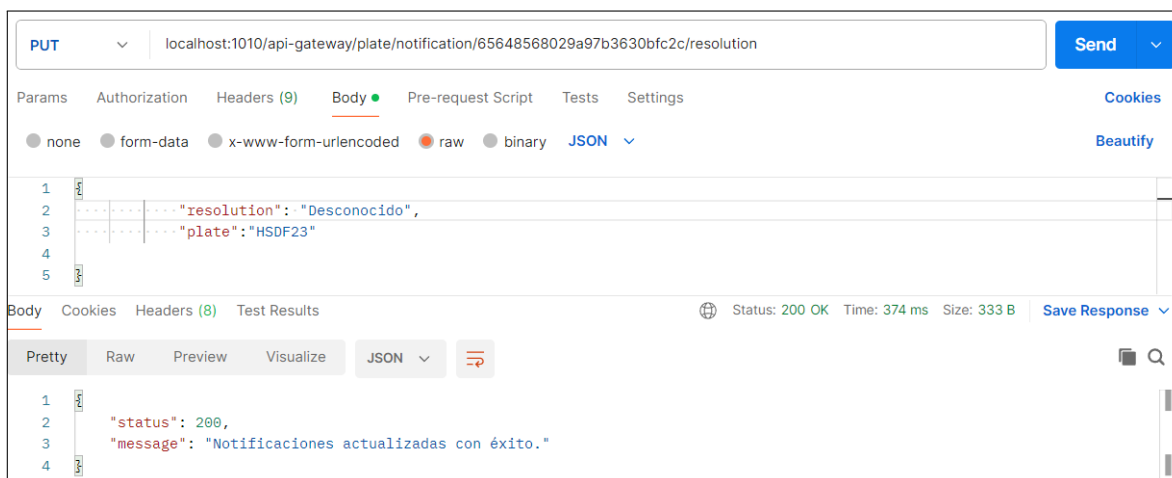


Figura 4.32: Respuesta Correcta de updateResolutionNotification de API-Gateway.

4.2. Front-end

Con el fin de mejorar la comprensión, esta sección se organizará en cuatro subsecciones. Esto se debe a que la plataforma consta de una vista de inicio de sesión y otra compuesta por tres paneles principales, según la solicitud explícita de la empresa de mantener la interfaz simple.

Cada subsección detallará los requisitos funcionales que se cumplen, junto con el correspondiente diagrama de flujo de usuario para describir el proceso de navegación.

4.2.1. Vista de Inicio de Sesión

La vista de inicio de sesión es un elemento fundamental para autenticar a los usuarios antes de que accedan a la plataforma (RF9). La Figura 4.33 representa la interacción entre los componentes durante este proceso. Desde el Frontend, se emite una solicitud a la API-Gateway, la cual valida las credenciales mediante la API-Usuarios. Si las credenciales son válidas, se genera un token necesario para todas las solicitudes posteriores a la API-Gateway.

Además, en la Figura 4.34 se presenta el diagrama de flujo de usuario, que detalla el proceso de acceso a la plataforma.

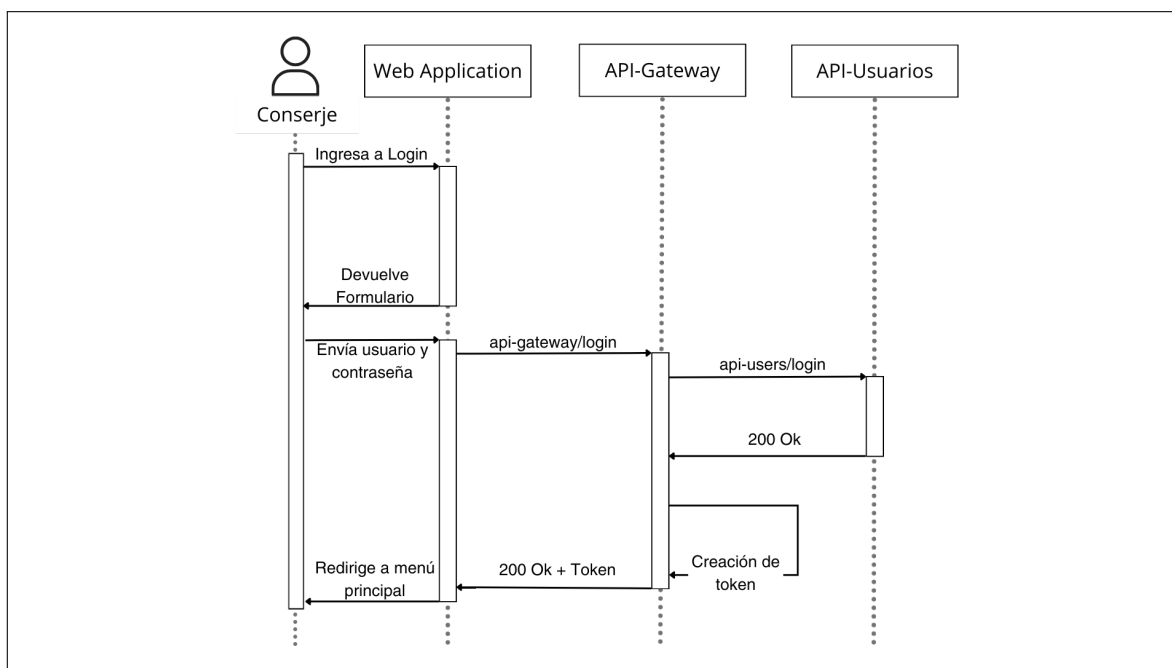


Figura 4.33: Diagrama de Secuencia Inicio de Sesión.

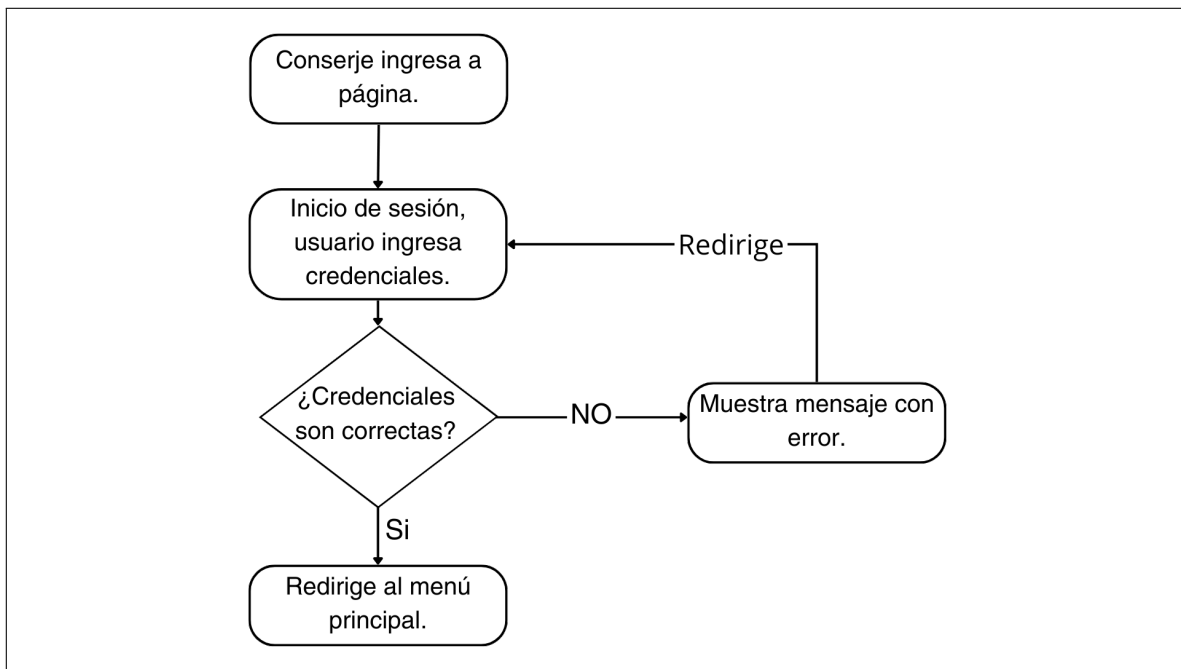


Figura 4.34: Diagrama de Flujo de Inicio de Sesión.

