

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA E INFORMÁTICA
CONCEPCIÓN - CHILE



**“SOLUCION BACKEND PARA LA PLATAFORMA WEB
DE RAD”**

BENJAMIN ISAAC FUENTES GALLEGOS

**MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO EN INFORMÁTICA**

Profesor Guía: RHODDY ANGEL VIVEROS MUÑOZ

Diciembre - 2024



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

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

Título del trabajo: Solución backend Para la plataforma web de RAD.

Nombre del candidato(a): Benjamin Isaac Fuente Galdames

Carrera / Grado: Ingeniería en informática

Campus: León Concepción Departamento: electrónica e informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Rhaddy Angel Viveros Muñoz, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

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

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

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (marcar una opción):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

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

Fecha: 24/10/2025 Firma: 

Estudiante o Candidato(a):

Fecha: 24/10/2025 Firma: 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

Dedico este trabajo a mi madre y a mi familia, quienes siempre han estado a mi lado brindándome su amor y apoyo incondicional. También dedico este logro a mis colegas y amigos, cuya colaboración y amistad hicieron posible este proyecto. Finalmente, dedico este esfuerzo a aquellas personas que no creyeron en mí, ya que, de cierto modo, fueron una motivación para terminar la carrera y demostrarme lo contrario.

AGRADECIMIENTOS

Quisiera agradecer a todas las personas que fueron parte de este proceso, ya sea ayudándome o apoyándome para seguir adelante en mis estudios. Me gustaría agradecer especialmente a mi madre y a mi familia por su apoyo incondicional hasta el final. También quiero agradecer a mis colegas, quienes fueron compañeros y amigos; sin ellos, este proyecto no habría sido posible.

RESUMEN

Resumen— El backend asume las responsabilidades y se ocupa de la adecuada implementación de la lógica empresarial, además de la preservación de los datos, para posteriormente poder usar estos datos al crear un *frontend* y/o donde se requiera, sin tener en cuenta la tecnología, y sin obstaculizar la producción de este. Se trabajó pensando en que la solución debería incluir atributos de alta calidad, como alta escalabilidad, seguridad (confidencialidad), comunicación con otras tecnologías y habilidad para poner en marcha microservicios.

Para lograr esto, se trabajó con metodologías, estándares y marcos de trabajo que facilitaran la implementación de estas características, como el empleo de Spring Boot, un marco de trabajo con un fuerte respaldo comercial, perfecto para manejar microservicios y/o modelos vistas controladores. La adopción del estándar *APIs RESTful* para la interacción con otros servicios. *JSON Web Token* (RFC 7519) facilita la transferencia de datos entre los participantes en forma de un objeto *JSON* y garantiza la autenticación y autorización de los *endpoints*. MySQL y Workbench para la administración de los datos, Docker y Docker Compose para la puesta en marcha de microservicios y la coordinación de contenedores.

La solución *backend* está desplegada en los servicios *cloud* de AWS, lo que garantiza la disponibilidad, seguridad y escalabilidad necesarias. Además, el *backend* podrá conectarse con el *frontend* y luego desplegarse en internet, asegurando una integración fluida y eficiente.

Palabras Clave— Backend; Escalabilidad; Seguridad; Microservicios; APIs RESTful

ABSTRACT

Abstract— The backend assumes responsibilities and takes care of the proper implementation of business logic, as well as data preservation, to later use this data when creating a *frontend* and/or where required, regardless of the technology, without hindering its production. The solution was designed to include high-quality attributes such as high scalability, security (confidentiality), communication with other technologies, and the ability to implement microservices.

To achieve this, methodologies, standards, and frameworks were used to facilitate the implementation of these characteristics, such as the use of Spring Boot, a framework with strong commercial support, perfect for handling microservices and/or model-view-controller patterns. The adoption of the RESTful APIs standard for interaction with other services, JSON Web Token (RFC 7519) facilitates the transfer of data between participants in the form of a JSON object and ensures authentication and authorization of the *endpoints*. MySQL and Workbench for data management, Docker and Docker Compose for the deployment of microservices and container orchestration.

With this, a quality platform solution was achieved, allowing it to expand in the future without complications.

Keywords— Backend; Scalability; Security; Microservices; RESTful APIs.

GLOSARIO

API: Interfaz de programación de aplicaciones que permite la comunicación entre diferentes sistemas de *software*.

Amazon EC2 (*Elastic Compute Cloud*): Servicio de AWS que proporciona capacidad de computación escalable en la nube.

Autenticación: Proceso de verificar la identidad de un usuario o una aplicación.

Autorización: Proceso de determinar si un usuario o una aplicación tiene permiso para acceder a un recurso.

Backend: Parte del sistema que se encarga de la lógica de negocio y la gestión de datos.

Certificate Manager: Servicio de AWS que facilita la gestión de certificados SSL/TLS.

CIDR (*Classless Inter-Domain Routing*): Método para asignar direcciones IP y enrutamiento IP.

Cloud: Servicios de computación en la nube que permiten almacenar y acceder a datos y programas a través de Internet.

Configuraciones centralizadas: Gestión de configuraciones de manera centralizada para asegurar consistencia y facilidad de administración en un sistema distribuido.

Cortacorrientes: Dispositivo que interrumpe el flujo de corriente eléctrica para prevenir el robo de vehículos.

CORS: Mecanismo de seguridad que permite a los servidores web controlar el acceso a sus recursos desde otros dominios.

CRUD (*Create, Read, Update, Delete*): Operaciones básicas de almacenamiento persistente utilizadas en bases de datos y aplicaciones web.

Elastic Load Balancer (ELB): Servicio de AWS que distribuye automáticamente el tráfico de red entrante entre múltiples instancias.

Endpoint: Punto de entrada a una API donde se pueden realizar solicitudes y recibir respuestas.

ERP: Sistema de planificación de recursos empresariales.

Escalabilidad: Capacidad de un sistema para manejar una cantidad creciente de trabajo o su potencial para ser ampliado para acomodar ese crecimiento.

Facilidad de uso: Grado de simplicidad con el que los usuarios pueden operar un sistema.

Firewall: Dispositivo o *software* que controla el acceso a una red, permitiendo o bloqueando el tráfico de acuerdo con un conjunto de reglas de seguridad predefinidas.

Framework: Conjunto de herramientas y bibliotecas que proporcionan una estructura estándar para el desarrollo de *software*.

Frontend: Parte visual del sistema con la que interactúa el usuario.

Gateway: Punto de entrada que controla el acceso a los servicios de una red.

Gestor de base de datos: Sistema de *software* que utiliza el motor de base de datos para interactuar con los datos, generalmente con una interfaz de usuario.

Grupo de seguridad: Conjunto de reglas de *firewall* que controlan el tráfico entrante y saliente de las instancias de AWS.

HTTP (Hypertext Transfer Protocol): Protocolo utilizado para la transferencia de datos en la web.

HTTP Status Codes: Códigos de estado que indican el resultado de una solicitud *HTTP* (por ejemplo, 200 OK, 201 Created, 204 No Content).

IP: Una dirección *IP* es un número que se asigna a cada dispositivo que está conectado a una red.

JSON (JavaScript Object Notation): Formato de texto ligero para el intercambio de datos.

Licencia: Permiso legal para usar un *software* bajo ciertas condiciones.

Lógica de negocios: Conjunto de reglas y procesos que definen cómo se manejan los datos y se realizan las operaciones en una aplicación.

Lumpen: Término que se refiere a la clase social más baja y marginada, a menudo asociada con actividades delictivas.

Metodologías ágiles: Conjunto de principios y prácticas que fomentan el desarrollo iterativo e incremental, con colaboración constante y adaptación a los cambios.

Modus operandi: Método de operación o forma de actuar de los delincuentes.

Motor de base de datos: *Software* que facilita la generación, acceso, administración y manejo de bases de datos.

ORM: Mapeo Objeto-Relacional, técnica para convertir datos entre sistemas incompatibles usando programación orientada a objetos.

Orquestación de contenedores: Proceso de gestión y coordinación de contenedores en una infraestructura de TI.

Portonazo: Forma de robo violento en el cual el vehículo es despojado al conductor, generalmente con el vehículo encendido.

RAD (Remote Auto Disengage): Dispositivo inventado para este proyecto que inutiliza el vehículo de manera segura.

Rendimiento: Velocidad y eficiencia con la que un sistema realiza tareas.

Rol: Conjunto de permisos y responsabilidades asignados a un usuario dentro de un sistema.

RSA (*Rivest-Shamir-Adleman*): Algoritmo de criptografía de clave pública utilizado para asegurar datos.

SCP (*Secure Copy Protocol*): Protocolo para transferir archivos de manera segura entre hosts en una red.

Soporte: Asistencia técnica y recursos disponibles para ayudar a los usuarios a resolver problemas y aprender a usar el *software*.

SSL (*Secure Sockets Layer*): Protocolo de seguridad que establece un enlace cifrado entre un servidor web y un navegador.

Subredes: Segmentos de una red más grande, utilizados para organizar y gestionar el tráfico de red.

Tablas de enrutamiento: Tablas que determinan la ruta que deben seguir los paquetes de datos en una red.

Target Group: Grupo de recursos *backend* que reciben tráfico de un balanceador de carga.

Token: Cadena de texto generada por un servidor que contiene información de autenticación y autorización.

UI/UX: Diseño de interfaz de usuario y experiencia de usuario.

VPC (*Virtual Private Cloud*): Red privada virtual en la nube que permite la creación de una red aislada dentro de AWS.

INDICE DE CONTENIDOS

RESUMEN.....	4
ABSTRACT	5
INDICE DE TABLAS.....	15
INTRODUCCIÓN	16
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	17
CAPÍTULO 2: MARCO CONCEPTUAL	19
2.1 Producto Web (<i>Backend</i>)	19
2.1.1 Lenguaje de programación y marco de trabajo.....	19
2.1.2 <i>Spring Boot</i>	20
2.1.3 <i>Api RestFul</i>	21
2.1.4 Modelo vista controlador (MVC)	22
2.1.3 Microservicio.....	22
2.1.5 <i>Docker</i>	22
2.1.6 <i>Docker compose</i>	23
2.1.6 Motor y gestor de base de datos.....	23
2.1.6 <i>Spring Security</i>	25
2.1.7 <i>Token y JSON web Token</i>	25
2.1.8 <i>Postman</i>	25
2.1.9 <i>Amazon Web Services</i>	25
2.2 Metodología del proyecto.....	26
2.2.1 Scrum	27
CAPÍTULO 3: PROPUESTA DE SOLUCION	28
3.1 Identificación de requerimientos.....	28
3.2 Diseño de la Arquitectura.	33
3.3 Alcance del proyecto	33

3.4 Gestión de Proyecto y versiones	34
3.5 Aspectos técnicos	34
3.5.1 Modelo vista controlador	34
3.5.2 Microservicio.....	36
3.5.3 <i>Docker</i>	36
3.5.4 Docker Compose	37
3.6 Base de datos	37
3.7 Implementación de la seguridad	39
3.8 <i>Endpoint</i> y controladores.	39
3.8.1 Controlador del usuario	39
3.8.2 Controlador del inventario	40
3.8.3 Controlador de dispositivo	41
3.8.4 Controlador de agenda	42
3.8.5 Controlador de boleta.....	42
3.9 Implementación y puesta en marcha.....	43
3.9.1 implementación de la solución modelo vista controlador (MVC) a la plataforma <i>cloud</i>	43
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	45
4.1 Pruebas de seguridad.....	45
4.1.1 Caso 1: Usuario no autenticado.....	45
4.1.2 Caso 2: Usuario autenticado con el <i>token</i> expirado	46
4.1.3 Caso 3: Usuario autenticado sin autorización	47
4.1.4 Caso 4: Usuario modifica el <i>token</i>	48
4.2 Pruebas de comunicación (Microservicios)	51
4.3 Prueba de implementación a <i>cloud</i>	53
CAPÍTULO 5: CONCLUSIONES	55
5.1 objetivos realizados.....	55

5.2 Trabajo a futuro.....	56
---------------------------	----

INDICE DE FIGURAS

Figura 1: <i>Lean Canvas RAD</i> . Fuente: Elaboración propia.	28
Figura 2: Caso de uso Persona natural Fuente: Elaboración propia.	29
Figura 3: Caso de uso Estadista Fuente: Elaboración propia.....	30
Figura 4: Caso de uso Operador Fuente: Elaboración propia.	31
Figura 5: Caso de uso instalador Fuente: Elaboración propia.....	32
Figura 6: Caso de uso Administrador Fuente: Elaboración propia.....	32
Figura 7: Arquitectura AWS. Fuente: Elaboración propia.	33
Figura 8: Entidad relación del sistema. Fuente: Elaboración propia.....	37
Figura 9: Test de seguridad usuario no autenticado. Fuente: Elaboración propia.	45
Figura 10: Test de seguridad usuario con token expirado. Fuente: Elaboración propia.	46
Figura 11: Mensaje de seguridad usuario con token expirado. Fuente: Elaboración propia.	46
Figura 12: Test de seguridad usuario con token expirado. Fuente: Elaboración propia	47
Figura 13: Login del Usuario Matías. Fuente: Elaboración propia	47
Figura 14: Token del usuario Matías sin modificar. Figura: Elaboración propia.	48
Figura 16: Test de seguridad usuario Matías con <i>token</i> modificado Fuente: Elaboración propia.....	49
Figura 15: Token del usuario Matías modificado. Figura: Elaboración propia.....	49
Figura 17: Mensaje de seguridad usuario Matías con <i>token</i> expirado Fuente: Elaboración propia.....	50
Figura 19: Datos de los usuarios (<i>msvc-auth</i>) Fuente: Elaboración propia.....	51
Figura 18: Contenedores funcionando (<i>Docker-Compose</i>) Fuente: Elaboración propia	51
Figura 20: Datos de los inventarios (<i>msvc-inventory</i>) Fuente: Elaboración propia	52
Figura 21: Pruebas de comunicación (<i>msvc-invoice</i>) Fuente: Elaboración propia.....	52
Figura 23: Acceso al sistema con el Usuario Edison Fuente: Elaboración propia.	53
Figura 22: Configuración y datos de la instancia AWS Fuente: Elaboración propia.	53

Figura 24: Prueba al <i>endpoint inventory</i> Fuente: Elaboración propia.	54
---	----

INDICE DE TABLAS

Tabla 1: Tabla comparativa de lenguajes y marcos de trabajos Fuente: Elaboración propia.	19
Tabla 2: Tabla comparativa motores de bases de datos. Fuente: Elaboración propia.	24
Tabla 3: tabla comparativa de metodologías Fuente: Elaboración propia.	26

INTRODUCCIÓN

El problema ha tratado con los delitos de *portonazos* y encerronas, que en el país han aumentado desde 2021 y se han mantenido sobre los 7.000 casos desde entonces, sin promesas de disminuir este año (Portal CEAD, 2024).

Además, hay problemas con los datos de los autos robados, debido a varios factores, como si el vehículo está asegurado o no y si se denunció el hecho o no. La Asociación de Aseguradores de Chile (AACH) indica que hubo un total de 14.297 autos robados (Cooperativa.cl, 2024). En contraste, el Centro de Estudios y Análisis del Delito (CEAD) reporta un total de 10.688 robos de vehículos con violencia (Portal CEAD, 2024), y según Carabineros, hay un total de 10.601 robos de vehículos violentos (Ortuya, 2024).

Para esto, se plantea la elaboración de un dispositivo que quemará los fusibles del vehículo en el que está instalado, con el objetivo de dejar el móvil inutilizado y frustrar el robo ya consumado. Esto permitirá que el usuario recupere su bien sustraído y mejorará los índices de recuperación de vehículos robados, que actualmente no superan el 50% del total de autos robados en algunas regiones (Delegación Presidencial Provincial de Arauco, 2024).

Para resolver la gran diferencia en la analítica sobre los robos de vehículos y reflejar de la mejor manera posible la realidad, se recogerán los datos derivados del funcionamiento y las configuraciones del dispositivo y del sistema.

Esta información permitirá disponer de datos para instituciones como el Estado, las policías y distintas entidades que trabajan con este tipo de información, ya sea de forma personalizada.

