UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Seguridad en Plataforma SaaS de Gestión de Flota: Control de Acceso y Protección de Datos en FlotUp

Felipe A. Sandoval Cornejo

sandovalcornejo9@gmail.com

Gonzalo Mendoza
Profesor Guía

Rodrigo Pinochet
Profesor Correferente

Resumen: La gestión de flotas vehiculares en instituciones tanto públicas como privadas, enfrenta desafíos asociados a procesos manuales, falta de trazabilidad y limitada fiscalización del uso de los recursos. Este escenario impacta en la eficiencia operativa y en la transparencia administrativa. En respuesta, se propone FlotUp, una plataforma SaaS de gestión inteligente de flotas que centraliza la solicitud, asignación y control de viajes, garantizando trazabilidad, eficiencia y seguridad de la información. El presente trabajo se enfoca en el diseño e implementación de mecanismos de seguridad para la plataforma, considerando protección de credenciales mediante hashing, autenticación basada en tokens, implementación de rate-limiting en inicio de sesión, medidas contra vulnerabilidades reconocidas por estándares de seguridad como OWASP y control de acceso por roles. La validación se realizará mediante pruebas funcionales y documentales, verificando la efectividad de las medidas implementadas y el cumplimiento de los objetivos de seguridad. Como resultado, se logrará fortalecer la confidencialidad, integridad y disponibilidad de la plataforma, estableciendo una base sólida para su despliegue en la Municipalidad de Viña del Mar como cliente piloto y su posterior adopción en otras organizaciones.

Palabras Clave: FlotUp, Gestión de flotas, seguridad informática, autenticación, control de acceso, protección de datos.

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

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

Título del trabajo: Seguridad en Plataforma SaaS de Gestión de Flota: Control de Acceso y Protección de Datos en FlotUp

Nombre del candidato(a): Felipe Andrés Sandoval Cornejo

Carrera / Grado: Ingeniería en Informática

Campus: Sede Viña del Mar **Departamento:** Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Gonzalo Mendoza Cárdenas, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

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

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

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

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

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

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

_____

_____

_____

4.- FIRMAS

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

Fecha: 24-02-2026 **Firma:** _____

Estudiante o Candidato(a):

Fecha: 24-02-2026 **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.

1 Introducción

La flota municipal constituye un recurso estratégico fundamental para el funcionamiento operativo de la institución. A través de estos vehículos se materializan gran parte de las actividades en terreno, las gestiones interdepartamentales y las labores de coordinación con la comunidad. Su disponibilidad permite que funcionarios y equipos técnicos puedan desplazarse de manera oportuna para ejecutar tareas administrativas, operativas y de servicio público.

Al estar destinada exclusivamente al personal municipal, la flota requiere una administración responsable y transparente que garantice un uso eficiente, equitativo y alineado con los objetivos institucionales.

Dado que se trata de un recurso limitado y de alto costo operativo, su gestión adecuada impacta directamente en la continuidad del trabajo administrativo, la planificación de actividades, y la capacidad de respuesta del municipio ante las necesidades de la ciudadanía.

1.1 Presentación del desafío

En la actualidad, la gestión de flotas vehiculares en instituciones públicas —como las municipalidades— y en diversas organizaciones privadas continúa realizándose, en gran medida, mediante procedimientos manuales: la coordinación de solicitudes de viajes se efectúa por vía telefónica, los registros permanecen dispersos en planillas locales, las asignaciones dependen del criterio subjetivo de los encargados de transporte y el control sobre el uso de vehículos y conductores es limitado.

Este escenario provoca:

- Ineficiencias operativas (tiempos de espera innecesarios y uso poco óptimo de los recursos)
- Falta de trazabilidad (dificultad para auditar qué vehículo fue usado, por quién y en qué condiciones)
- Ausencia de transparencia en la administración, un factor crítico especialmente en organismos públicos que deben rendir cuentas de manera clara a la ciudadanía.

Un aspecto fundamental a considerar es que, bajo la lógica institucional de las empresas del sector público, toda solicitud de viaje debe ser revisada y aprobada por el administrador de la flota, quien mantiene la responsabilidad final sobre la gestión de los vehículos y conductores. Esto obedece tanto a restricciones normativas propias de los organismos públicos, como a una cultura organizacional donde la autorización jerárquica asegura la responsabilidad administrativa y la rendición de cuentas.

De esta forma, el desafío no se limita únicamente a digitalizar los procesos, sino también a comprender y respetar este contexto institucional: un escenario en que los responsables de la flota necesitan agilidad y apoyo en la toma de decisiones, sin perder el control ni la capacidad de validar cada movimiento de los recursos bajo su cargo.

La falta de un sistema digital integrado dificulta la toma de decisiones basada en datos, la priorización de solicitudes según criterios objetivos y la implementación de políticas modernas de movilidad (medición de tiempos, distancias y niveles de ocupación) y sustentabilidad (reducción del consumo de combustible y de emisiones contaminantes, evitando traslados redundantes).

En consecuencia, surge la necesidad de una solución tecnológica centralizada y confiable, capaz de transformar estos procedimientos manuales en procesos digitales, eficientes y seguros.