4.2.1.1. Implementación de Vista

Figura 4.35: Vista implementada de Login.

4.2.2. Vista de Menú Principal

El menú principal se estructura en tres paneles distintos como se puede visualizar en la figura 4.36 en la parte superior derecha se exhiben las alertas de Videovigilancia, en la parte superior izquierda se presentan las alertas de vehículos, mientras que la parte inferior contiene una bitácora con todos los eventos ocurridos en el día. Con el fin de describir cada panel de manera clara, los separaremos según su funcionalidad.

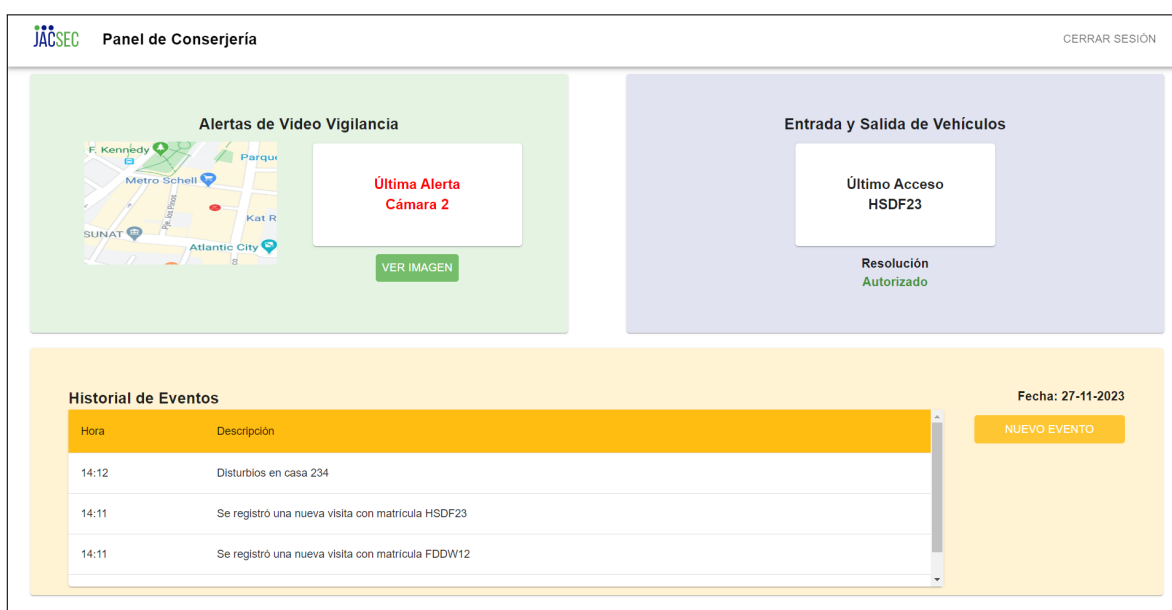


Figura 4.36: Vista implementada de Menú principal.

4.2.2.1. Panel de Alertas de Videovigilancia

Este panel está diseñado para mostrar la notificación más reciente enviada por el detector de Intrusos (RF7). Aquí, el usuario tiene la capacidad de modificar la gravedad y la descripción de la última alerta (RF6). En la Figura 4.37, se representa cómo interactúan los componentes para obtener la última notificación publicada. Por otro lado, en la Figura 4.38, se detalla la interacción entre los componentes al realizar modificaciones en la última notificación.

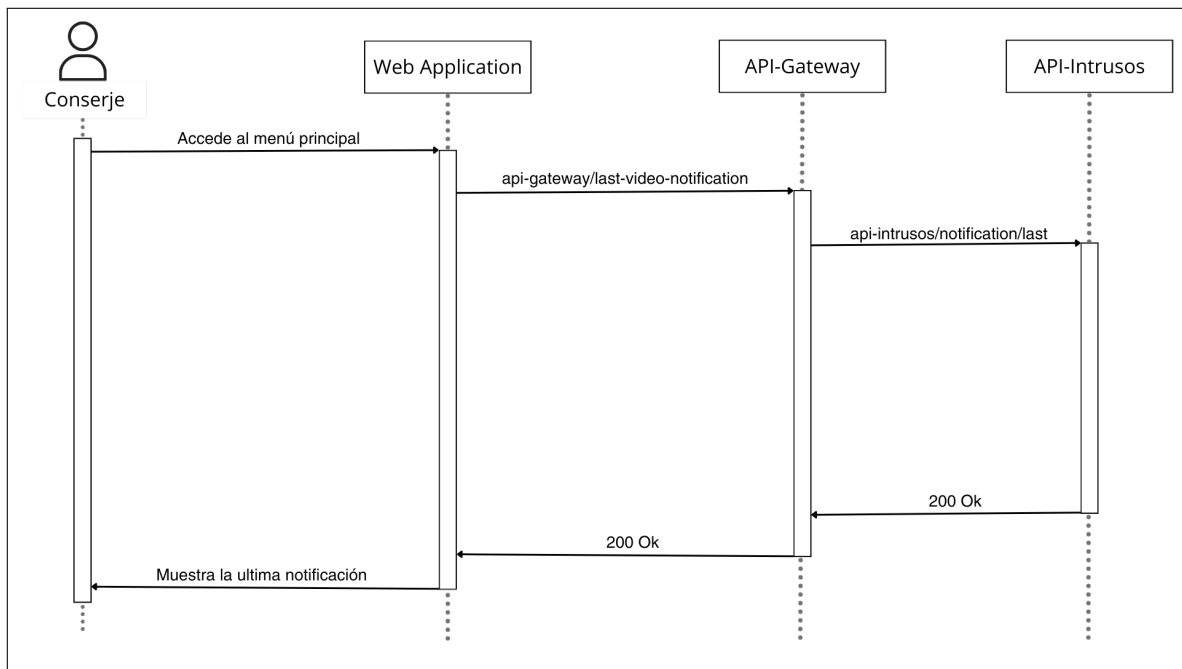


Figura 4.37: Diagrama de secuencia RF7.