El *backend* cumplirá la función de ser parte de la plataforma digital mediante la cual se llevará a cabo la operación de control y gestión del sistema *Remote Auto Disengage* (RAD), asignando prioridades a los colaboradores y llevando registro de horas agendadas para instalación. Además, gestionará el inventario, así como los dispositivos existentes, la autenticación de los usuarios, la tokenización de la sesión para mayor seguridad y las boletas de los usuarios (no contables). También gestionará la data entrante mediante *API REST*, aplicará la lógica de negocios y establecerá una comunicación con la base de datos para la persistencia de los datos.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

El robo de autos en Chile es un delito que ha experimentado un aumento sostenido durante los años 2022 y 2023, pasando de 12.901 vehículos robados en 2022 a 10.688 en 2023 (Portal CEAD, 2024). Este incremento en la persistencia del delito está relacionado con el modus operandi conocido como “portonazo”. Se trata de una forma violenta de delinquir, en la cual el vehículo es arrebatado al conductor, ya sea mientras está en movimiento o con las llaves puestas, y generalmente con el motor encendido. Este método de operación reduce significativamente la eficacia de la mayoría de los dispositivos de seguridad, ya que el vehículo se encuentra en funcionamiento.

Para enfrentar esta problemática, las personas recurren a métodos preventivos como alarmas, cortacorrientes y algunas técnicas poco ortodoxas —por ejemplo, ocultar el vehículo bajo lonas o estacionarlo en lugares poco transitados para evitar ser detectado—, además de guardarlo en estacionamientos cerrados o recintos privados destinados a su resguardo. Estos medios resultan relativamente efectivos para prevenir el delito, pero fallan por completo cuando se trata de frustrarlo una vez consumado. Es en ese punto donde se plantea la intervención mediante la creación de un dispositivo capaz de inutilizar el vehículo tras ser robado.

Esta solución aumentará significativamente las probabilidades de recuperación del automóvil y, con el tiempo, contribuirá a desincentivar el robo de vehículos, volviéndolo una actividad poco rentable. El esfuerzo requerido para cometer el delito y el riesgo de no obtener resultados harán que esta práctica se torne frustrante para los delincuentes.

Actualmente, existen diversos dispositivos en el mercado que permiten apagar el vehículo de forma remota mediante aplicaciones móviles conectadas por red Wi-Fi, además de sistemas preventivos como alarmas, cortacorrientes y otros similares, cada uno con sus limitaciones. Sin embargo, ninguno de estos dispositivos opera bajo el mismo principio que RAD, ya que la mayoría se enfocan en la prevención del delito, mientras que RAD incorpora un enfoque reactivo, diseñado para actuar una vez que el robo ha sido consumado.

Una de las principales debilidades de los sistemas convencionales es que, si el vehículo ha sido sustraído con éxito, el delincuente puede volver a encenderlo sin mayores obstáculos, ya sea porque logró forzar el sistema de encendido o porque dispone de las llaves. Además, muchos de estos dispositivos ya han sido ampliamente estudiados por quienes cometen estos delitos, lo que ha permitido que sean vulnerados o desactivados con relativa facilidad.

Nuestro enfoque reactivo al robo supone que el usuario de RAD estará a una distancia segura de los delincuentes, lo que permite evitar represalias inmediatas y aumenta significativamente la posibilidad de recuperar el vehículo tras el robo. El sistema actúa dañando partes específicas del automóvil de forma segura y deliberada, facilitando una reparación rápida y económica.

Además, se detectó la necesidad de crear una plataforma en línea que facilite la administración y control del sistema RAD de forma eficaz. Este necesitará el desarrollo del *backend*, que es la sección del sistema que gestiona la lógica empresarial, la administración de datos y la interacción con la base de datos. El propósito principal del *backend* será satisfacer los casos de uso que se presenten, garantizando que todas las funcionalidades necesarias estén accesibles y funcionando.

Uno de los aspectos críticos del desarrollo del *backend* será la implementación de medidas de seguridad robustas. Esto abarca la autenticación, que asegura que únicamente los usuarios autorizados tengan acceso al sistema, y la autorización, que gestiona los permisos y grados de acceso de cada individuo. Estas acciones son fundamentales para salvaguardar la información delicada y preservar la integridad del sistema.

Adicionalmente, se elaborará una solución fundamentada en el modelo de diseño MVC (Modelo-Vista-Controlador). Este modelo de diseño distingue la lógica empresarial (Modelo), la interfaz de usuario (Vista) y la gestión del flujo de aplicaciones (Controlador), lo que simplifica el mantenimiento y la expansión del sistema. La implementación de MVC posibilitará una estructura más ordenada y flexible, favoreciendo actualizaciones y mejoras futuras.

Otro objetivo importante será la migración a una arquitectura de microservicios. Los microservicios segmentan la aplicación en pequeños servicios autónomos que se comunican entre ellos. Esta estructura brinda múltiples beneficios, tales como una mayor adaptabilidad, escalabilidad y resistencia. Cada microservicio tiene la capacidad de ser desarrollado, desplegado y escalado de forma autónoma, incrementando así la eficiencia y la capacidad de respuesta del sistema.

Finalmente, esta solución se pondrá en marcha en una plataforma de *cloud* para propagarla a través de internet. La utilización de una plataforma en la nube ofrecerá múltiples ventajas, tales como alta disponibilidad, escalabilidad automática y mejora en la seguridad. La capacidad para incrementar los recursos en función de la demanda facilitará la gestión de picos de tráfico sin perjudicar el desempeño del sistema. Además, la infraestructura *cloud* proporcionará sistemas de seguridad avanzados para salvaguardar la información y asegurar la continuidad del servicio.

CAPÍTULO 2: MARCO CONCEPTUAL

La base conceptual de nuestra propuesta se fundamenta en diversas disciplinas y tecnologías que convergen para dar solución al problema del robo de autos en Chile. En esta sección, se describen los principales conceptos técnicos, metodologías y herramientas involucradas en la creación del dispositivo RAD, proporcionando un marco claro y unificado para entender la memoria.

2.1 Producto Web (*Backend*)

Las herramientas y metodologías que se usaron y que se usarán para implementar nuestro producto *web* se pensaron para hacer una implementación que contenga las siguientes características:

- Producto altamente escalable.
- Producto que contemple seguridad (confidencialidad).
- Producto que pueda trabajar con otras tecnologías sin importar el lenguaje.
- Producto en el que se puedan implementar microservicios.
- Herramienta que soporte proyectos de gran envergadura.

2.1.1 Lenguaje de programación y marco de trabajo

En este apartado, se ha considerado cuál sería el lenguaje de programación y marco de trabajo óptimo para trabajar en la solución web. Se evaluaron varios y se analizaron las ventajas y desventajas que presenta cada uno de ellos:

Tabla 1: Tabla comparativa de lenguajes y marcos de trabajos

Fuente: Elaboración propia.

Característica	Python Django	Python Flask	Java Spring Boot	JavaScript Next.js
Facilidad de aprendizaje	Alta	Alta	Baja	Media
Adecuado para aplicaciones grandes	Limitado	Limitado	Alta	Limitado
Creación de APIs	Soportado, pero no es el enfoque principal	Muy bueno, enfoque principal	Excelente, muy buen soporte	Muy bueno
Escalabilidad	Moderada	Moderada	Alta	Moderada
Soporte para microservicios	Limitado	Bueno	Excelente	Bueno

Documentación	Buena	Buena	Excelente	Buena
Seguridad	Alta, incluye medidas integradas como <i>CSRF</i> y <i>XSS protection</i>	Moderada, requiere más configuración manual	Alta, con soporte robusto para seguridad	Moderada, requiere cuidado adicional para seguridad

Nota: A la hora de elaborar la tabla se tomaron como referencia (Unbiased Coder., 2024), (Framework Training., 2024), y (Next.js, 2024).

Teniendo esto en cuenta, se decidió el lenguaje para trabajar en la solución. El lenguaje y marco de trabajo elegido es *Java Spring Boot*, debido a que se ajustaba más a los puntos requeridos para el producto. Además, ya se tenía experiencia previa trabajando con este marco de trabajo.

2.1.2 Spring Boot

Java Spring Boot es un *software* que agiliza y simplifica la creación de microservicios y aplicaciones web con *Spring Framework*, gracias a tres funciones fundamentales:

- Configuración automática
- Un enfoque de configuración obstinado
- La capacidad de crear aplicaciones autónomas

Estas características, combinadas, conforman una herramienta que le permite configurar una aplicación basada en Spring con el mínimo de instalación y configuración (IBM, 2024).

Spring Boot es fundamental para el desarrollo de la solución debido a su alta escalabilidad y fácil implementación del código, además de estar fuertemente documentado.

La elección de este lenguaje (*Java*) y del marco de trabajo (*Spring Boot*) fue por las siguientes razones:

- Configuración Automática y Rápida: este marco de trabajo te permite crear aplicaciones listas para producción sin requerir una configuración extensa. La capacidad de configurar automáticamente beans, propiedades y dependencias ahorra tiempo y reduce los errores.
- Esta fuertemente documentado: la documentación de Spring Framework, que es la base de Spring Boot, ofrece una documentación extensa, además de una comunidad extensa y soporte activo, facilitando la solución a problemas complejos.

- Modularización y escalabilidad: otro punto fuerte es que la metodología promueve una arquitectura basada en capas (controladores, servicios, repositorios, modelos), lo que mejora la organización y el mantenimiento del código, haciéndolo una herramienta robusta e ideal para proyectos a gran escala.
- Crear APIs RESTful es fácil: el marco provee una solución fácil y rápida para la creación de APIs mediante anotaciones, para así tener un diseño limpio y mantenible, y con un soporte nativo para JSON.
- Aplicaciones autónomas: al crear aplicaciones ejecutables (a la hora de compilar) como JAR, este le asigna servidores embebidos como Tomcat, eliminando la necesidad de asignar servidores externos y facilitando la implementación y despliegue a entornos de producción, servicios cloud o en contenedores de Docker.
- Compatibilidad: Es compatible con tecnologías modernas como contenedores (Docker, Kubernetes), bases de datos no relacionales (MongoDB, Cassandra) y entornos en la nube (AWS, Azure, GCP).
- Lenguaje Java: Es un lenguaje multiplataforma, robusto y orientado a objetos, con un excelente soporte empresarial y una amplia base de desarrolladores experimentados.

2.1.3 Api RestFul

Para la comunicación de estos servicios hacia el exterior, ocuparemos la tecnología *API RESTful*, ya que, independientemente de la tecnología, herramienta o marco de trabajo que se utilice a la hora de consumir estos datos, no se verá afectada la *API* y puede ser entendida por todos.

Una *API RESTful* es muy escalable ya que separa el cliente del servidor, lo que facilita la escalabilidad, esencial para construir proyectos de gran envergadura. Es flexible e independiente a la hora de hacer cambios, permitiendo modificaciones sin preocuparse por el *frontend*, ya que puede estar en servidores distintos. Además, se puede implementar seguridad utilizando autenticación y autorización para proteger rutas o datos sensibles.

Un *API* es un conjunto de funcionalidades o recursos que un sistema nos proporciona para poder interactuar con él desde otro sistema, independientemente de los lenguajes de programación utilizados en cada uno, la tecnología o cualquier otro aspecto (Martínez, 2021).

Por otra parte, *REST*, que proviene de Transferencia de Estado Representacional, es un estilo de arquitectura de *software* destinado a llevar a cabo una comunicación entre clientes y servidores. El protocolo *HTTP* se utiliza para la comunicación con el servidor, y los mensajes enviados y recibidos pueden ser en formato *XML* o *JSON*. (Martínez, 2021)

Por lo tanto, en términos básicos, una *API RESTful* es una *API* creada utilizando la arquitectura *REST* (Martínez, 2021).

2.1.4 Modelo vista controlador (MVC)

Para empezar el proyecto y entregar productos mínimos viables (*MVPs*) rápidos y funcionales, la solución se va a trabajar primeramente como modelo vista controlador (*MVC*). Esto se debe a que es un patrón ampliamente reconocido a la hora de separar responsabilidades en aplicaciones de software, lo que facilita la organización del código y mejora la mantenibilidad.

MVC es una arquitectura de *software* que se utiliza para categorizar la información, la lógica del sistema y la interfaz que se muestra al usuario. En esta arquitectura se encuentra un sistema central o controlador que administra los ingresos y egresos del sistema, uno o varios modelos responsables de buscar los datos e información requeridos y una interfaz que presenta los resultados al usuario final. Es muy utilizado en el desarrollo web debido a que al necesitar interactuar con varios lenguajes para construir una página web, es muy sencillo causar confusión entre cada elemento si no se separan correctamente. Este patrón facilita la modificación de cada uno de sus elementos sin la necesidad de impactar en los demás (García, 2017).

2.1.3 Microservicio

La ventaja de implementar nuestra solución con microservicios es la alta escalabilidad, la independencia de los componentes y la disponibilidad del servicio, ya que, si uno de estos servicios falla, no comprometerá el funcionamiento del sistema completo.

De esta manera, podemos programar funcionalidades y/o módulos totalmente independientes sin afectar al producto que está en funcionamiento.

El microservicio es un método de desarrollo de *software* que segmenta una aplicación en pequeños servicios autónomos, cada uno opera en su propio procedimiento y puede ser implementado, expandido y mantenido de manera autónoma. Es importante resaltar que el concepto de microservicios hace referencia tanto a un estilo arquitectónico como a una forma de programar programas informáticos (whitestack, 2023).

2.1.5 Docker

Para soportar la implementación y el despliegue de estos microservicios respetando la arquitectura, usaremos *Docker* con la finalidad de contener estos servicios y que funcionen en un entorno ideal y separado, para así enviar estos servicios a producción para su implementación.

Además, podremos supervisar el funcionamiento de cada contenedor, gestionar las fallas y respaldar los datos en volúmenes.

Docker es una plataforma de *software* libre que facilita a los programadores la automatización del despliegue, escalado y administración de aplicaciones mediante la tecnología de contenerización. Ofrece un método estandarizado de agrupar *software* y sus dependencias en una única unidad denominada contenedor (Andrus, 2024).

Los contenedores son entornos ligeros y autocontenidos que incluyen todo lo que una aplicación requiere para funcionar: incluyen el sistema operativo, el código, el tiempo de ejecución, herramientas del sistema, bibliotecas y ajustes. Ofrecen un método uniforme y repetible para la implementación de aplicaciones en diversos contextos, desde el desarrollo hasta la prueba y la producción (Andrus, 2024).

2.1.6 Docker compose

Para soportar la implementación y la orquestación de estos microservicios respetando la arquitectura definida, utilizaremos *Docker Compose*. Esta herramienta nos permitirá definir y gestionar múltiples contenedores de *Docker* como un sistema único, facilitando el despliegue, la configuración y la interconexión entre los servicios.

Mediante un archivo de configuración *YAML*, podremos declarar todos los servicios, sus imágenes, redes, volúmenes y variables de entorno necesarios. Esto garantiza que la implementación sea reproducible y consistente entre los distintos entornos, desde desarrollo hasta producción.

Docker Compose no solo simplifica la puesta en marcha de entornos complejos, sino que también permite escalar servicios de manera sencilla, gestionar fallas y aislar dependencias. Además, su integración con volúmenes permite respaldar y conservar los datos, asegurando la persistencia incluso en casos de fallos del sistema.

Entonces para sintetizar, *Docker Compose* es una herramienta versátil que facilita la definición y administración de aplicaciones multi-contenedor. Donde es posible detallar la configuración del ambiente de desarrollo en un documento *YAML*, especificando los servicios, volúmenes y redes requeridos para tu aplicación, Así poder crear y ejecutar todos los contenedores establecidos en tu archivo de configuración con un solo comando (Imagina, 2024).

2.1.6 Motor y gestor de base de datos

El *software* que facilita la generación, acceso, administración y manejo de bases de datos se conoce como *motor de base de datos*. Se ocupa de realizar consultas, llevar a cabo operaciones de lectura y escritura, y asegurar la integridad y protección de la información. Modelos de motores de base de datos como *MySQL*, *PostgreSQL*, *SQLite* y *Oracle* son algunos ejemplos. Por otra parte, el *gestor de base de datos* es el sistema de *software* que utiliza el motor de base de datos para interactuar con los datos, y por lo general contiene una interfaz de usuario que permite a los administradores y desarrolladores trabajar con las bases de datos de manera más eficiente.