1.2 Descripción de la propuesta de solución general

La propuesta consiste en el desarrollo de FlotUp, una plataforma digital bajo el modelo SaaS (Software como Servicio), entendido como un enfoque en el que las aplicaciones se entregan a los usuarios a través de Internet y son administradas completamente por el proveedor, quien se encarga de la infraestructura y el mantenimiento del servicio [1]. FlotUp está orientada a la gestión inteligente de flotas vehiculares en instituciones públicas y privadas. Su objetivo principal es modernizar la forma en que se administran los recursos de transporte, integrando en un solo sistema todos los procesos que actualmente se realizan de manera manual, dispersa y de forma ineficiente.

El diseño de la solución considera la intervención de tres actores principales dentro del sistema:

- Funcionario: es quien registra directamente su solicitud de traslado a través de la plataforma. Desde allí puede revisar la información del viaje asignado, visualizar la ruta planificada, el punto de partida, el destino y la ubicación en tiempo real del vehículo.

- Administrador: tiene a su cargo la supervisión de la flota. A partir de las recomendaciones automáticas que entrega el sistema, evalúa y decide si aprueba, rechaza o modifica las asignaciones de viaje. Mantiene control total sobre el proceso y acceso integral a los datos de la organización.

- Conductor: opera mediante una aplicación móvil con una interfaz sencilla, diseñada para facilitar su uso durante la conducción. Solo puede acceder al viaje activo y reporta su ubicación en tiempo real, permitiendo el seguimiento continuo y seguro del trayecto.

Como funcionalidades generales, la plataforma contempla:

- Solicitud en línea de viajes por parte de funcionarios autorizados, desde plataforma web.
- Propuesta automática de asignación de vehículos (basada en criterios objetivos como disponibilidad, horarios, capacidad y ubicación).
- Monitoreo en tiempo real de las solicitudes y de la utilización de la flota.
- Trazabilidad completa, con registro de cada etapa del proceso, desde la creación de la solicitud hasta la finalización del viaje.
- Transparencia administrativa, al centralizar la información en una base de datos que permite auditar y justificar decisiones.
- Entrega de reportes para la toma de decisiones basadas en datos (porcentaje de utilización de flota, conductores con más viajes, destinos más solicitados, etc.)

Un aspecto central es que la herramienta no elimina el rol del administrador de flota, sino que lo fortalece. El sistema entrega sugerencias inteligentes de asignación, pero la decisión final continúa en manos del administrador, quien conserva la responsabilidad de aprobar o modificar cada solicitud en conformidad con las normativas institucionales y la lógica jerárquica de gestión.

La digitalización del proceso permite que cada solicitud y viaje quede registrado digitalmente, lo que facilita el seguimiento y control en tiempo real, abriendo paso a una fiscalización más efectiva y transparente del uso de los vehículos municipales. Este nivel de trazabilidad contribuye directamente a una gestión más eficiente de los recursos, al reducir gastos asociados a combustible, mantenciones y tiempos no productivos. Además, la plataforma optimiza la asignación y coordinación entre vehículos y conductores, disminuyendo los tiempos de espera y aumentando la productividad general del sistema.

La Municipalidad de Viña del Mar será el cliente piloto de la solución, lo que permitirá validar su funcionamiento en un entorno real de alta exigencia, caracterizado por la necesidad de eficiencia, transparencia y rendición de cuentas. La experiencia obtenida en este piloto servirá como base para proyectar y estandarizar la plataforma hacia otras municipalidades y organizaciones privadas que requieran modernizar su gestión de flotas.

Con el fin de representar de manera estructurada el flujo operativo propuesto para la gestión de viajes en la plataforma FlotUp, se presenta el siguiente diagrama BPMN, el cual describe los actores involucrados, las actividades ejecutadas, los puntos de decisión y la secuencia de eventos dentro del sistema de forma general.