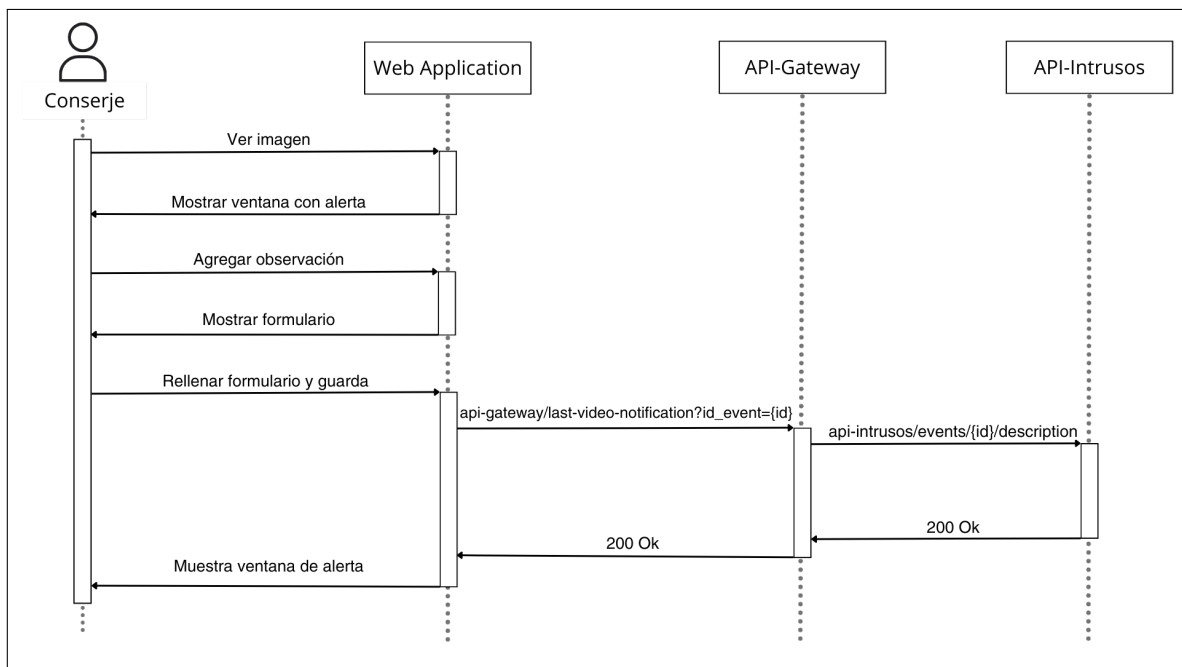


Figura 4.38: Diagrama de secuencia RF6.

A continuación, procederemos a describir el diagrama de flujo del usuario para estos requerimientos en la Figura 4.39.

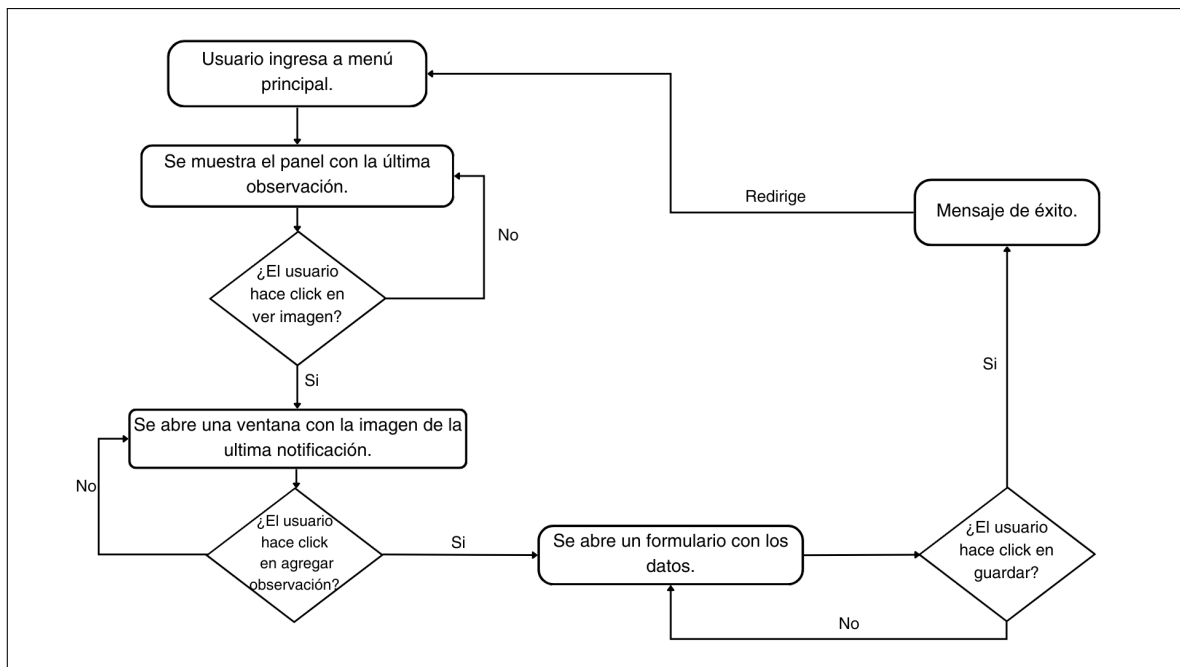


Figura 4.39: Diagrama de Flujo de Usuario para el Panel de Videovigilancia.

En la figuras 4.40 y 4.41 se ilustran las vistas asociadas a los requerimientos.



Figura 4.40: Vista de notificación de intrusos.

JACSEC Panel de Conserjería CERRAR SESIÓN

Alerta de Intruso Cámara 2

Gravedad de Alerta

☒ Baja

☐ Media

☐ Alta

Observación

Descripción

GUARDAR

Historial

Hora
14:12
14:11
14:11

11-2023

Figura 4.41: Vista de formulario para modificar notificación de intrusos.

4.2.2.2. Panel de Entrada y Salida de Vehículos

El Panel de Entrada y Salida de Vehículos tiene la responsabilidad de mostrar la última notificación enviada por el detector de patentes (RF8), lo que simplifica la visualización para determinar si el vehículo ha sido aceptado o marcado como desconocido. En esta sección, describiremos el diagrama de secuencia relacionado con el panel, el cual se presenta a la derecha en la Figura 4.36.