Existen múltiples opciones en la actualidad, por lo que se comenzó a evaluar las opciones a través de estos puntos:

- **Escalabilidad:** Debe de tener una alta escalabilidad.
- **Licencia:** La licencia debe de ser gratuita.

- **Rendimiento:** debe de tener una buena velocidad en las transacciones de lectura.
- **Facilidad de uso:** No debe de ser muy complejo de manejar.
- **Soporte:** Contener un buen soporte y comunidad para aprender de este.

Tabla 2: Tabla comparativa motores de bases de datos.**Fuente: Elaboración propia.**

Característica	MySQL	PostgreSQL	MariaDB	SQL Server
Escalabilidad	Alta	Muy alta	alta	Muy alta
Licencia	<i>Open Source (GPL)</i>	<i>Open Source (PostgreSQL)</i>	<i>Open Source (GPL)</i>	Comercial, versión gratuita limitada (<i>Express</i>)
Rendimiento	Muy Bueno	Excelente en operaciones complejas	Muy Bueno	Excelente, especialmente en entornos empresariales
Facilidad de uso	Fácil de configurar y usar	Complejo pero potente	Fácil de usar (similar a MySQL)	Relativamente sencillo con GUI, pero complejo en configuraciones avanzadas
Soporte	Amplio soporte comunitario	Amplio soporte comunitario	Amplio soporte comunitario	Soporte comercial robusto, gran comunidad

Nota: A la hora de realizar la tabla se tuvo como referencia (db-engines, 2024) y (Agarwal, 2024).

Viendo la comparativa, se decidió trabajar con el motor de MySQL y con su respectivo gestor MySQL Workbench, ya que se ajustaba más a los puntos e intereses, además de contar con experiencia previa trabajando con este gestor.

2.1.6 Spring Security

Spring Security es una dependencia de *Spring Boot* y nos entrega un *framework* de todas las configuraciones de seguridad de las *APIs*, *endpoints*, *CORS* y la aplicación en general.

Spring Security es un marco de trabajo que ofrece autenticación, autorización y seguridad para aplicaciones *Java Spring Boot*. Es sumamente personalizable tanto en su aspecto de autenticación como en el control de acceso para los usuarios, dado que puede ser ampliado con facilidad para satisfacer las necesidades que se requieran (Aviles, 2022).

2.1.7 Token y JSON web Token

A la hora de implementar seguridad, se pensó que podíamos implementar un sistema para que nuestra aplicación tuviera autenticación y autorización de una manera efectiva. Para esto existen los *tokens*, que son una cadena de texto generada por un servidor que contiene información de autenticación y autorización. Los *tokens* son usados para validar la identidad de un usuario o una aplicación, permitiendo acceso a recursos protegidos sin necesidad de enviar repetidamente las credenciales del usuario.

Para implementar *tokens*, se decidió trabajar con el estándar *JSON Web Token (JWT)*, definido en el RFC 7519, ofrece un procedimiento para comunicar de forma segura la identidad de un usuario entre dos partes. Este procedimiento comprende una serie de reivindicaciones o privilegios, que se codifican en objetos *JSON* y se integran en el núcleo de un mensaje que contiene una firma digital para asegurar su autenticidad (Magaña, 2020).

2.1.8 Postman

Cuando se tendrá que realizar pruebas a las *APIs* y validar el funcionamiento de estos, necesitaremos una herramienta que permita hacer pruebas de funcionamiento, que tenga la posibilidad de automatizar estos procesos, que sea intuitivo y permita visualizar las peticiones, además de poder realizar peticiones *REST*.

Es en este punto donde surge Postman, que es una plataforma que, en esencia, nos facilita la realización de peticiones de forma sencilla para examinar *APIs REST* propias o de terceros y más importante de todo es que es gratis (Muradas, 2019).

2.1.9 Amazon Web Services

Amazon Web Services (AWS) es una plataforma de informática en la nube que proporciona una extensa variedad de servicios que posibilitan a las compañías la creación, puesta en marcha y expansión de productos de forma eficaz. *AWS* ofrece servicios tales como almacenamiento, bases de datos, redes, aplicaciones móviles y herramientas de inteligencia artificial, lo que simplifica la creación y administración de aplicaciones en la nube sin necesidad de hacer una inversión en infraestructura (Gimenez, 2020).

Esto representa un beneficio tanto para pequeñas y medianas empresas que aún no tienen la capacidad de invertir considerablemente en infraestructura, como para aquellos que, a pesar de poder hacer la inversión, optan por posponerla u optan por un sistema de pago por uso, parecido a los beneficios entre la adquisición de un vehículo y el arrendamiento de una empresa. Así podrán transformarlo en un costo variable que solo se incrementará si requieren más capacidad, o sea, si el negocio se expande (Gimenez, 2020).

Con esta solución informática, implementaremos el sistema para ponerlo en producción, almacenarlo y hacerlo público, permitiendo que las personas accedan a nuestra solución informática a través de Internet.

2.2 Metodología del proyecto

A la hora de empezar el proyecto se pensó en como deberíamos llevar a cabo la implementación y la ejecución de los objetivos, por lo cual se tenía que escoger una metodología que se ajustase a nuestras necesidades del proyecto.

El proyecto contaba con un límite de tiempo corto y establecidos los alcances de este, se aclaró que deberíamos pensar en una metodología que se adapte a:

- Proyecto corto con una duración corta de tiempo.
- Mostrar *MVPs* del sistema rápidos.
- Proyecto con pocos recursos humanos (tamaño del equipo).
- Que se adapte a los cambios rápido.

Viendo estas necesidades, se tuvo que decidir qué metodologías usar. Entre ellas se consideraron las siguientes:

Tabla 3: tabla comparativa de metodologías
Fuente: Elaboración propia.

Metodologías	Descripción	Caso de Uso	Desventajas
Cascada	Modelo secuencial donde cada fase (requisitos, diseño, implementación, pruebas, despliegue) debe completarse antes de pasar a la siguiente.	Proyectos con requisitos claros y bien definidos desde el inicio.	- Poco flexible ante cambios. - Difícil retroceder a fases anteriores. - Alto riesgo de descubrimiento tardío de problemas.

Agile	Conjunto de principios que fomentan el desarrollo iterativo e incremental, con colaboración constante y adaptación a los cambios.	Proyectos que requieren flexibilidad y entregas continuas de valor.	- Puede ser difícil de implementar sin una buena cultura de equipo. - Requiere experiencia para manejar cambios frecuentes.
Scrum	Marco ágil que organiza el trabajo en <i>sprints</i> (iteraciones cortas), con roles definidos (<i>Scrum Máster</i> , <i>Product Owner</i> , Equipo de Desarrollo).	Proyectos con entregas incrementales, especialmente con equipos pequeños y plazos cortos.	- Puede ser difícil de adoptar sin formación adecuada. - Reuniones diarias pueden ser vistas como una carga si no se manejan bien.
Extreme Programming (XP)	Metodología ágil que enfatiza la entrega frecuente de <i>software</i> funcional, con prácticas como programación en pareja, pruebas continuas, y retroalimentación frecuente.	Proyectos con requisitos cambiantes que se benefician de entregas rápidas y una alta calidad de código.	- Puede requerir un cambio cultural significativo. - Programación en pareja y otras prácticas pueden ser difíciles de aceptar.
Kanban	Sistema visual de gestión de trabajo que utiliza tarjetas para representar tareas, enfocándose en la entrega continua.	Proyectos con flujos de trabajo continuos, ideal para mantenimiento o desarrollo continuo.	- Puede ser menos estructurado que otras metodologías, lo que puede llevar a problemas de gestión del tiempo y de las prioridades.

Nota: A la hora de realizar la tabla se tuvo como referencia (Eby, 2017), (Burak, 2024).

Una vez vista la comparativa, se tomó la decisión de que la metodología que más se adapta es Scrum, además de contar con experiencia previa trabajando con Scrum.

2.2.1 Scrum

Scrum es un enfoque de trabajo ágil que permite a los individuos enfrentar problemas complejos adaptativos mientras se proporcionan productos de manera

eficiente y creativa con el máximo valor. Por lo tanto, *Scrum* es una metodología que asiste a los equipos a cooperar y llevar a cabo una labor de gran impacto. La metodología *Scrum* ofrece un esquema de valores, roles y directrices para asistir al equipo a enfocarse en la iteración y el perfeccionamiento constante en proyectos de gran complejidad. Además, debes tener en cuenta que en *Scrum* se colabora con grupos pequeños multidisciplinares en ciclos de investigación enfocados en el cliente y se desarrolla un producto de manera progresiva (Martins, 2024).

CAPÍTULO 3: PROPUESTA DE SOLUCION

3.1 Identificación de requerimientos.

Para la identificación de requerimientos, los pasos clave que se identificaron fueron los de clientes y/o actores que interactuarán con el sistema, para satisfacer las necesidades que tendrá cada uno. Para este paso, se tomó como referente el “Lean Canvas” (véase la figura 1), ya que estos usuarios serán posibles usuarios finales del sistema.

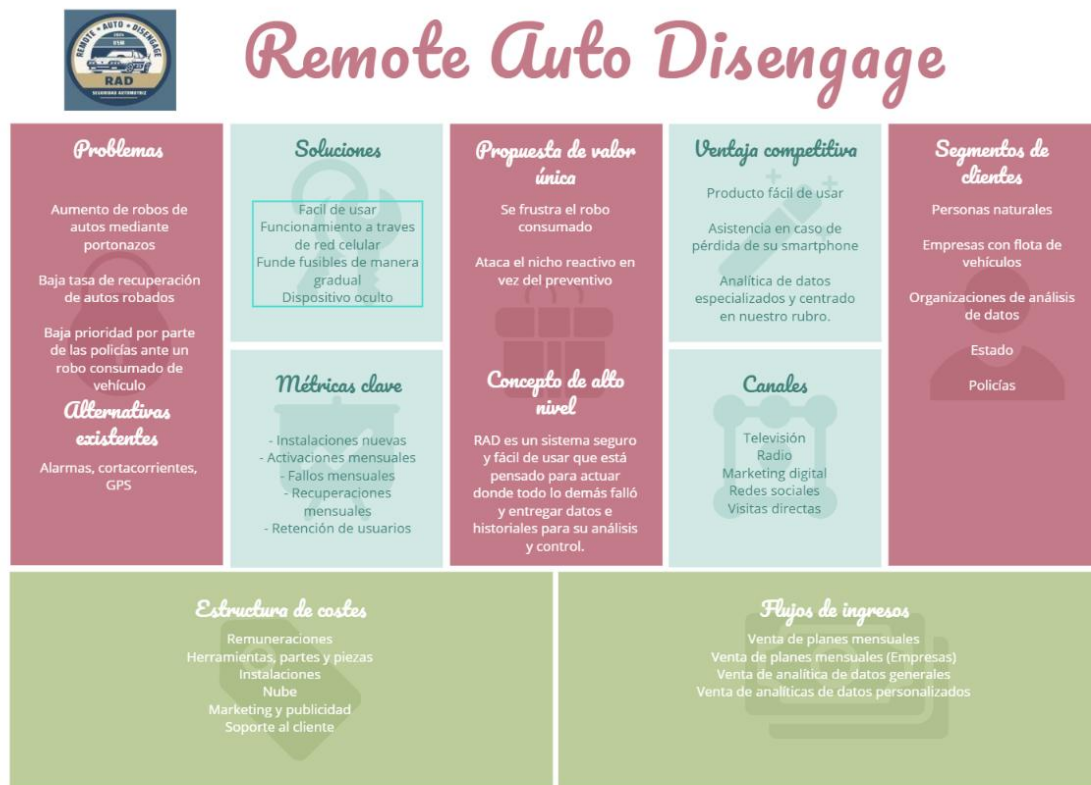


Figura 1: Lean Canvas RAD.

Fuente: Elaboración propia.

A continuación, se identificarán los actores y se verán sus casos de uso a nivel general por cada actor y se realizara una breve descripción de este:

Usuarios Común: Es una persona natural con un vehículo o una Empresa con una flota de vehículos que le han puesto al vehículo el dispositivo RAD. El desea saber el listado de sus vehículos, el detalle de sus vehículos, el estado del dispositivo (Activado o Desactivado) y el plan que tiene su dispositivo, además de la fecha de pago. En relación con el *Canvas* este actor incluye a la empresa y persona natural. Siendo su caso de uso:

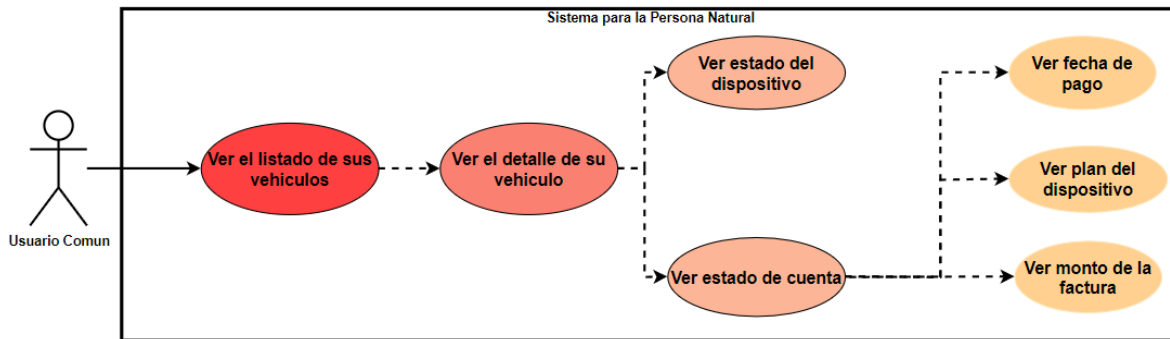


Figura 2: Caso de uso Persona natural

Fuente: Elaboración propia.

Estadista: El estadista es la persona que quiere consumir la analítica que se ha recopilado con el dispositivo, el desea ver todo el análisis que se ha realizado ya sea en mapa o en gráficos y hacer consultas. En relación con el *Canvas* este actor incluye Organizaciones de análisis de datos, Estado y las Policías. Siendo su caso de uso:

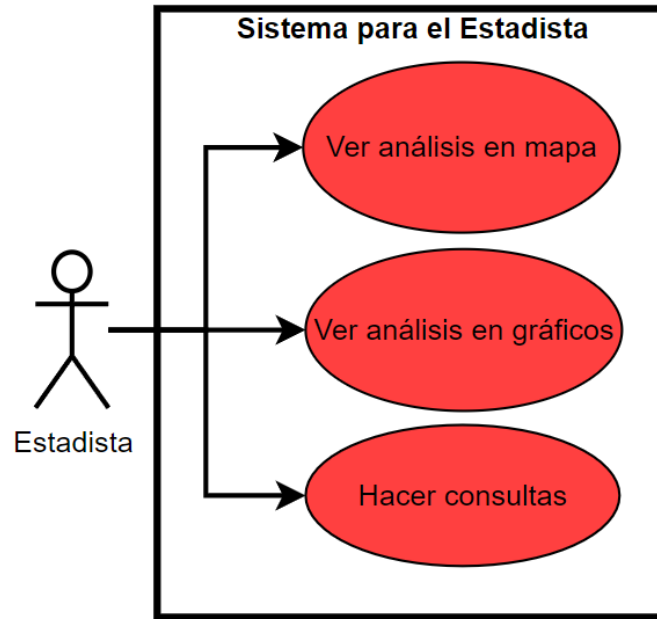


Figura 3: Caso de uso Estadista

Fuente: Elaboración propia.