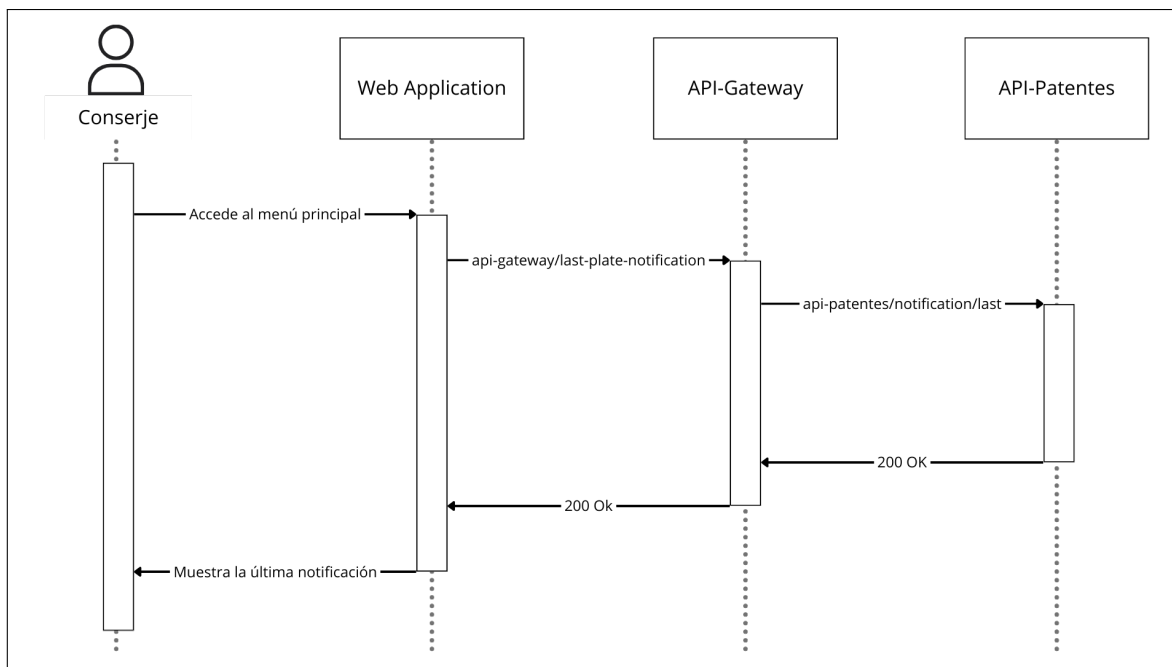


Figura 4.42: Diagrama de secuencia RF8.

Además, se generan notificaciones a través del detector de patentes (RF2). A continuación, delinearemos, mediante un diagrama de flujo de usuario, la secuencia que el usuario puede seguir al recibir este tipo de alerta.

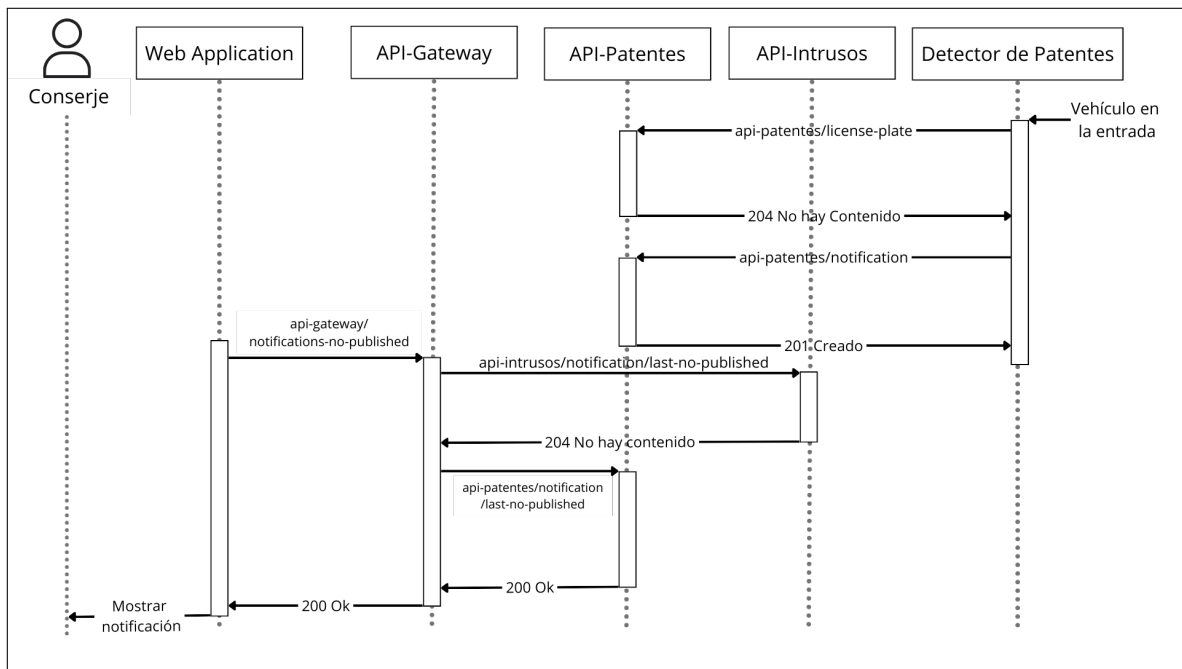


Figura 4.43: Diagrama de secuencia RF2.

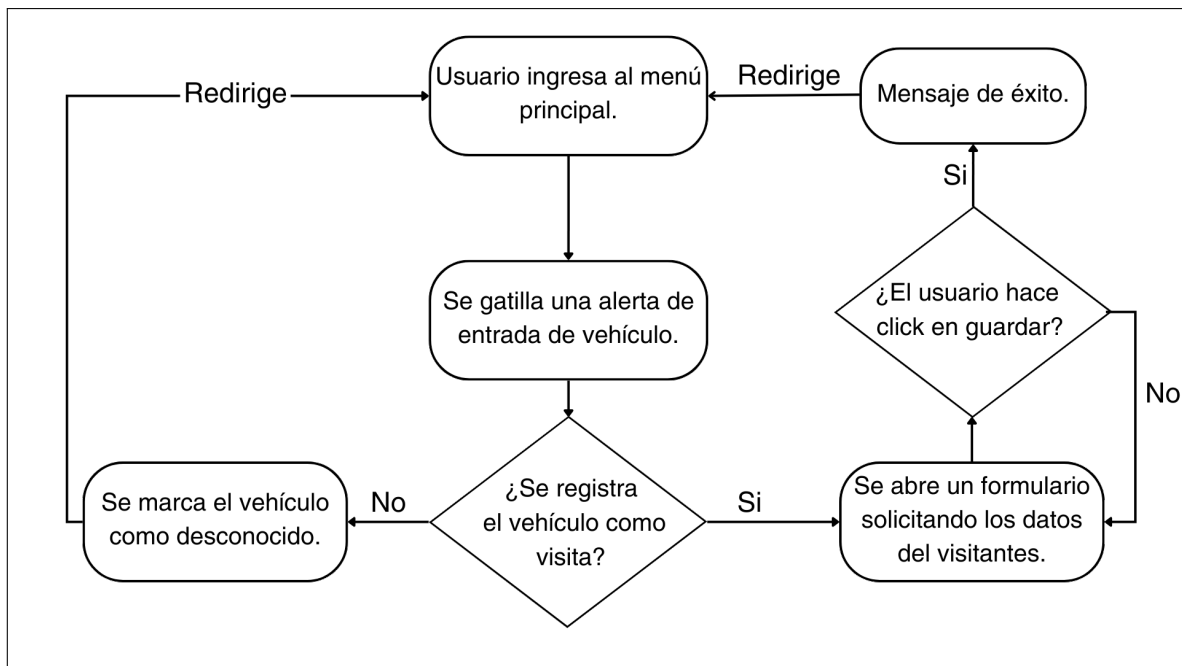


Figura 4.44: Diagrama de Flujo RF2.

A continuación, se enseñan las vistas asociadas al RF2.