Operador: El operador es quien va a asesorar o guiar al cliente, para la compra del dispositivo, pues tiene un rol más ejecutivo, El desea saber las plantas de instalación que están operativas y el sector de estas, ver los horarios disponibles para agendar, agendar hora, modificar las horas agendadas, ver los dispositivos disponibles para esa planta, crear una cuenta al cliente y editar el detalle del auto del cliente. Este actor es inherente del modelo de negocio. Siendo su caso de uso:

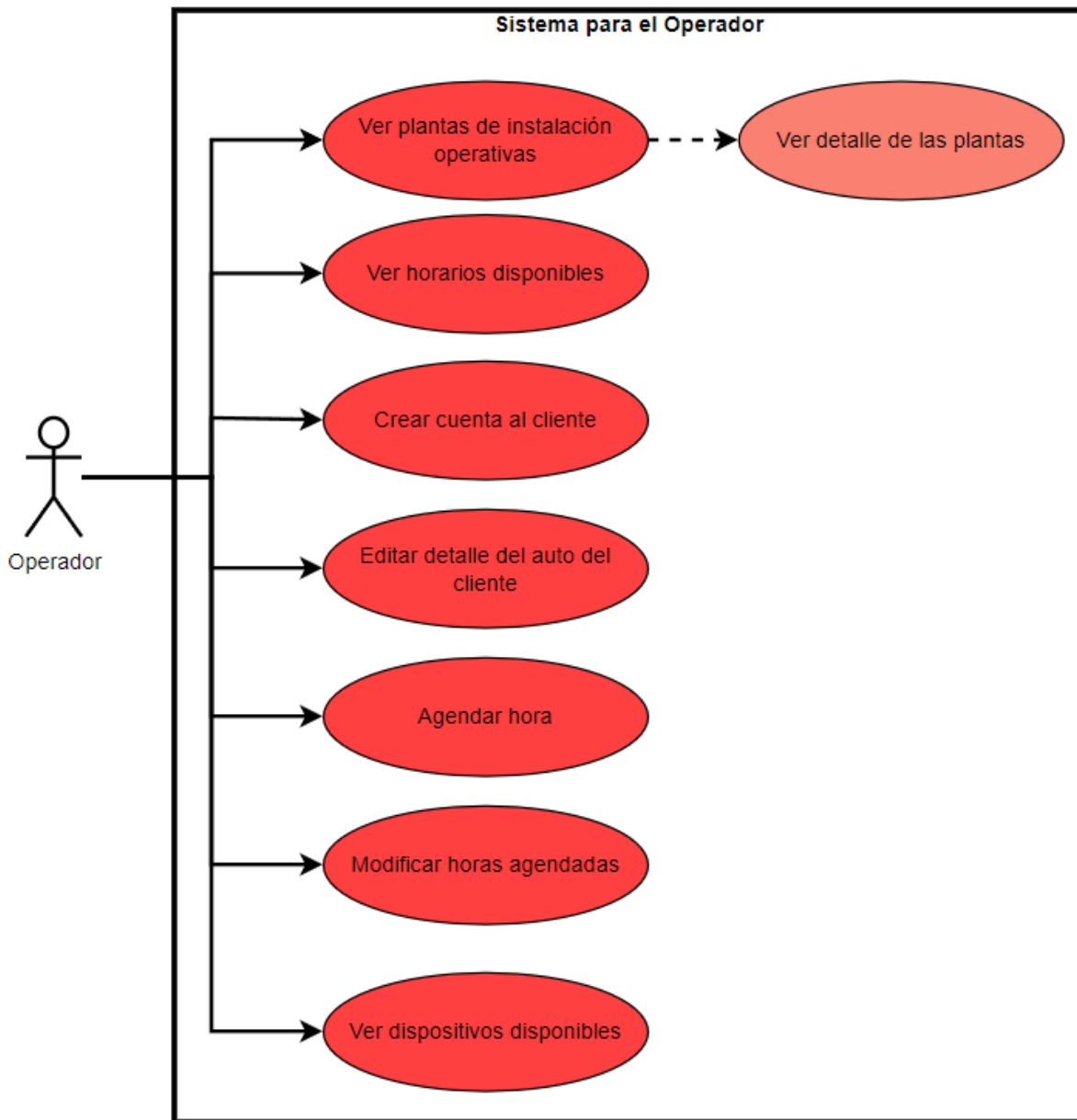


Figura 4: Caso de uso Operador
Fuente: Elaboración propia.

Instalador: El instalador es quien va a instalar el dispositivo rad, El desea ver su agenda para ver las horas que tiene agendadas, además de modificar el estado de la agenda, y el estado de instalación, además de agregar nuevos dispositivos al inventario, modificar el estado del dispositivo en el inventario. Este actor es inherente del modelo de negocio. Siendo su caso de uso:

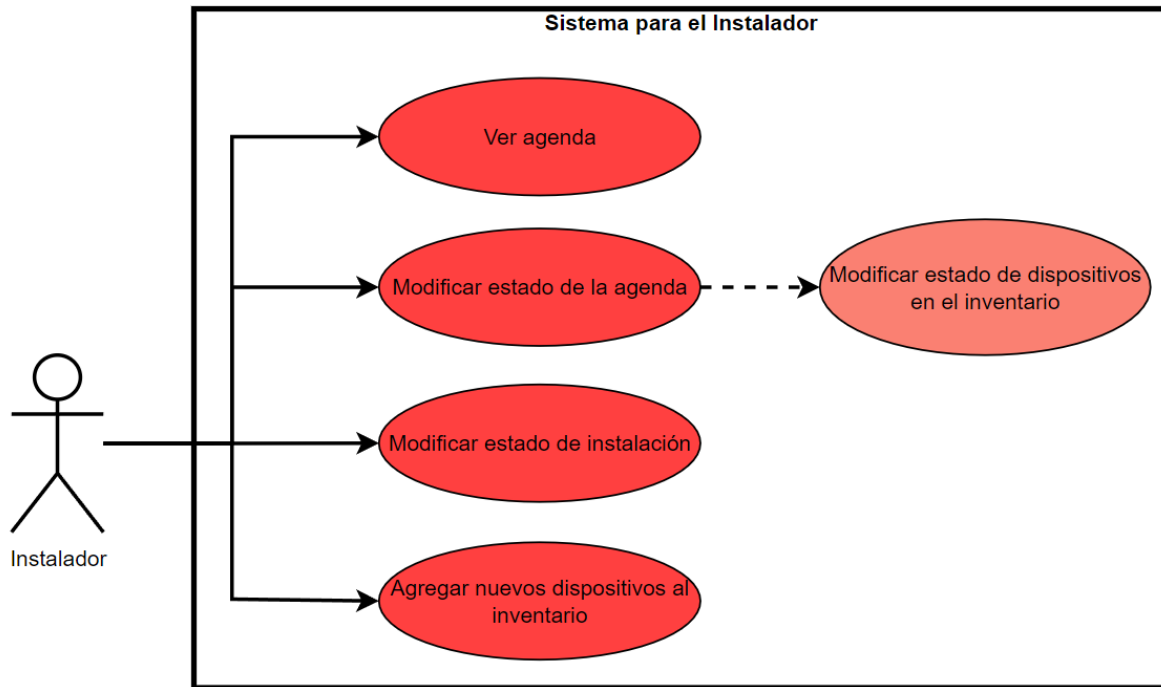


Figura 5: Caso de uso instalador

Fuente: Elaboración propia.

Administrador: El instalador es inherente a cualquier sistema, pues es quien debe controlar la plataforma, además de poder tener acceso a todos los recursos del sistema. El administrador debe crear usuarios con el rol que especifique y ver la lista de estos. Siendo su caso de uso:

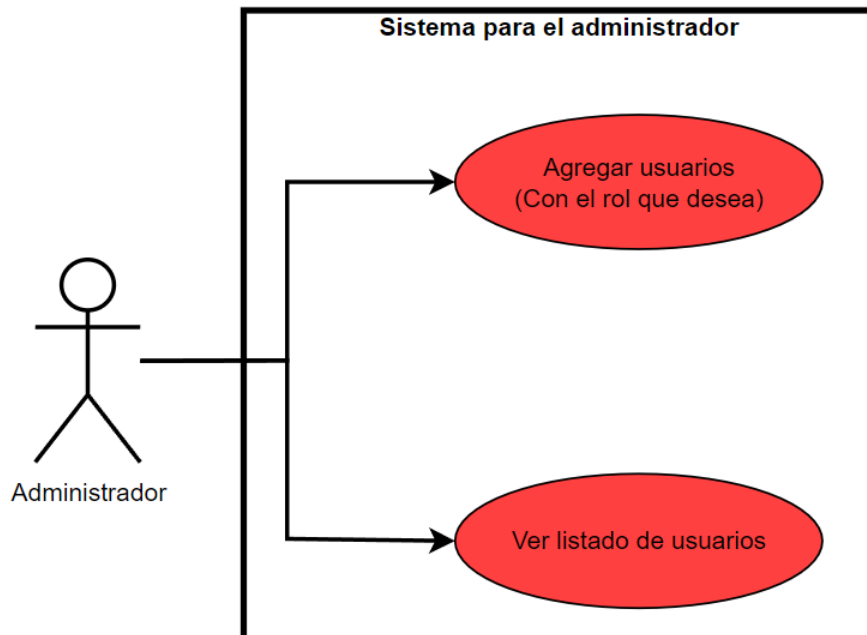


Figura 6: Caso de uso Administrador

Fuente: Elaboración propia

3.2 Diseño de la Arquitectura.

La arquitectura que se ha realizado contiene los elementos de cómo va a trabajar en ambiente de producción en la plataforma de AWS. Se ha realizado una arquitectura para la puesta en marcha, representando la implementación del producto hecho en MVC (véase Figura 7).

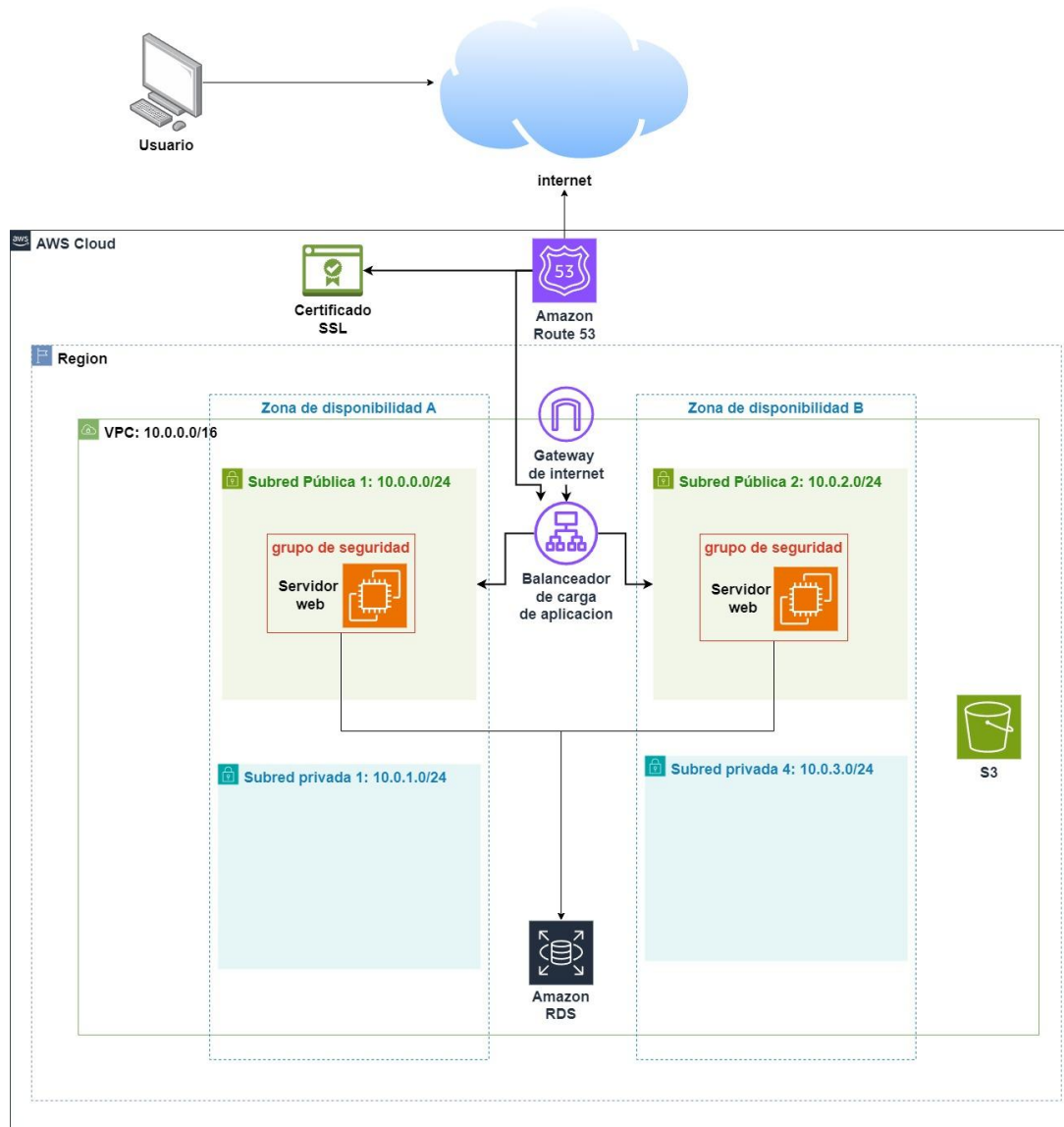


Figura 7: Arquitectura AWS.

Fuente: Elaboración propia.

3.3 Alcance del proyecto

El alcance del proyecto se ha definido para que sea logable y cumpla con los objetivos, respetando el límite de tiempo de 6 meses. No se realizará la visualización o creación de *UI/UX*. Una vez establecidos los requisitos, se determinó qué aspectos son alcanzables y cuáles no, para poder comenzar a desarrollar la plataforma como solución.

Los puntos que se establecieron como objetivos a realizar son:

- Los casos de uso de Persona natural (véase la figura 2).
- Los casos de uso del Operador (véase la figura 4).
- Los casos de uso del Instalador (véase la figura 5).
- Los casos de uso del Administrador (véase la figura 6).
- La implementación de la seguridad de los *endpoints*.
- La implementación del *token*.
- La implementación del producto hacia los servicios en la nube."

3.4 Gestión de Proyecto y versiones

A la hora de gestionar un proyecto, es esencial una administración eficaz de proyectos y versiones en el desarrollo de una plataforma web, particularmente cuando se colabora en equipo y se afrontan tiempos limitados. En este proyecto, se decidió emplear Git como sistema de control de versiones y GitHub como plataforma de colaboración, gracias a sus múltiples beneficios y sencillez de manejo.

Esto permitirá tener un control de versiones distribuido que permite a los desarrolladores trabajar en paralelo, manteniendo un historial completo de cambios en el código fuente. Esto se debe a:

- **Gestión del historial:** Git ofrece un registro exhaustivo de todas las modificaciones efectuadas en el proyecto, lo que posibilita retroceder a versiones previas si se requiere.
- **Trabajo colaborativo:** Facilitando que el equipo colabore en distintas secciones del proyecto al mismo tiempo, evitando disputas.
- **Ramas (*Branch*):** La formación de ramas permite crear nuevas propiedades o rectificar fallos sin perjudicar la rama principal del proyecto. Una vez finalizado y verificado el trabajo en la rama, se puede reintegrar a la rama principal.
- **Repositorio Centralizado:** Gracias a que *github* provee acceso al repositorio a todos los desarrolladores.

3.5 Aspectos técnicos

En este aspecto se establecieron como debería de ser construido la solución, para establecer un orden y trabajar modularmente y separar lo máximo posible las responsabilidades de cada clase.