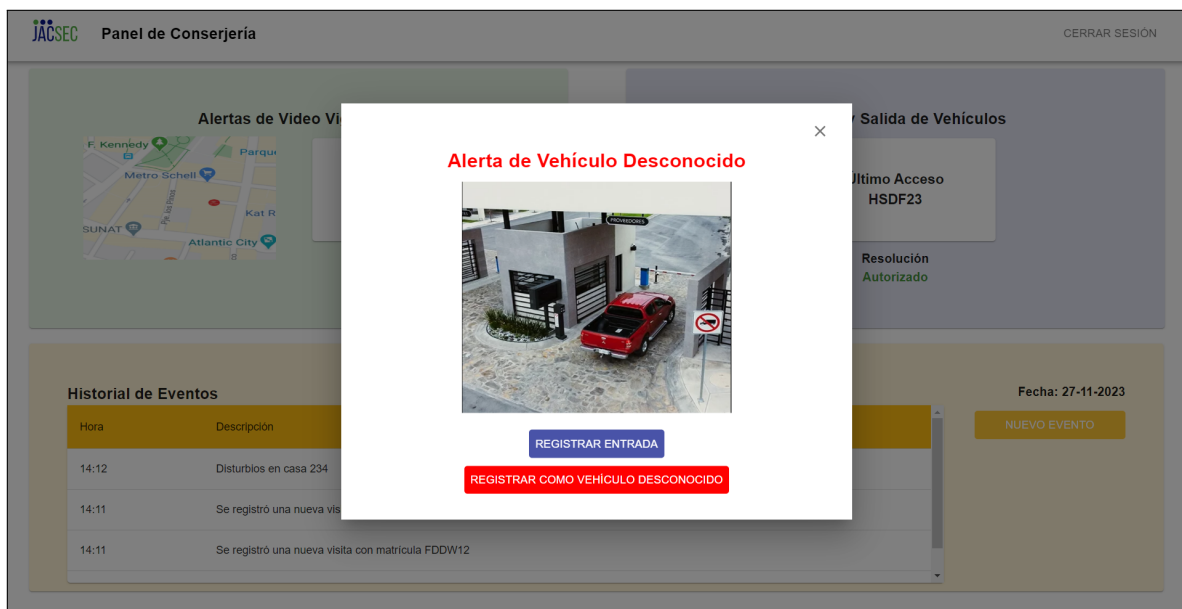


Figura 4.45: Vista de notificación de vehículo desconocido.

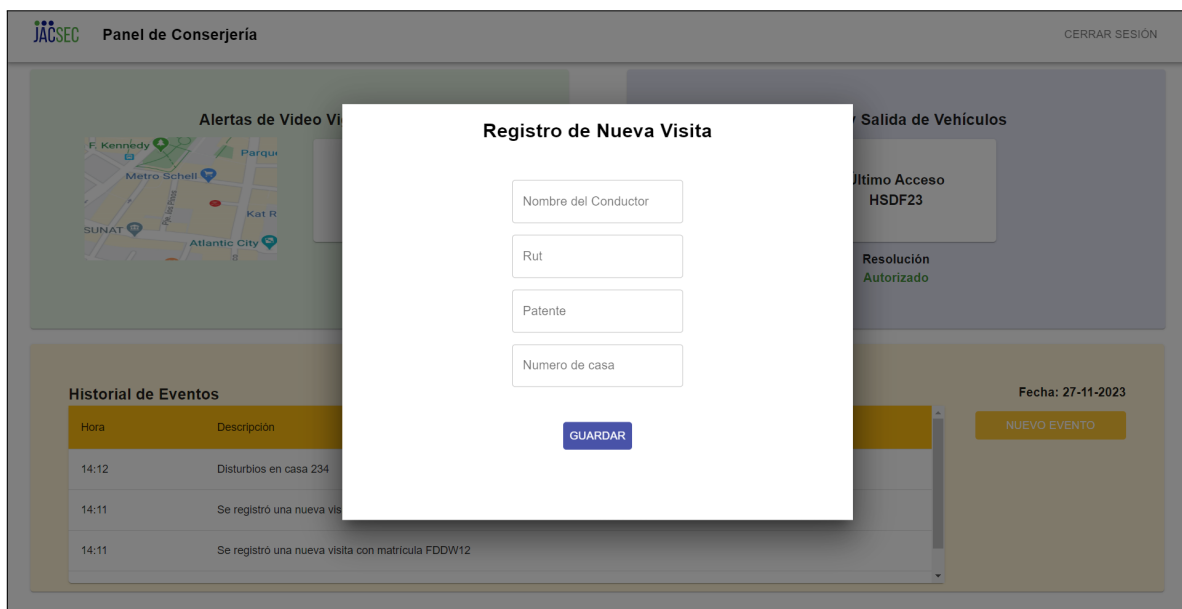


Figura 4.46: Vista de registro de visita.

4.2.2.3. Historial de Eventos

El panel de historial de eventos presenta una bitácora que recopila todos los eventos registrados a lo largo del día (RF3). Además, brinda la posibilidad al conserje de registrar nuevos eventos, ya sea programando visitas o reportando incidencias inesperadas durante su turno laboral.

A continuación, se presentará el diagrama de secuencia para la obtención de todos los eventos del día.

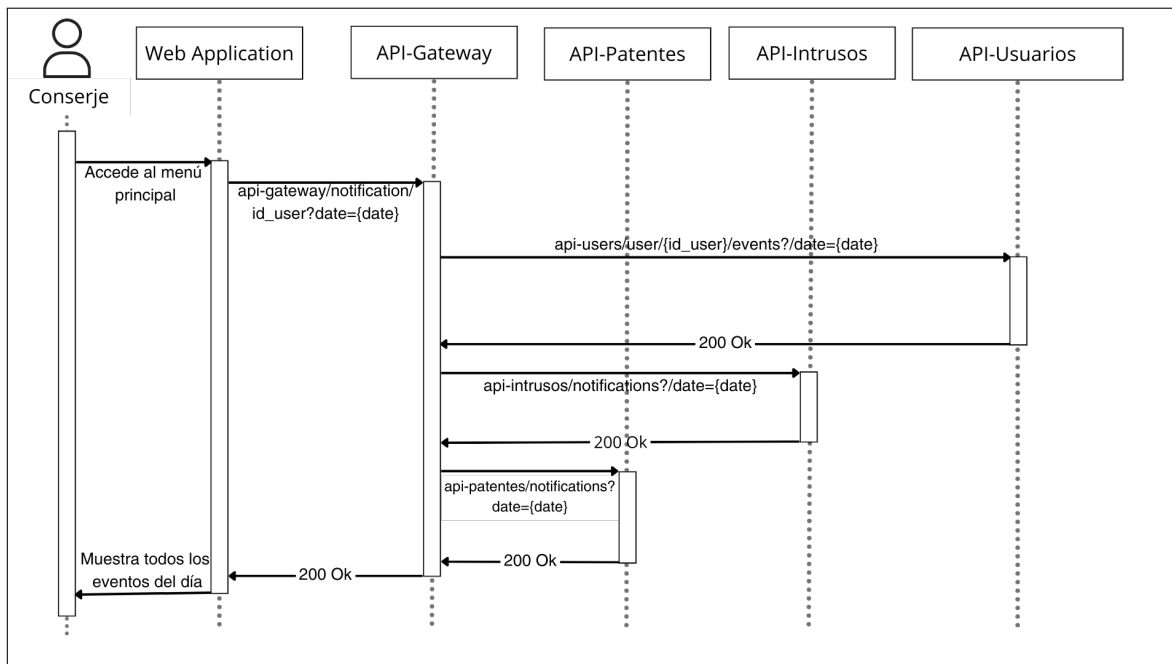


Figura 4.47: Diagrama de secuencia RF3.

Como se mencionó previamente, en este panel se pueden registrar tanto visitas como eventos anómalos. Para el registro de visitas (RF4), los mismos componentes interactúan como se ilustra en la figura 4.48. Por otro lado, en la figura 4.49 se detalla cómo interactúan los componentes para el registro de un evento anómalo (RF5).

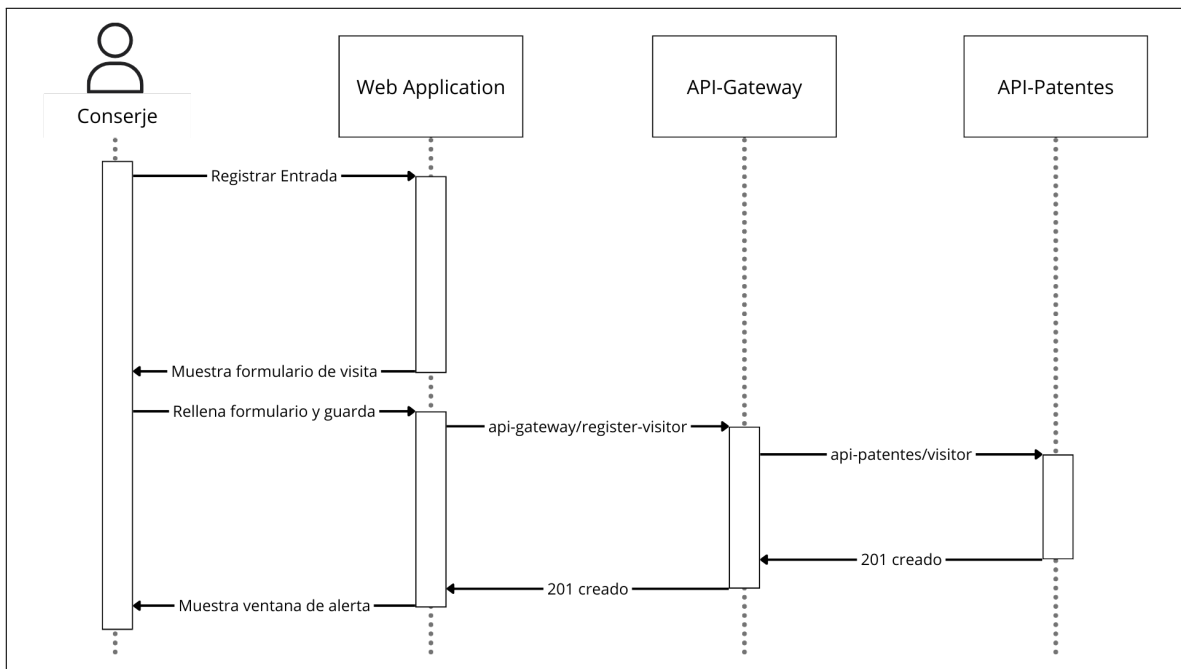


Figura 4.48: Diagrama de secuencia RF4.

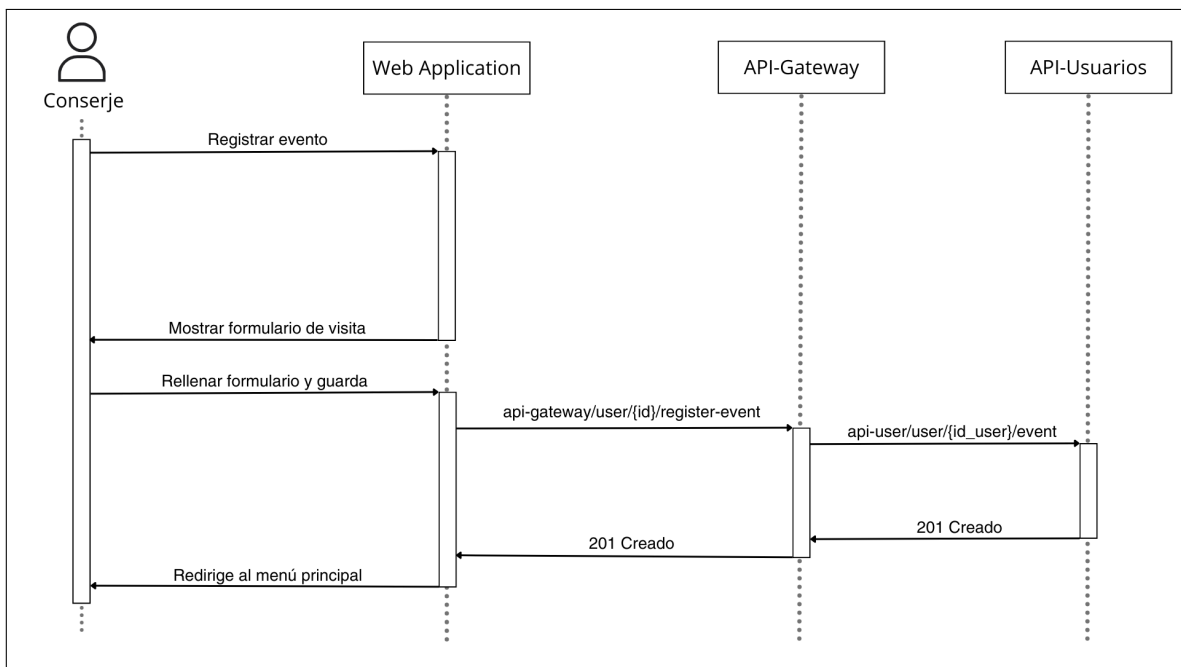


Figura 4.49: Diagrama de secuencia RF5.

Seguidamente, se exhibirá el diagrama de flujo del usuario asociado al registro de un nuevo evento.