3.5.1 Modelo vista controlador

Para este tipo de arquitectura se estableció trabajar con estándar que ofrece *Spring*, y es la separación por capas que sigue principios y patrones comunes en la arquitectura de aplicaciones web. Las capas con las que trabajaremos son:

- **Controllers layer:** esta capa se encargará del manejo de las peticiones *HTTP*, y dar repuestas a estas peticiones con el mismo estándar. En *Spring* existen varios tipos de *Controller* pero principalmente se ocuparan:
 - **RestController:** es una anotación de *Spring* que sirve para identificar un controlador y especificar que es del tipo *APIs RestFull*.
 - **AdviceController:** otra anotación de *Spring* para especificar que es un controlador, pero esta vez se especializa en caso de que ocurra algún error y este lo maneje.
- **Services layer:** en esta capa se procesa principalmente la lógica de negocios, esta capa se trabaja por lo general creando una clase abstracta que represente los métodos que debería contener, para luego implementarla. Es necesario que la implementación contenga su anotación correspondiente
- **Repositories layer:** Aquí se maneja la interacción con la base de datos para realizar las operaciones *CRUD*, aquí trabaja el *ORM* de *Java Hibernate*. Que es utilizada con la anotación *Repository* o extendiendo de alguna clase. Para este caso se trabajó con extendiendo la clase *CrudRepository*, con la que tiene establecido métodos *CRUD* listos para usar o personalizar consultas.
- **Models layer:** En esta capa se definen los objetos de la aplicación, además de establecer como se trabajarán los datos. Hay varias maneras de representar modelos y con sus propósitos, por lo cual se utilizarán estas anotaciones:
 - **Entity:** esta anotación sirve para que el *ORM* persista los datos y los mapee a el objeto a una tabla de *SQL*.
 - **DTO:** DTO significa objeto de transferencia de datos, sirve comúnmente para cuando quieres que se muestren o ingresen ciertos datos de algún *entity* o *model* pero excluyendo algunos campos

Estos son las capas más comunes de visualizar y el estándar para realizar una plataforma web. Aun así, *Spring* y algunas dependencias permiten crear más capas, que se utilizaran para la creación de la plataforma que se listara y explicaran:

- **Security layer:** capa en donde se especifican las configuraciones de seguridad, como lo son filtros, codificaciones, llaves, *cors*, etc. Pertenece a dependencia de *Spring Security*.
- **Filters layer:** Que deriva de la capa de seguridad, aquí se implementan los filtros y como trabajarán. Pertenece a dependencia de *Spring Security*.

- ***Resolvers layer***: Capa en donde se encargará de resolver y traducir mensajes de errores para personalizarlos, aquí por lo general se implementa la clase *MessageResolver*.
- ***Validations layer***: Aquí se manejan validaciones personalizadas como anotaciones, se establecen para que campo validan, el mensaje por default, y su método de ejecución. Pertenece a dependencia de *Spring Validators*.

3.5.2 Microservicio

Para transformar lo que se realizó en modelo vista controlador se respetara la separación por capas, pero se separa los microservicios por responsabilidades y por casos de uso. Se agregará una nueva capa para cuando los microservicios necesiten comunicarse con otros microservicios la cual es *Client* de la dependencia *OpenFeign* que permite comunicarse usando *API REST* y devuelve el objeto esperado, y para esto también trabajaremos con *models* para representar el objeto que se espera. Por lo cual se identificaron los siguientes microservicios:

- ***msvc-auth***: Encargado de la autenticación y seguridad de la solución.
- ***msvc-inventory***: Encargado del manejo de inventario.
- ***msvc-invoice***: Encargado del manejo de las facturas.
- ***msvc-scheduler***: Encargado de manejar la agenda.

Además, se crearon archivos *Dockerfile* para la creación de contenedores y poder estos implementar microservicios.

3.5.3 Docker

Los archivos Docker files se crearon con la intención de permitir a quien está implementando estas imágenes poder cambiar algunas variables si es que este lo requiere, además la construcción de la estructura sigue un enfoque multietapa para optimizar el tamaño de la imagen final y la eficiencia de la construcción.

Esta plantilla utiliza la imagen de *Bitnami* que esta optimizado para entornos de trabajos con java y su sistema operativo es *debian*. La plantilla consta de 2 capas la cual, la primera etapa se encarga de la construcción del contenedor en donde copia los archivos necesarios para el correcto funcionamiento del microservicio es instala las dependencias necesarias al igual que crea el compilado. La segunda capa es para el tiempo de ejecución de este microservicio en donde nos establece el directorio de trabajo, crea la carpeta para el almacenamiento de los logs, copia el *jar* que se creó en la etapa anterior a la carpeta de trabajo, luego se establecen las configuraciones del puerto en donde trabajara en donde contiene un puerto por default, aunque se puede cambiar la variable de entorno si es que lo necesita, para terminar con el comando de inicio del contenedor. Además, contiene un puro argumento que es el nombre del microservicio.

3.5.4 Docker Compose

Una vez creados los archivos *Dockerfile* se realizó un archivo *yaml* para la creación de docker compose, con el fin de gestionar el conjunto de contenedores y hacer más fácil el despliegue de estos, en donde se definieron servicios en donde se implementan los microservicios al igual que la base de datos, volúmenes para la persistencia de los datos y redes para la comunicación de los microservicios.

Se configuraron volúmenes para la persistencia de los datos y una red para la comunicación de los servicios llamada *msvc-rad*. Además, consta de 5 servicios, estos son un servicio para la base de datos llamado “mysql8” y los microservicios explicados anteriormente (véase el punto 3.5.2) que dependen del servicio de base de datos para su correcto funcionamiento.

3.6 Base de datos

En lo que compete a la base de datos se desarrolló el diagrama entidad relación (Figura 8) de cómo debería de funcionar a la hora de construir la solución.

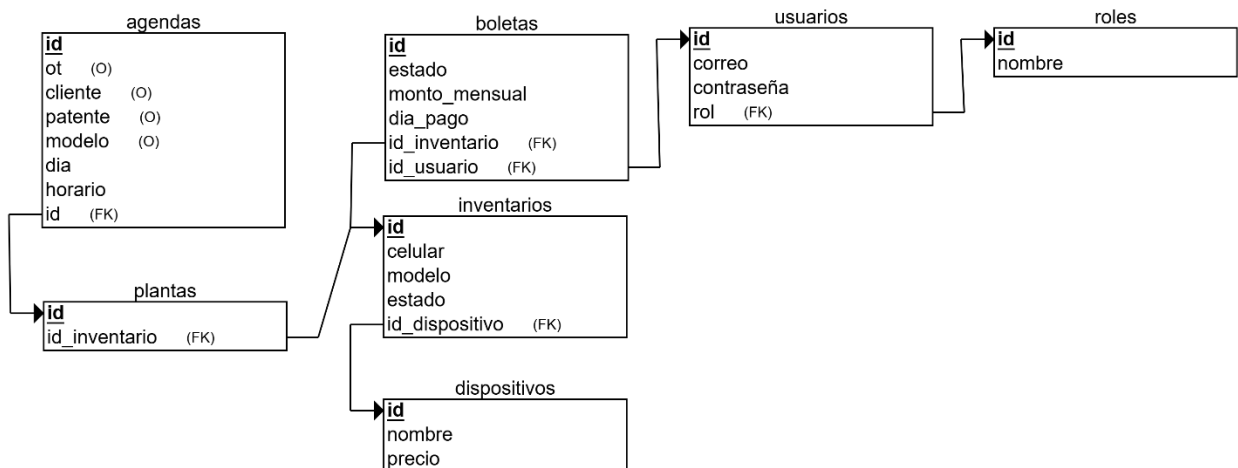


Figura 8: Entidad relación del sistema.

Fuente: Elaboración propia.

Ahora se explicará cada entidad y el tipo de dato de sus columnas.

- **Agendas:** esta entidad servirá para guardar la información sobre cuándo se va a realizar la implementación del dispositivo rad con su orden de trabajo correspondiente. Las columnas son:
 - **Id:** entero, identificador del registro, auto incremental.
 - **Ot:** cadena de texto, identificador de la orden de trabajo.
 - **Cliente:** cadena de texto, nombre del cliente quien va a ir a la planta.
 - **Patente:** cadena de texto, patente del vehículo que va a ir a la planta.
 - **Modelo:** cadena de texto, modelo del vehículo que va a ir a la planta.
 - **Día:** fecha, fecha disponible para la implementación del dispositivo.

- **Horario:** cadena de texto, bloque de hora disponible para la implementación del dispositivo.
- **Roles:** entidad que guarda los roles del sistema y las columnas son:
 - **Id:** entero, identificador del registro, auto incremental.
 - **Nombre:** cadena de texto, nombre del rol.
- **Usuarios:** en donde se guardan la información del usuario necesarias para el ingreso al sistema e información adicional. Sus columnas son:
 - **Id:** entero, identificador del registro, auto incremental.
 - **Correo:** cadena de texto, correo electrónico del usuario.
 - **Contraseña:** cadena de texto, contraseña del usuario.
 - **Rol:** llave foránea de la tabla rol, indica el rol del usuario.
- **Dispositivos:** aquí los dispositivos que existen como productos se registran. Las columnas son:
 - **Id:** entero, identificador del registro, auto incremental.
 - **Nombre:** cadena de texto, nombre del dispositivo.
 - **Precio:** *double*, indica el precio referencial sin IVA del producto.
- **Plantas:** la tabla plantas es solo referencial por ahora, debido a las discusiones que se tienen de cómo se va a trabajar con esta entidad. Por ahora cumple como puente y entender de cual van a ser las relaciones de esta entidad.
- **Inventarios:** en inventario es donde se guarda toda la información de los dispositivos que existe en esa planta, y se registran más detalles sobre estos. Las columnas de la tabla son:
 - **Id:** entero, identificador del registro, auto incremental.
 - **Celular:** cadena de texto, número de teléfono del dispositivo.
 - **Modelo:** cadena de texto, modelo y versión del dispositivo.
 - **Estado:** cadena de texto, se indica en qué estado se encuentra el dispositivo, los estados son, libre, asignado, en reparación, o de baja.
 - **Id del dispositivo:** llave foránea, se indica de que tipo es el dispositivo.
- **Boletas:** una de las entidades más importantes, debido a que aquí está el proceso central del sistema, este sirve para tener la trazabilidad de todo
 - **Id:** entero, identificador del registro, auto incremental.
 - **Estado:** cadena de texto, se indica en qué estado está la boleta, que son, pagado, por pagar, cancelado.
 - **Monto mensual:** *Double*, se indica el monto que el usuario debe pagar en total mensualmente.
 - **Día de pago:** fecha, indica el día en que se debe pagar la mensualidad.
 - **Id de inventario:** llave foránea de la tabla inventario, indica el dispositivo instalado.

- **Id de usuario:** llave foránea de la tabla usuarios, indica la persona a quien se le instalo el dispositivo.

Para pasar esto a microservicios se decidió que cada microservicio manejase su base de datos o esquema con el mismo nombre del microservicio.

3.7 Implementación de la seguridad

Spring boot nos ofrece una dependencia que nos entre una metodología de trabajo para poder implementar la seguridad en nuestro proyecto llama *Spring Security*, en conjunto implementaremos 2 dependencias más al proyecto que son *jsonwebtoken* de *jjwt-api* que nos entrega directamente *JWT*, y la otra dependencia es *passay* para las validaciones de contraseña (a la hora de crear una contraseña) que verifica si la contraseña cumple con requisitos mínimos para la creación de una contraseña tales como, contiene mayúsculas, minúsculas, números, no tiene números secuenciales, etc.

Se crearon 4 clases principales las cuales serán enumeradas y explicadas brevemente a continuación:

- **Configuraciones del token JWT:** Clase encargada de guardar constantes importantes para la implementación de los *tokens*. Aquí se dejan configuraciones de las llaves, prefijos (del *token*), cabeceras, tipo de contenido, etc.
- **Configuraciones de seguridad de Spring:** este se encarga las configuraciones generales de seguridad. Aquí se configuran los filtros y protección de los *endpoint*, los *CORS*, *CSRF*, Manejos de la sesión (políticas), etc.
- **Filtro de autenticación JWT:** aquí se trabaja la validación del ingreso al sistema (*login*), el caso exitoso (retornando el *token* firmado) y en el caso de error que retorna un mensaje de error y 401 *http*.
- **Filtro de validacion JWT:** aquí se programa que la solicitud entrante hacia un *endpoint* protegido venga con el *token*, y verifica si el *token* es válido o no (Revisando la integridad, la caducidad, etc.).

También se implementaron encriptaciones a los datos sensible como los son las contraseñas de los usuarios.

3.8 Endpoint y controladores.

En este apartado se detallará la implementación de los *endpoint*, los nombres de los controladores, y como se trabaja con ellos, explicando que se espera a la hora de ingresar o mostrar datos.

3.8.1 Controlador del usuario

Este se encargará de mostrar el detalle del usuario, la lista de usuarios, registrar los usuarios, y modificar los usuarios. Se decidió no implementar una

eliminación de usuarios para tener un registro de estos, en cambio se tomó la decisión de habilitar o deshabilitar los usuarios.

En lo que compete a la seguridad de este *endpoint* se protege con autenticación y solo los administradores pueden acceder a este recurso. La ruta (*endpoint*) raíz de este controlador es */api/users*. A continuación, se especificará cada ruta y una breve explicación de lo que debe de entrar o salir, la respuesta *HTTP* y los permisos.

- **Método *GET /api/users*:** devuelve la lista de usuarios con un 200 *ok*. Requiere autorización y autenticación.
- **Método *GET /api/users/{identificador}*:** devuelve el detalle del usuario con un 200 *ok*. Requiere autorización y autenticación.
- **Método *POST /api/users/register*:** espera un ingreso en formato *JSON* del modelo usuario y devuelve el detalle del usuario ingresado con un 201 *created*. No requiere autorización ni autenticación.
- **Método *POST /api/users*:** espera un ingreso en formato *JSON* del modelo usuario y devuelve el detalle del usuario ingresado con un 201 *created*. Requiere autorización y autenticación.
- **Método *PUT /api/users/{identificador}*:** espera un ingreso en formato *JSON* del campo habilitado y devuelve el detalle del usuario ingresado con un 201 *created*. Requiere autorización y autenticación.

3.8.2 Controlador del inventario

Aquí se trabajará los *CRUD* del inventario donde se mostrará, el detalle del inventario, la lista de los inventarios, registro del inventario, la modificación del inventario, y la eliminación de un registro.

En lo que compete a la seguridad de este *endpoint* se protege con autenticación y autorización, y solo los Administradores o Instaladores pueden acceder a estos recursos. La ruta raíz de este controlador es */api/inventory*. A continuación, se especificará cada ruta y una breve explicación de lo que debe de entrar y/o salir, la respuesta *HTTP* y los permisos.

- **Método *GET /api/inventory*:** devuelve la lista de los inventarios con un 200 *ok*. Requiere autorización y autenticación.
- **Método *GET /api/inventory/{identificador}*:** devuelve el detalle del inventario con un 200 *ok*. Requiere autorización y autenticación.

- **Método *POST /api/inventory***: espera un ingreso en formato *JSON* del modelo inventario y devuelve el detalle del inventario ingresado con un 201 *created*. Requiere autorización y autenticación.
- **Método *PUT /api/inventory/{identificador}***: espera un ingreso en formato *JSON* del modelo inventario y devuelve el detalle del inventario ingresado con un 201 *created*. Requiere autorización y autenticación.
- **Método *DELETE /api/inventory/{identificador}***: no espera nada en el cuerpo de la petición devuelve un 204 *no content*.

3.8.3 Controlador de dispositivo

Aquí se trabajará los *CRUD* del dispositivo donde se mostrará, el detalle del dispositivo, la lista de los dispositivos, registro del dispositivo, la modificación del dispositivo, y la eliminación de un registro.

En lo que compete a la seguridad de este *endpoint* se protege con autenticación y autorización, y solo los Administradores pueden acceder a estos recursos. La ruta raíz de este controlador es */api/device*. A continuación, se especificará cada ruta y una breve explicación de lo que debe de entrar y/o salir, la respuesta *HTTP* y los permisos.

- **Método *GET /api/device***: devuelve la lista de los dispositivos con un 200 *ok*. Requiere autorización y autenticación.
- **Método *GET /api/device/{identificador}***: devuelve el detalle del dispositivo con un 200 *ok*. Requiere autorización y autenticación.
- **Método *POST /api/device***: espera un ingreso en formato *JSON* del modelo dispositivo y devuelve el detalle del dispositivo ingresado con un 201 *created*. Requiere autorización y autenticación.
- **Método *PUT /api/device/{identificador}***: espera un ingreso en formato *JSON* del modelo inventario usuario y devuelve el detalle del inventario ingresado con un 201 *created*. Requiere autorización y autenticación.
- **Método *DELETE /api/device/{identificador}***: no espera nada en el cuerpo de la petición devuelve un 204 *no content*.

3.8.4 Controlador de agenda

Aquí se trabajará los *CRUD* de la agenda donde se mostrará, el detalle de la agenda, la lista de la agenda, registro de una nueva agenda, la modificación de la agenda, y la eliminación de un registro.

En lo que compete a la seguridad de este *endpoint* se protege con autenticación y autorización, y solo los Administradores y operadores pueden acceder a estos recursos. La ruta raíz de este controlador es */api/scheduler*. A continuación, se especificará cada ruta y una breve explicación de lo que debe de entrar y/o salir, la respuesta *HTTP* y los permisos.

- **Método GET */api/scheduler*:** devuelve la lista de la agenda con un 200 ok. Requiere autorización y autenticación.
- **Método GET */api/scheduler/{identificador}*:** devuelve el detalle de la agenda con un 200 ok. Requiere autorización y autenticación.
- **Método POST */api/scheduler*:** espera un ingreso en formato *JSON* del modelo agenda y devuelve el detalle de la agenda ingresada con un 201 *created*. Requiere autorización y autenticación.
- **Método PUT */api/scheduler/{identificador}*:** espera un ingreso en formato *JSON* del modelo agenda y devuelve el detalle de la agenda ingresada con un 201 *created*. Requiere autorización y autenticación.
- **Método DELETE */api/scheduler/{identificador}*:** no espera nada en el cuerpo de la petición devuelve un 204 *no content*.

3.8.5 Controlador de boleta

Aquí se trabajará los *CRUD* del dispositivo donde se mostrará, el detalle de la boleta, la lista de las boletas, registro de una boleta, la modificación de la boleta, y la eliminación de un registro.

En lo que compete a la seguridad de este *endpoint* se protege con autenticación y autorización. La ruta raíz de este controlador es */api/invoice*. A continuación, se especificará cada ruta y una breve explicación de lo que debe de entrar y/o salir, la respuesta *HTTP* y los permisos.

- **Método GET */api/invoice*:** devuelve la lista de las boletas con un 200 ok. Requiere autorización y autenticación.
- **Método GET */api/invoice/{identificador}*:** devuelve el detalle de la boleta con un 200 ok. Requiere autorización y autenticación.

- **Método *POST /api/invoice*:** espera un ingreso en formato *JSON* del modelo boleto y devuelve el detalle de la boleto ingresado con un *201 created*. Requiere autorización y autenticación.
- **Método *PUT /api/invoice/{identificador}*:** espera un ingreso en formato *JSON* del modelo boleto y devuelve el detalle de la boleto ingresado con un *201 created*. Requiere autorización y autenticación.
- **Método *DELETE /api/invoice/{identificador}*:** No espera nada en el cuerpo de la petición devuelve un *204 no content*.

3.9 Implementación y puesta en marcha

Veremos que trabajo sea realizado para la implementación de la solución hacia la plataforma de *Amazon web services (AWS)*. En total se realizó la implementación respetando la arquitectura que se propuso anteriormente.

3.9.1 implementación de la solución modelo vista controlador (MVC) a la plataforma *cloud*

Se han realizado la implementación de la arquitectura en la cual se utilizaron servicios tales como, la red virtual privada en la nube (*VPC*), en donde se creó.

- **Subredes:** se crearon en total 4 subredes 2 privadas y 2 públicas, en donde se les asignaron las *CIDR* de *IPv4* correspondiente a la arquitectura.
- **Tablas de enrutamiento:** se establecieron tablas de enrutamientos para las conexiones entre subredes.
- **Gateway de internet:** para la conexión de las subredes publicas hacia el internet.

También se utilizó *Amazon EC2*, donde se crearon las dos instancias que aparece en la arquitectura que se propuso, para luego conectar estas instancias con el servicio *Elastic Load Balacer*, los trabajos que se realizaron son.

- **Grupo de seguridad:** se crearon 3 grupos de seguridad, el primero es para la base de datos dejando libre el puerto 3306 *TCP MySQL* cualquier *IP*, el segundo grupo de seguridad es para *load balancer* y se habilita el puerto 80 *TCP HTTP* cualquier *IP* y 443 *TCP HTTPS* cualquier *ip*, por último, el grupo de seguridad de las instancias dejando los puertos 22 *TCP SSH* cualquier *ip*, 80 *TCP HTTP* conectada al grupo de seguridad del *load balancer*, 8080 para la *API backend* cualquier *ip*.

- **Instancia EC2 A:** esta instancia se creó usando la imagen de Amazon Linux, se le asignó la zona de disponibilidad us-east-1-a, se le creó su archivo *pem* para el acceso a esta instancia, se instaló java, y se mandó el compilado *Jar* de la aplicación a través de *SCP* para una transferencia de archivos seguro
- **Instancia EC2 B:** se realizó exactamente el mismo trabajo y con las mismas características que en la instancia A con la única diferencia de que la zona de disponibilidad es us-east-1-b.
- **Target group:** este tiene la finalidad de enrutar el tráfico y realizar *health checks* a los recursos *backend*, por lo cual se le asignó que los *targets* son de tipo instancias y apunta a las dos instancias de EC2.
- **ELB:** se comenzó creando un *Application Load Balancer* (ALB) ya que es ideal para manejar tráfico *HTTP* y *HTTPS*, luego se le asignó el grupo de seguridad, después el *target group* creado previamente, se le asigna la zona de disponibilidad us-east-1-a us-east-1-b y se crea.

Para la base de datos se utilizó *RDS*, servicio que nos entrega el motor de base de datos con la ventaja de que viene administrado por *AWS*. Aquí se realizó la creación de la base de datos *MySQL*.

Y por último se utilizó *Route 53* para gestionar el *DNS* y *certificate manager* para el certificado *SSL* así lograr que se pudiera visualizar la página web a través de internet accediendo a nuestro dominio. Se realizó la compra y el registro del dominio *remoteautodisengage.cl* por *nic.cl* (Nic, 2024), luego lo que se realizó fue.

- **Zona hospedada:** se creó una zona hospedada de *remoteautodisengage.cl* en la cual tiene 4 registros que son el *remoteautodisengage.cl* de tipo *NS* en donde se tomó los valores y los registramos a *NIC*, "*remoteautodisengage.cl*" de tipo *SOA*, el del certificado *SSL* de tipo *CNAME*, y por último el sub dominio *www* de tipo *A* en donde redirige hacia el *load balancer* mostrando la página web.
- **Solicitud del certificado:** Se envió una solicitud del certificado registrando el dominio de la web del proyecto "*remoteautodisengage.cl*" y el subdominio de la web "*www.remoteautodisengage.cl*" para el *DNS*, utilizando un algoritmo *RSA* de 2048 bits para la llave.

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

4.1 Pruebas de seguridad.

En este apartado se realizarán pruebas de seguridad las cuales demostraran que la autenticación y autorización funcionan en el sistema. Para esto mostraremos ejemplos utilizando la herramienta *Postman* usando scripts y revisando las respuestas del servidor.

4.1.1 Caso 1: Usuario no autenticado.

Cuando un usuario no autenticado desea ingresar algún recurso protegido, se debería rechazar cualquier tipo de ingreso ya sea mostrar información, modificar o eliminar en esa parte del sistema. Se espera que el sistema responda con un estado de *HTTP* 403 indicando la denegación de la petición. A continuación, mostraremos el caso utilizando la herramienta *Postman* (figura 9).

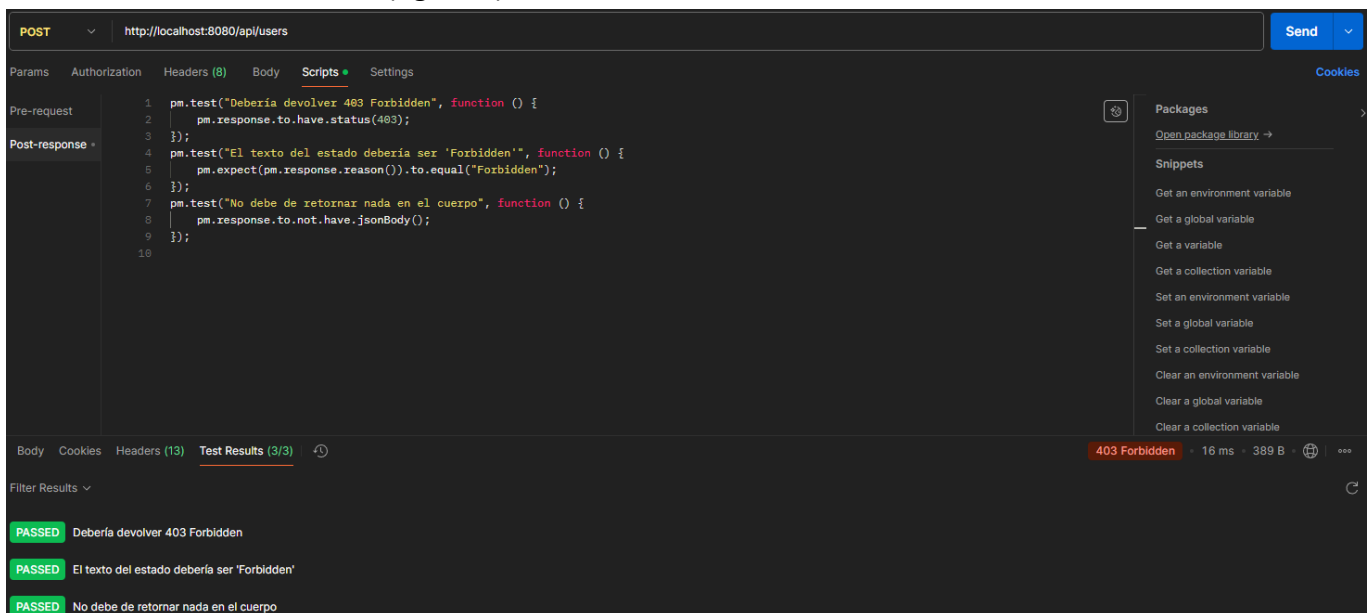


Figura 9: Test de seguridad usuario no autenticado.
Fuente: Elaboración propia.

4.1.2 Caso 2: Usuario autenticado con el *token* expirado

Cuando el usuario ingresa con un *token* expirado a la hora de usar *bearer token* el sistema debería de denegar la petición y responder con un *HTTP 401* e indicar el error y un mensaje. A continuación, mostraremos el caso utilizando la herramienta *Postman* (Véase la Figura 10 y 11).

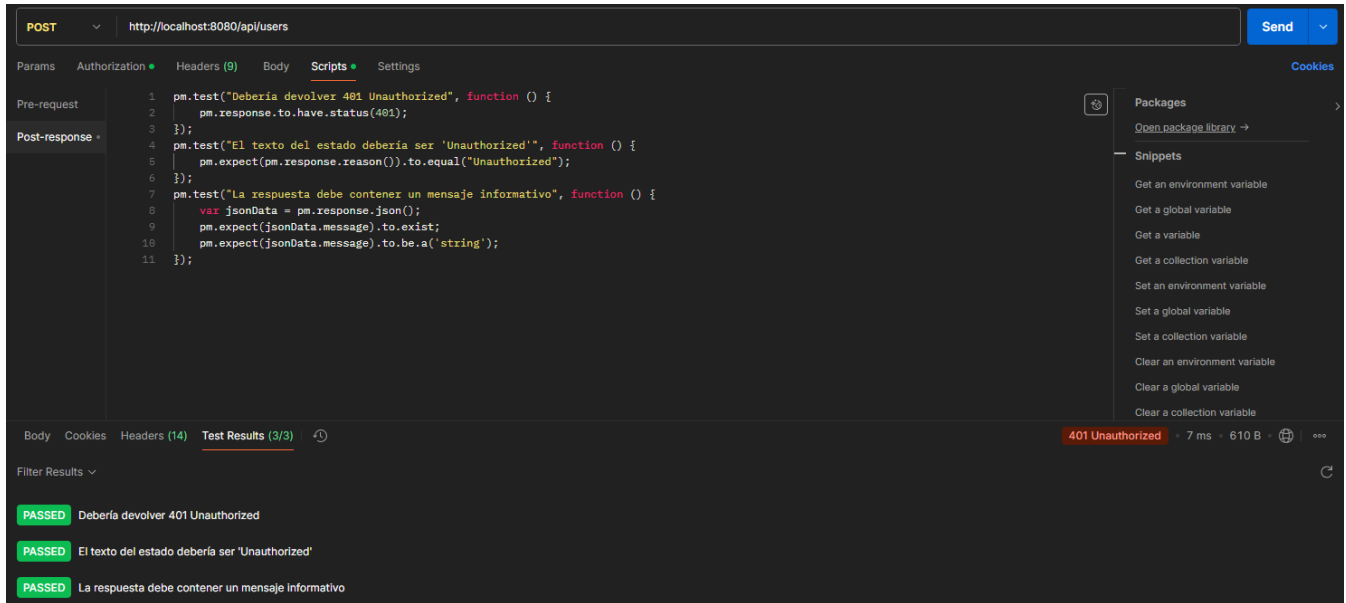


Figura 10: Test de seguridad usuario con token expirado.
Fuente: Elaboración propia.

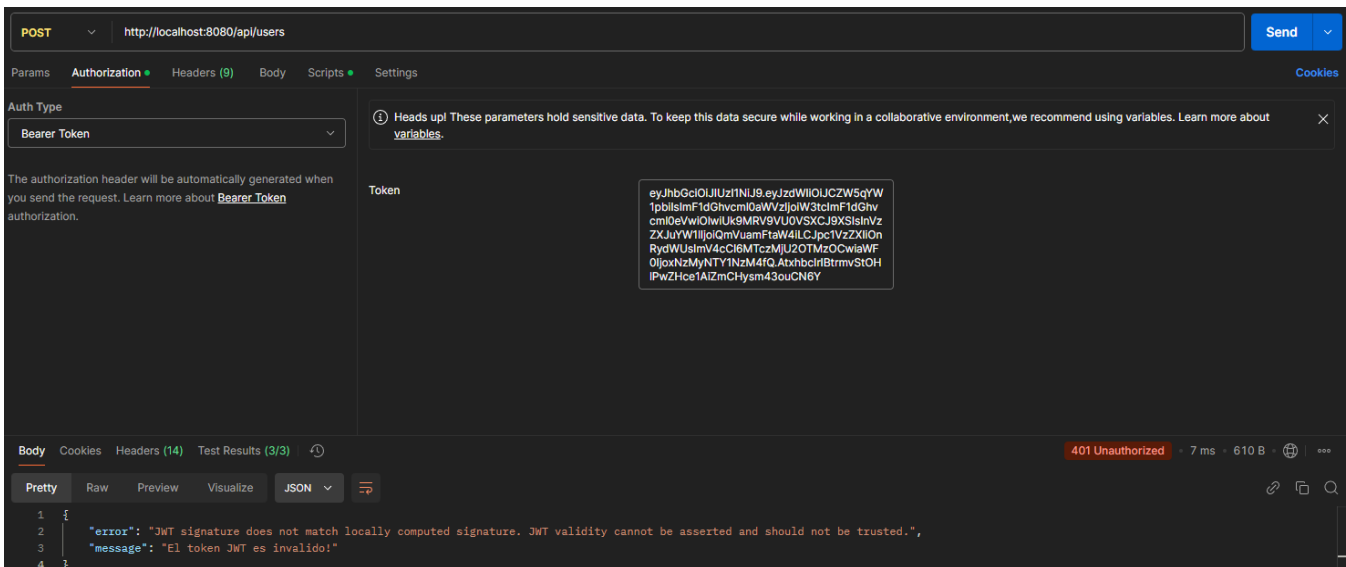


Figura 11: Mensaje de seguridad usuario con token expirado.
Fuente: Elaboración propia.

Si un usuario existente en el sistema y a ingresado puede ocurrir que intencionalmente o por error trate de acceder a un recurso que no tiene autorización, en estos casos el sistema debe denegar la petición y responder con un *HTTP* 403 y el cuerpo debe de estar vacío. A continuación, mostraremos el caso (figura 13 y 14), accediendo con las credenciales de Matías que es un usuario existente del sistema y tiene el rol de instalador, utilizando la herramienta *Postman* accederemos a la ruta *“api/scheduler”* que está autorizado solo para Administradores y Operadores.



4.1.4 Caso 4: Usuario modifica el *token*

Si el usuario modifica el *token* que se le entrega al hacer *login*, el sistema debe de denegar la petición y responder con un *HTTP 401* e indicar el error y un mensaje. A continuación, mostraremos el caso (figura 14, 15, 16 y 17) utilizando la herramienta *Postman* con el *token* que se mostró anteriormente (véase la figura 12 o 14) modificado y accediendo a la misma ruta.