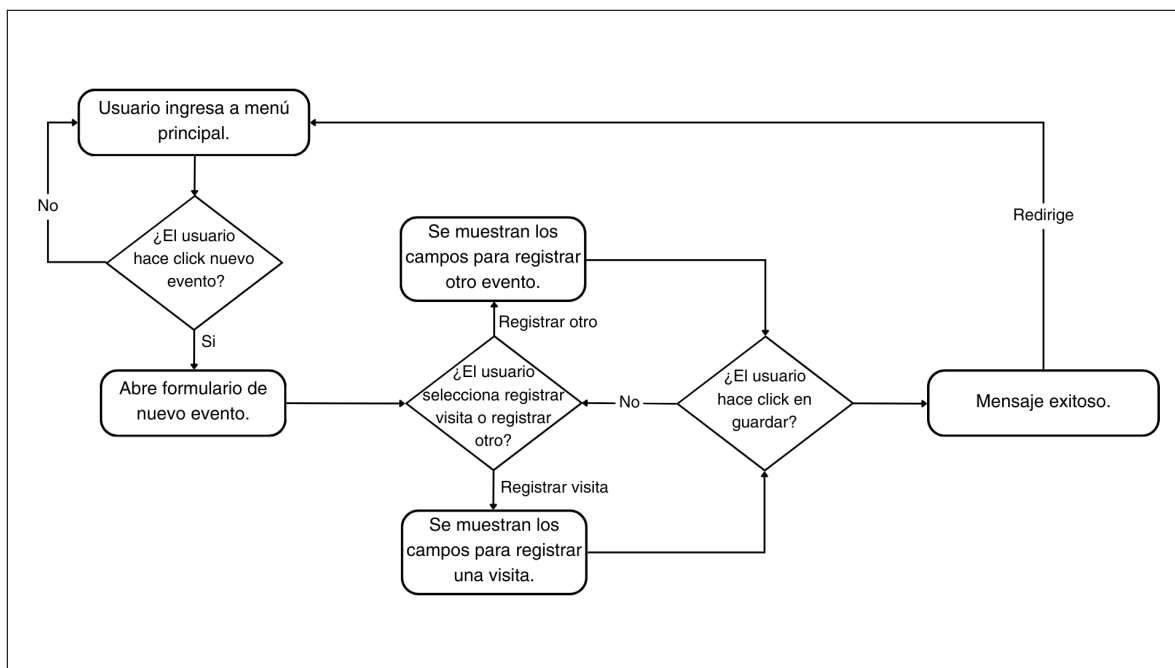


Figura 4.50: Diagrama de Flujo de Nuevo Evento.

Por último, se presentarán las vistas relacionadas con este panel.

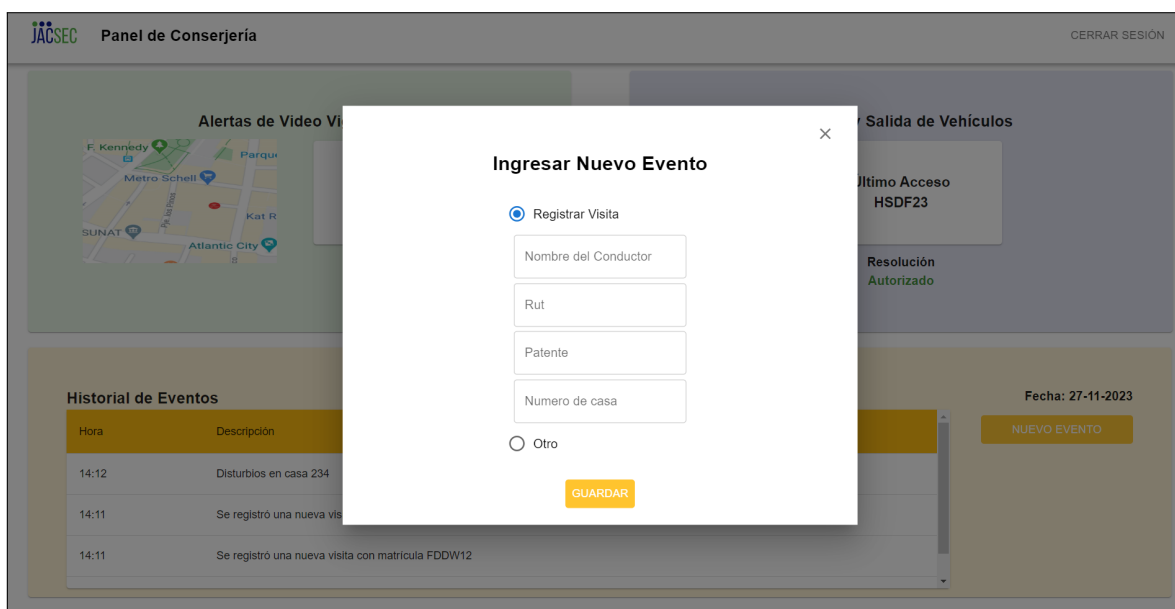


Figura 4.51: Vista de registro de nueva visita.

JACSEC Panel de Conserjería CERRAR SESIÓN

Alertas de Video Vi

Salida de Vehículos

Ultimo Acceso
HSDF23

Resolución
Autorizado

Fecha: 27-11-2023

NUEVO EVENTO

Ingresar Nuevo Evento

☐ Registrar Visita

☒ Otro

Descripción

GUARDAR

Historial de Eventos

Hora	Descripción
14:12	Disturbios en casa 234
14:11	Se registró una nueva visita
14:11	Se registró una nueva visita con matrícula FDDW12

Figura 4.52: Vista de Registro de evento anómalo.

Capítulo 5

Validaciones y Pruebas

En esta sección se presentará la verificación, validación y pruebas de los componentes del sistema.

5.1. Verificación y Pruebas

Para asegurar el correcto funcionamiento de todas las características, es necesario realizar pruebas exhaustivas. Esto implica llevar a cabo pruebas unitarias y de integración.

Con el fin de cumplir con este objetivo, se ejecutó una suite de pruebas diseñada para evaluar a fondo la funcionalidad de cada microservicio. Se emplearon herramientas como Jest, Sinon y Supertest para las pruebas de los servicios, ya que estas herramientas están especializadas en pruebas unitarias y de integración.

A continuación, se presentan los resultados de la cobertura de código para cada aplicación proporcionados por Jest:

5.1.1. API-Usuarios

All files
 100% Statements 165/165 100% Branches 27/27 100% Functions 13/13 100% Lines 165/165

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.
 Filter:

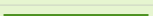
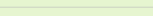
File		Statements	Branches	Functions	Lines
users_backend		100%	15/15	100%	15/15
users_backend/controllers		100%	54/54	27/27	54/54
users_backend/models		100%	22/22	0/0	22/22
users_backend/repository		100%	30/30	0/0	30/30
users_backend/routes		100%	33/33	0/0	33/33
users_backend/utils		100%	7/7	0/0	7/7

Figura 5.1: Resultados de Code Coverage de API-Usuarios

5.1.2. API-Intrusos

All files
 100% Statements 167/167 100% Branches 33/33 100% Functions 12/12 100% Lines 167/167

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.
 Filter:

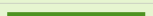
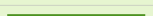
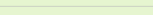
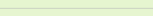
File		Statements	Branches	Functions	Lines
intrusos_backend		100%	17/17	0/0	17/17
intrusos_backend/src/controllers		100%	65/65	29/29	65/65
intrusos_backend/src/models		100%	3/3	0/0	3/3
intrusos_backend/src/repository		100%	37/37	4/4	37/37
intrusos_backend/src/routes		100%	45/45	0/0	45/45

Figura 5.2: Resultados de Code Coverage de API-Intrusos

5.1.3. API-Patentes

All files
 100% Statements 283/283 100% Branches 64/64 100% Functions 19/19 100% Lines 283/283

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.
 Filter:

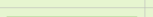
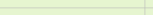
File		Statements	Branches	Functions	Lines
patentes_backend		100%	17/17	0/0	17/17
patentes_backend/src/controllers		100%	116/116	60/60	116/116
patentes_backend/src/models		100%	9/9	0/0	9/9
patentes_backend/src/repository		100%	72/72	4/4	72/72
patentes_backend/src/routes		100%	69/69	0/0	69/69

Figura 5.3: Resultados de Code Coverage de API-Patentes

5.1.4. API-Gateway

All files									
100% Statements 338/338 100% Branches 77/77 100% Functions 29/29 100% Lines 338/338									
Press <i>n</i> or <i>j</i> to go to the next uncovered block, <i>b</i> , <i>p</i> or <i>k</i> for the previous block.									
Filter:									
File ▾	Statements ▾	Branches ▾	Functions ▾	Lines ▾					
api_gateway		100%	10/10	100%	0/0	100%	0/0	100%	10/10
api_gateway/controllers		100%	196/196	100%	77/77	100%	11/11	100%	196/196
api_gateway/repository		100%	51/51	100%	0/0	100%	15/15	100%	51/51
api_gateway/routes		100%	63/63	100%	0/0	100%	0/0	100%	63/63
api_gateway/utils		100%	18/18	100%	0/0	100%	3/3	100%	18/18

Figura 5.4: Resultados de Code Coverage de API-Gateway

5.2. Validaciones

Para validar el sistema, presentamos el prototipo final a la empresa. La retroalimentación recibida fue positiva, lo que nos llevó a considerar este trabajo en conjunto como la primera versión funcional del software.