Encoded
PASTE A TOKEN HERE

```

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJNYXRpYX
MiLCJhdXRob3JpdG1lcYI6Ilt7XCJhdXRob3Jpd
HlcIjpcIlJPTEVfSU5TVEFMTEVSXCJ9XSIsInVz
ZXJ1eW11IjoiTWF0aWFzIiwiaXNjbN0YWxsZSI
iOnRydWUsImV4cCI6MTczNjczMzI2OSwiaWF0Ij
oxNzM2NzI5NjY5fQ.q1iJLotayVsNCptzQ_18fC
5bNvgJ2o3Ay0r3Gxz120k

```

Decoded
EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

PAYLOAD: DATA

```

{
  "sub": "Matias",

  [
    {
      "username": "Matias",
      "isInstaller": true,
    }
  ]
}

```

VERIFY SIGNATURE

Figura 14: Token del usuario Matías sin modificar.

Figura: Elaboración propia.

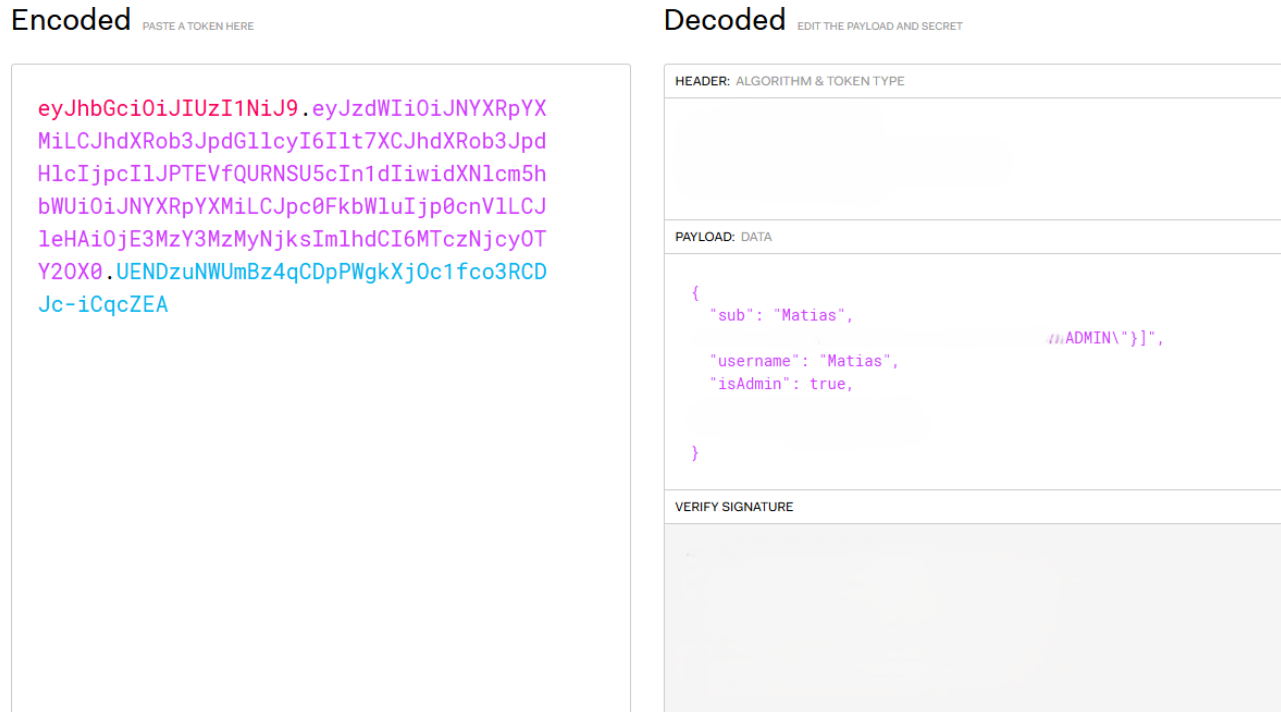


Figura 16: Token del usuario Matías modificado.
Figura: Elaboración propia.

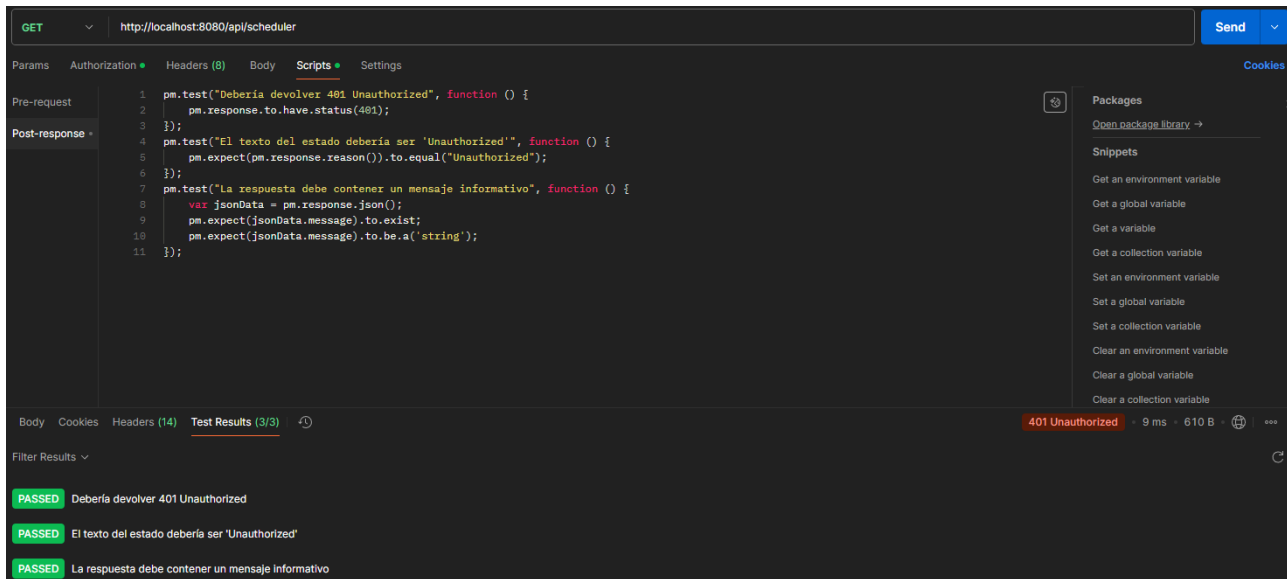
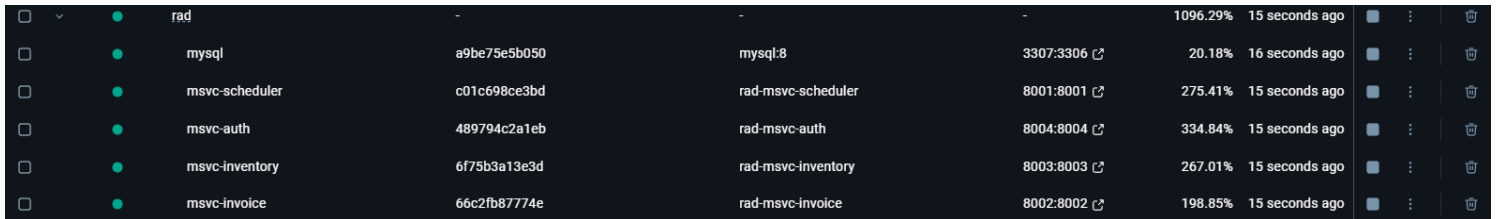


Figura 15: Test de seguridad usuario Matías con *token* modificado
Fuente: Elaboración propia

4.2 Pruebas de comunicación (Microservicios)

Para hacer pruebas de comunicación de estos microservicios, se levantará el *Docker-compose* que se ha configurado anteriormente. Luego, se verificará el funcionamiento de estos microservicios en Docker (véase la figura 18). Después, se probará el microservicio “*msvc-invoice*” que maneja las boletas (no contables). Este requiere información del usuario y del inventario, ambos datos se encuentran en dos contenedores distintos. Por lo tanto, se mostrarán los datos que tienen esos microservicios (figura 19 y 20) y se verificará la consistencia de los datos a la hora de la comunicación entre estos (figura 21).



☐	●	rad	-	-	-	1096.29%	15 seconds ago			
☐	●	mysql	a9be75e5b050	mysql:8	3307:3306	20.18%	16 seconds ago			
☐	●	msvc-scheduler	c01c698ce3bd	rad-msvc-scheduler	8001:8001	275.41%	15 seconds ago			
☐	●	msvc-auth	489794c2a1eb	rad-msvc-auth	8004:8004	334.84%	15 seconds ago			
☐	●	msvc-inventory	6f75b3a13e3d	rad-msvc-inventory	8003:8003	267.01%	15 seconds ago			
☐	●	msvc-invoice	66c2fb87774e	rad-msvc-invoice	8002:8002	198.85%	15 seconds ago			

Figura 19: Contenedores funcionando (*Docker-Compose*)

Fuente: Elaboración propia

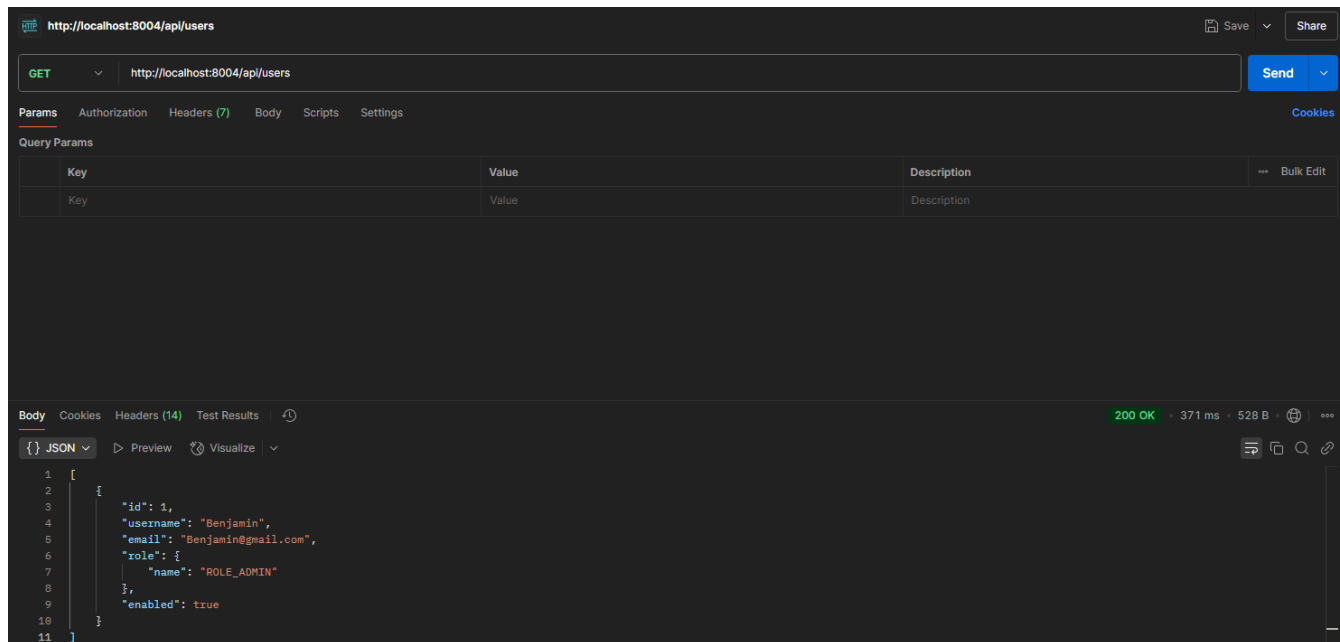


Figura 18: Datos de los usuarios (*msvc-auth*)

Fuente: Elaboración propia

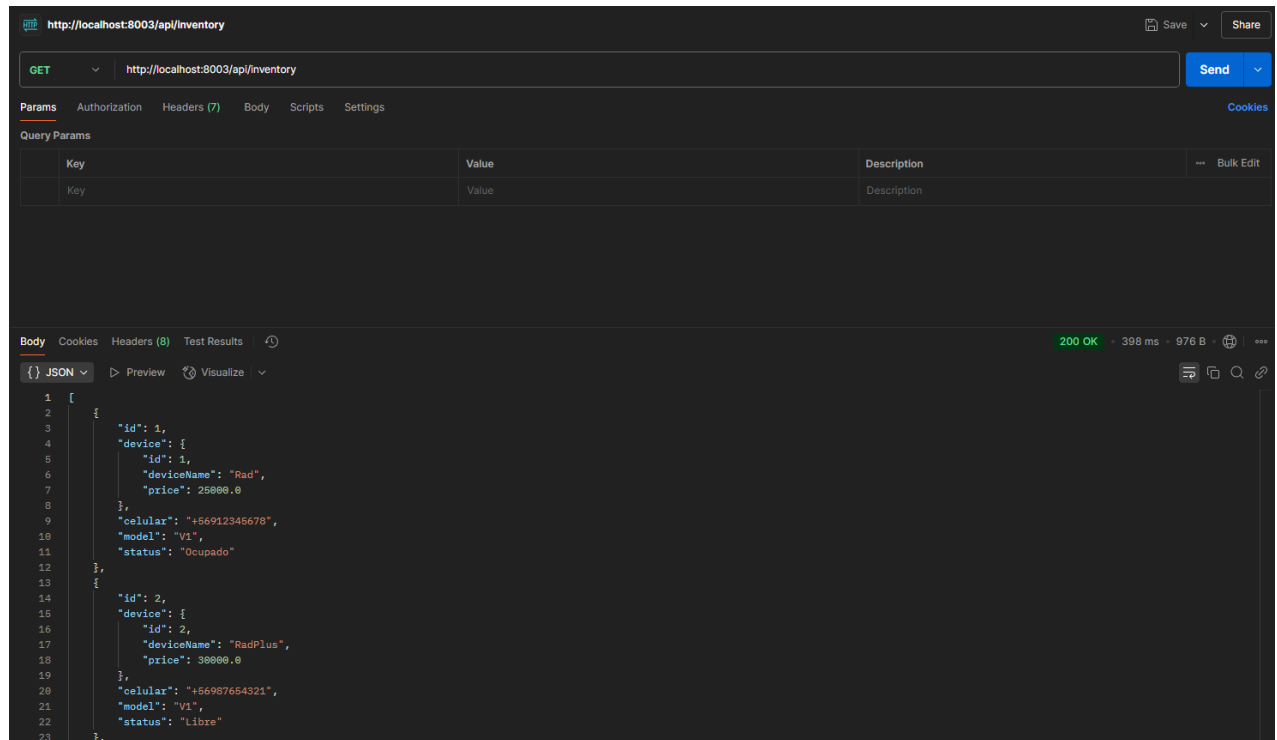


Figura 20: Datos de los inventarios (*msvc-inventory*)
Fuente: Elaboración propia