Capítulo 6

Conclusiones y Trabajos futuros

En la sociedad actual, nos enfrentamos a un desafío significativo, la delincuencia. Recuperar el sentimiento de seguridad tras ser vulnerados es una tarea ardua. Es por ello que esperamos que nuestra solución brinde una mayor facilidad para detectar a intrusos o delincuentes antes de que cometan un delito, y que podamos recuperar de esta forma la sensación de seguridad en nuestros hogares.

Además, confiamos en que este prototipo facilite el trabajo de los conserjes, quienes se encuentran constantemente atentos a múltiples eventos simultáneos. A menudo, estas responsabilidades resultan abrumadoras para una sola persona.

En relación con la aplicación web, esta ha sido sometida a pruebas conjuntas con el detector de intrusos, arrojando resultados positivos. No obstante, la integración del módulo de detección de patentes aún no se ha completado debido a inconvenientes de desarrollo en esa área. A pesar de ello, no anticipamos encontrar dificultades significativas durante la integración, puesto que se basa en el consumo de la aplicación API-Patentes, la cual ha sido exhaustivamente verificada.

El éxito final de este trabajo estará determinado por la validación de la experiencia del usuario final, en este caso, el conserje, así como por la integración completa de todos los módulos previstos.

6.1. Trabajos futuros

Este trabajo de titulación fue concebido como un prototipo inicial que aborda los módulos más relevantes para la empresa. Sin embargo, se contempla la posibilidad de expandir su funcionalidad en el futuro, incluyendo características adicionales como el reconocimiento facial para facilitar el acceso de personas. Para lograr esto, se planea implementar un nuevo microservicio en conjunto con la interfaz correspondiente.

En cuanto al prototipo funcional actual, se han identificado áreas de mejora que podrían optimizar el rendimiento de la aplicación. Entre estas mejoras se encuentra la posible incorporación de GraphQL [38] en la capa de API-Gateway. GraphQL es un lenguaje de consulta para APIs que proporciona una forma más eficiente y flexible de acceder a los recursos de un servidor. Su adopción en la API-Gateway permitiría una comunicación más armoniosa y eficaz entre los distintos servicios de la aplicación, mejorando así la experiencia del usuario final.

Además, se tiene la expectativa de integrar la aplicación con servicios en la nube en un futuro próximo.

Bibliografía

- [1] 19ª Encuesta Nacional Urbana de Seguridad Ciudadana ENUSC 2022. https://www.ine.gob.cl/docs/default-source/seguridad-ciudadana/publicaciones-y-anuarios/2022/nacional/s%C3%ADntesis-de-resultados-19-enusc-2022---nacional.pdf?sfvrsn=2854ad13_2.
- [2] Crisis de seguridad: Cifras del gobierno revelan explosivo crecimiento de la delincuencia en Chile en 2022 (y el severo riesgo en el norte). <https://www.ex-ante.cl/crisis-de-seguridad-cifras-del-gobierno-revelan-explosivo-crecimiento-de-la-delincuencia-en-chile-en-2022-y-el-severo-riesgo-en-el-norte/>.
- [3] Watsor. <https://github.com/asmirnou/watsor>.
- [4] Machine Vision Security System. <https://github.com/jasondeglin/MaViS>.
- [5] Frigate. <https://frigate.video/>.
- [6] ¿Qué son los microservicios? <https://aws.amazon.com/es/microservices/#:~:text=Los%20microservicios%20son%20un%20enf>.
- [7] ¿Qué es una base de datos? <https://aws.amazon.com/es/what-is/database/#:~:text=Una%20base%20de%20datos%20es,almacenar%2C%20recuperar%20y%20editar%20datos>.
- [8] Qué es BackEnd? <https://rafarjonilla.com/que-es/backend/>.

- [9] Qué es Frontend y Backend: características, diferencias y ejemplos. <https://platzi.com/blog/que-es-frontend-y-backend/>.
- [10] Plataforma digital. <https://www.cognizant.com/es/es/glossary/digital-platform>.
- [11] Qué es una API REST, para qué sirve y ejemplos. <https://blog.hubspot.es/website/que-es-api-rest>.
- [12] ¿Qué es un lenguaje de programación? <https://openwebinars.net/blog/que-es-un-lenguaje-de-programacion/>.
- [13] How can JavaScript revolutionize your backend with the top frameworks. <https://www.softwebsolutions.com/resources/backend-javascript-frameworks.html>.
- [14] Documentación de Express.js. <https://expressjs.com/es/>.
- [15] Los top frameworks para Front End. <https://coderslink.com/talento/blog/los-top-frameworks-para-front-end/>.
- [16] Documentación de React. <https://es.react.dev/>.
- [17] Documentación de PostgreSQL. <https://www.postgresql.org/>.
- [18] MongoDB. <https://www.mongodb.com/es>.
- [19] Documentación de Axios. <https://axios-http.com/es/docs/intro>.
- [20] Documentación de bcrypt. <https://www.npmjs.com/package/bcrypt>.
- [21] Documentación de body-parser. <https://www.npmjs.com/package/body-parser>.
- [22] Documentación de cors. <https://www.npmjs.com/package/cors>.
- [23] Documentación de dotenv. <https://www.npmjs.com/package/dotenv>.

- [24] Documentación de jest. <https://jestjs.io/es-ES/>.
- [25] Documentación de jsonwebtoken. <https://www.npmjs.com/package/jsonwebtoken>.
- [26] Documentación de moment. <https://momentjs.com/>.
- [27] Documentación de mongoose. <https://mongoosejs.com/>.
- [28] Documentación de Mui-Material. <https://mui.com/>.
- [29] Documentación de nodemon. <https://www.npmjs.com/package/nodemon>.
- [30] Documentación de pg. <https://node-postgres.com/>.
- [31] Documentación de sequelize. <https://sequelize.org/>.
- [32] Documentación de sinon. <https://sinonjs.org/>.
- [33] Documentación de supertest. <https://www.npmjs.com/package/supertest?activeTab=readme>.
- [34] Documentación de Docker. <https://www.docker.com/>.
- [35] ¿Qué es el modelo C4? <https://miro.com/es/diagrama/que-es-modelo-c4/>.
- [36] Postman. <https://www.postman.com/>.
- [37] Códigos de Estado HTTP. <https://developer.mozilla.org/es/docs/Web/HTTP/Status>.
- [38] Documentación de GraphQL. <https://graphql.org/>.