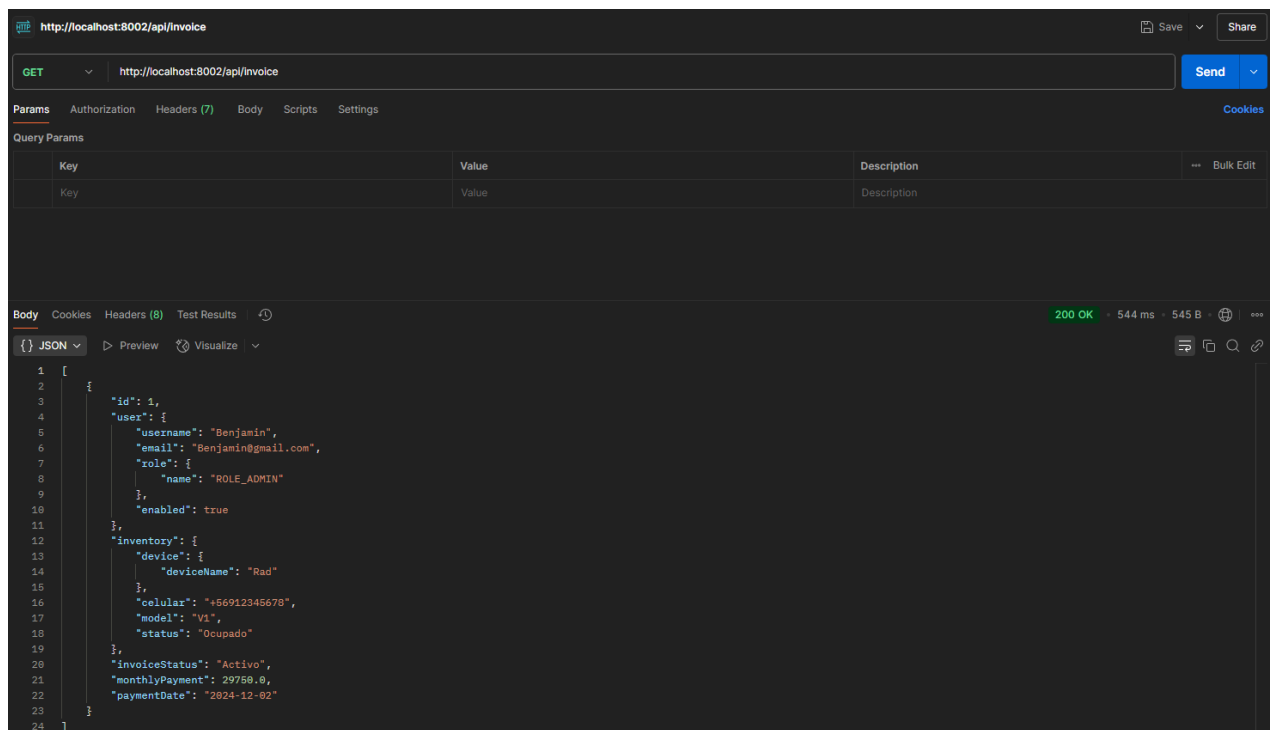
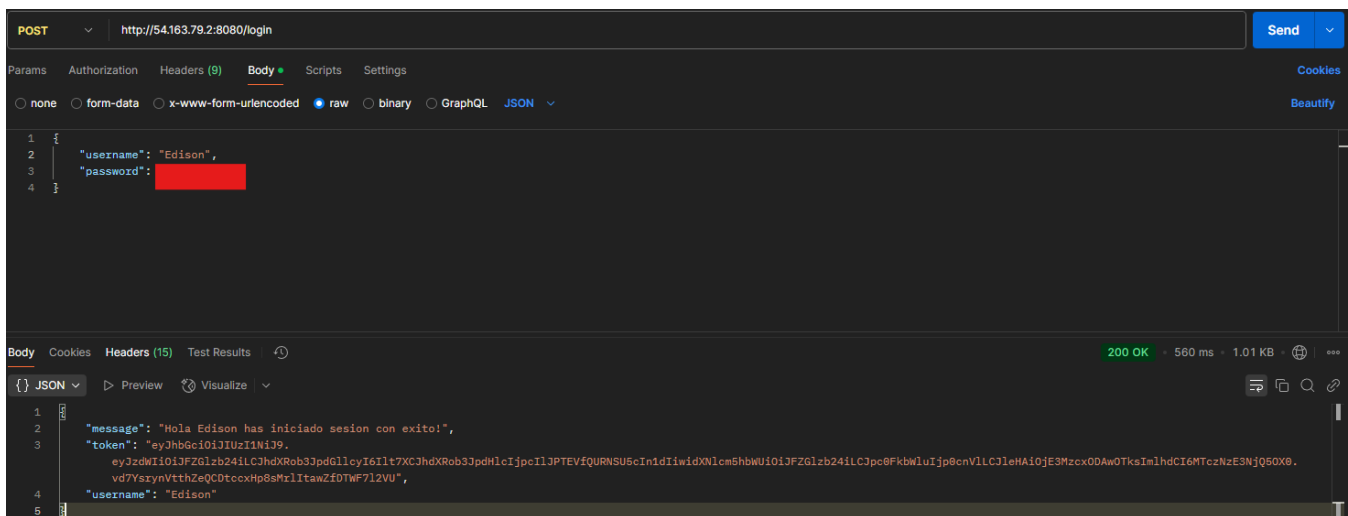


Figura 21: Pruebas de comunicación (*msvc-invoice*)
Fuente: Elaboración propia

A la hora de probar la implementación del *Backend* en la plataforma *cloud* se revisará la correcta implementación de la aplicación en el servicio EC2, al igual que se harán pruebas de comunicación del *API* hacia el exterior para verificar el funcionamiento.

encia de i-0a751f1944b08a027 (backend-rd) Información	Dirección IPv4 pública 54.163.79.2 dirección abierta 🔗 Estado de la instancia En ejecución Nombre DNS de IP privada (solo IPv4) ip-10-0-4-117.ec2.internal Tipo de instancia t2.micro ID de VPC vpc-0e0a2c67f30d5faec (rad-vpc) 🔗 ID de subred subnet-0cafce62fd86502f7 (rad-subnet-public1-us-east-1a) 🔗 ARN de instancia arn:aws:ec2:us-east-1:442426897355:instance/i-0a751f1944b08a027	Conectar Estado de la instancia ▼ Acción Direcciones IPv4 privadas 10.0.4.117 DNS de IPv4 pública ec2-54-163-79-2.compute-1.amazonaws.com dirección abierta 🔗 Direcciones IP elásticas – Hallazgo de AWS Compute Optimizer 🔗 Suscribirse a AWS Compute Optimizer para recibir recomendaciones. Más información 🔗 Nombre del grupo de Auto Scaling – Managed falso
--	--	---

Fuente: Elaboración propia.



Fuente: Elaboración propia.

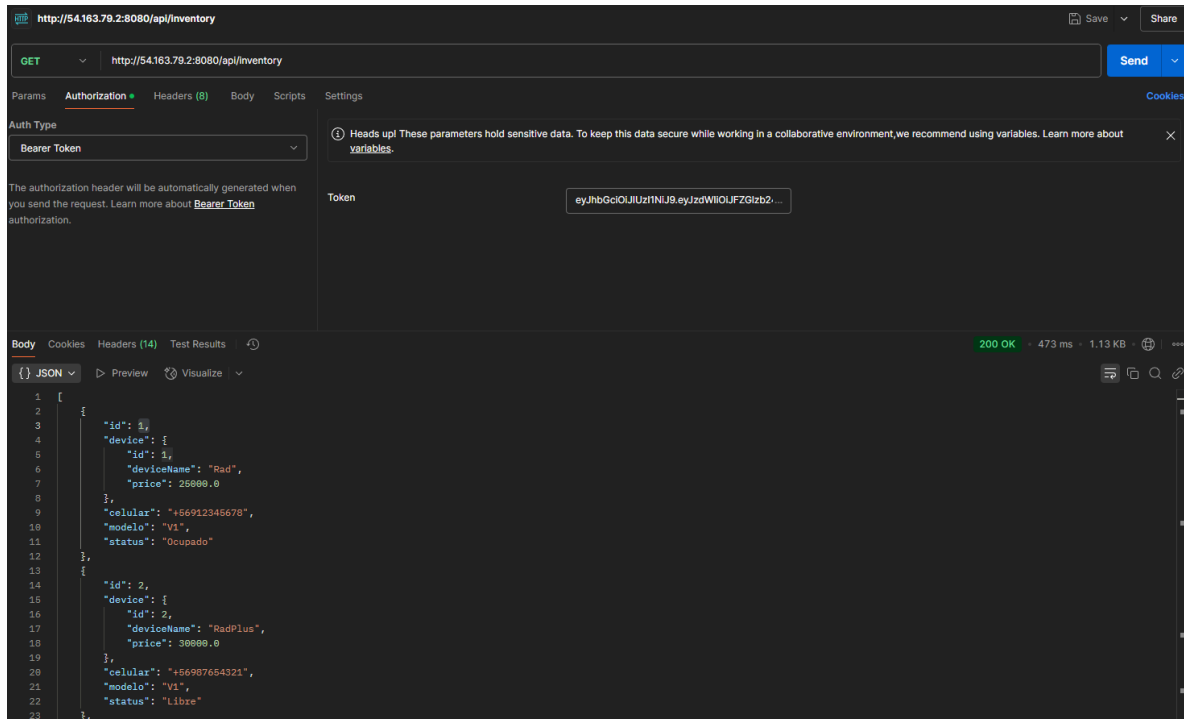


Figura 24: Prueba al *endpoint inventory*
Fuente: Elaboración propia.

CAPÍTULO 5: CONCLUSIONES

5.1 objetivos realizados

Los objetivos iniciales que se han realizado en su totalidad son:

- **Los casos de uso de Persona natural** (véase la figura 2): Se han implementado todos los casos de uso relacionados con los usuarios naturales, asegurando que puedan interactuar con la plataforma de manera efectiva.
- **Los casos de uso del Instalador** (véase la figura 5): Se han desarrollado los casos de uso específicos para los instaladores, permitiendo una gestión eficiente de las instalaciones.
- **Los casos de uso del Administrador** (véase la figura 6): Se han completado los casos de uso para los administradores, facilitando la administración y supervisión de la plataforma.
- **La implementación de la seguridad en los endpoints**: Se han aplicado medidas de seguridad robustas en todos los puntos de acceso de la plataforma para proteger los datos y las operaciones.
- **La implementación del token**: Se ha integrado un sistema de *tokens* para la autenticación y autorización segura de los usuarios.
- **La implementación del producto hacia los servicios Clouds**: Se ha desplegado el producto en servicios en la nube, asegurando su disponibilidad y escalabilidad.

Gracias a estos logros, el proyecto ha avanzado significativamente en la construcción de la plataforma web como solución. Esto ha permitido desacoplar responsabilidades y centrarse únicamente en la lógica de negocios y su correcta implementación. Además, se han aplicado correctamente los estándares, metodologías, patrones y marcos de trabajo para lograr que el proyecto sea lo más escalable, seguro y modular posible. Esto facilita la posibilidad de aumentar los requerimientos o realizar cualquier tipo de aporte al proyecto en el futuro sin incurrir en altos costos.

5.2 Trabajo a futuro

La implementación de los siguientes objetivos quedó como trabajo a futuro:

- **Los requerimientos del usuario estadista** (véase la figura 3): Debido a la necesidad de datos específicos y la falta de recursos para abordar estos casos de uso, su implementación se ha pospuesto.
- **El caso de uso ver plantas de instalación operativa** (véase la figura 4): La implementación de este caso de uso quedó en discusión debido a la complejidad de trabajar con los datos necesarios.
- **Migración de la orquestación de contenedores hacia Kubernetes:** Para la correcta implementación de microservicios y la gestión de configuraciones centralizadas, *gateways*, etc., se planea migrar la orquestación de contenedores a *Kubernetes*.
- **Elastic Container Balancer y Elastic Kubernetes Service:** Se planea utilizar estos servicios para la implementación de microservicios en la plataforma *cloud*, respetando lo creado y lo que se creará con Docker y Kubernetes.

Por lo tanto, el proyecto se considera concluido considerando los objetivos, y se toma en cuenta los avances a futuro que se han planificado para su pronta implementación.

REFERENCIAS BIBLIOGRÁFICAS

- Agarwal, R. (5 de Junio de 2024). *strapi*. Obtenido de Relational Databases: PostgreSQL Vs. MariaDB Vs. MySQL Vs. SQLite: <https://strapi.io/blog/relational-databases-postgresql-vs-mariadb-vs-mysql-vs-sqlite>
- Andrus, B. (2 de Abril de 2024). *dreamhost*. Obtenido de ¿Qué es Docker? Desglosando Esta Popular Plataforma de Aplicaciones: <https://www.dreamhost.com/blog/es/que-es-docker/#:~:text=En%20su%20n%C3%BAcleo%2C%20Docker%20es,utilizando%20la%20tecnolog%C3%ADa%20de%20contenerizaci%C3%B3n>.
- Aviles, R. A. (3 de Octubre de 2022). *onesaitplatform*. Obtenido de guia de uso de spring security: <https://blog.onesaitplatform.com/2022/10/03/guia-de-uso-de-spring-security/>
- Burak, A. (2024). *kanbanzone*. Obtenido de Agile Frameworks Compared: Scrum vs. Kanban vs. Lean vs. XP: <https://kanbanzone.com/2024/agile-frameworks-compared/>
- Cooperativa.cl. (30 de Enero de 2024). *cooperativa*. Obtenido de Los modelos de autos más robados durante el 2023 en Chile: <https://www.cooperativa.cl/noticias/pais/policial/robo-de-vehiculos/los-modelos-de-autos-mas-robados-durante-el-2023-en-chile/2024-01-30/204459.html>
- db-engines*. (2024). Obtenido de System Properties Comparison MariaDB vs. Microsoft SQL Server vs. PostgreSQL: <https://db-engines.com/en/system/MariaDB%3BMicrosoft%20SQL%20Server%3BPostgreSQL>
- Delegación Presidencial Provincial de Arauco*. (6 de Mayo de 2024). Obtenido de Vehículos robados que son recuperados sigue en alza en la provincia de Arauco : <https://dpparauco.dpp.gob.cl/2024/05/06/vehiculos-robados-que-son-recuperados-sigue-en-alza-en-la-provincia-de-arauco/#:~:text=Si%20bien%2C%20hasta%20el%202022,m%C3%B3viles%20con%20encargo%20de%20robo>.
- Eby, K. (15 de Febrero de 2017). *smartsheet*. Obtenido de What's the Difference? Agile vs Scrum vs Waterfall vs Kanban: <https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban>
- Framework Training*. (2024). Obtenido de Spring Boot vs Node.js vs .NET Core vs Django.: <https://frameworktraining.co.uk/news-insights/spring-boot-vs-node-js-versus-dot-net-core-vs-django/>
- García, M. (5 de octubre de 2017). *codingornot*. Obtenido de MVC (Modelo-Vista-Controlador): ¿qué es y para qué sirve?: <https://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve>

- Gimenez, M. (20 de julio de 2020). *hiberus*. Obtenido de Amazon Web Services (AWS): ¿qué es y qué ofrece?: <https://www.hiberus.com/crecemos-contigo/amazon-web-services-aws-que-es-y-que-ofrece/>
- IBM. (2024). Obtenido de ¿Qué es Java Spring Boot?: <https://www.ibm.com/es-es/topics/java-spring-boot>
- Imagina, E. d. (04 de Noviembre de 2024). <https://imaginaformacion.com/tutoriales/que-es-docker-compose>. Obtenido de Qué es Docker Compose y Cómo Usarlo | Tutorial Completo: <https://imaginaformacion.com/tutoriales/que-es-docker-compose>
- Magaña, L. M. (17 de Enero de 2020). *openwebinars*. Obtenido de Qué es Json Web Token y cómo funciona: <https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/>
- Martínez, G. A. (1 de Septiembre de 2021). *bravedeveloper*. Obtenido de Qué es REST, RESTful, API RESTful y JSON: <https://bravedeveloper.com/2021/09/01/que-es-rest-restful-api-restful-y-json/>
- Martins, J. (15 de Febrero de 2024). *asana*. Obtenido de Scrum: conceptos clave y cómo se aplica en la gestión de proyectos: <https://asana.com/es/resources/what-is-scrum>
- Muradas, Y. (3 de Junio de 2019). *openwebinars*. Obtenido de Qué es Postman y primeros pasos: <https://openwebinars.net/blog/que-es-postman/>
- Next.js. (2024). Obtenido de Next.js Documentation: <https://nextjs.org/docs>
- Nic. (2024). Obtenido de <https://nic.cl/>
- Ortuya, N. (25 de Julio de 2024). *autofact*. Obtenido de ¿Donde se roba mas autos en Chile en 2024?: <https://www.autofact.cl/blog/mi-auto/seguridad/donde-robo-autos#:~:text=El%20informe%20Carabineros%20en%20Cifras,Carabineros%20en%20Cifras%22%20de%202023.>
- Portal CEAD. (2024). Obtenido de ESTADÍSTICAS DELICTUALES: <https://cead.spd.gov.cl/estadisticas-delictuales/>
- Unbiased Coder. (2024). Obtenido de Python Django vs Flask.: <https://unbiased-coder.com/python-vs-java-springboot-apis/>
- whitestack. (22 de Agosto de 2023). Obtenido de Microservicios: ¿Qué son y qué tipos existen? ¿Cómo funciona su arquitectura?: <https://whitestack.com/es/blog/microservicios/>