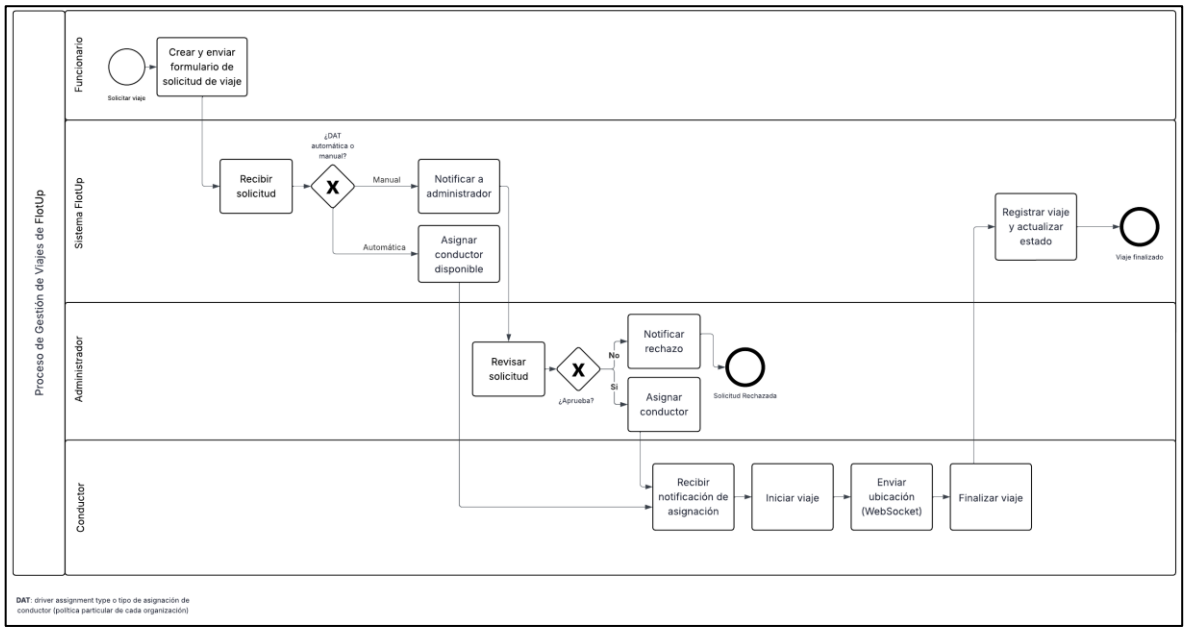

Figura 1-1. Diagrama BPMN del proceso de gestión de viaje de la plataforma FlotUp
Fuente: Elaboración propia

1.3 Descripción de cómo aporta el trabajo a la solución general

El aporte específico de este trabajo dentro del desarrollo de la plataforma FlotUp, se centra en la seguridad informática, aspecto esencial para garantizar la confiabilidad de una solución tecnológica que gestionará recursos públicos y privados. En este contexto, el trabajo se orienta a implementar y analizar mecanismos que aseguren que solo usuarios autorizados puedan acceder al sistema, que las credenciales y datos sensibles se encuentren debidamente protegidos, y que se establezcan controles que mitiguen riesgos propios de las aplicaciones web modernas.

Si bien el desarrollo inicial se realiza en un entorno local, la plataforma está proyectada para ser desplegada en el servicio de nube Google Cloud Platform (GCP) durante la etapa de pilotaje. No obstante, los mecanismos de seguridad abordados se diseñan de manera transversal, procurando que la aplicación sea independiente del proveedor de servicios de nube específico y pueda adaptarse a infraestructuras equivalentes sin requerir modificaciones sustanciales.

En términos prácticos, el trabajo abarca:

- El diseño y puesta en marcha de un esquema de autenticación y autorización robusto, basado en el uso de credenciales seguras, cifrado de contraseñas y emisión de tokens de acceso controlados.
- La incorporación de un modelo de control de acceso que limite de manera estricta las acciones que cada usuario puede ejecutar, de acuerdo con su rol dentro de la organización (funcionario, conductor o administrador de flota).
- La integración de mecanismos de protección frente a ataques comunes descritos en estándares de seguridad reconocidos como OWASP, CWE y CVE (ataques de fuerza bruta, inyecciones, XSS y CSRF, etc.)
- La aplicación de buenas prácticas de seguridad a nivel de infraestructura, mediante el uso de servicios perimetrales que refuercen la defensa contra ataques de denegación de servicio, accesos no autorizados o intentos de explotación.

De esta manera, el trabajo no solo fortalece a la solución general en lo tecnológico, sino que también incrementa la confianza y legitimidad de la plataforma, al asegurar que su adopción por parte de la Municipalidad de Viña del Mar —cliente piloto— y cualquier institución (sector público como privado) cumpla con estándares adecuados de seguridad, trazabilidad y protección de información.

1.4 Objetivo general de la propuesta de solución

Desarrollar una plataforma SaaS de gestión inteligente de flotas vehiculares que centralice y digitalice el proceso de solicitudes, asignaciones y control de viajes, fortaleciendo la trazabilidad, la eficiencia operativa y la transparencia en el uso de recursos.

1.5 Objetivo general del trabajo de título

Analizar, diseñar e implementar mecanismos de seguridad y control de acceso en la plataforma FlotUp, para garantizar la protección de credenciales, datos sensibles y la correcta administración de permisos de usuario.

1.6 Objetivos específicos del trabajo de título

1. Analizar algoritmos de hashing reconocidos para contraseñas, evaluando su nivel de seguridad y desempeño en entornos SaaS, para seleccionar e implementar el más adecuado en la plataforma FlotUp.

2. Adoptar un módulo de autenticación basado en tokens con expiración y renovación controlada para la aplicación móvil de conductores, asegurando que los endpoints críticos consumidos por este cliente solo puedan ser accedidos por usuarios autenticados.

3. Aplicar un sistema de rate-limiting en el endpoint de inicio de sesión que restrinja la cantidad de intentos fallidos de autenticación, con el fin de mitigar ataques de fuerza bruta mediante bloqueos temporales de acceso.

4. Implementar medidas de seguridad contra los principales ataques mencionados en estándares de seguridad reconocidos como OWASP.

5. Definir y configurar un esquema de control de acceso basado en roles utilizando el modelo de la plataforma, estableciendo los roles principales con permisos diferenciados sobre las vistas del sistema.

6. Configurar medidas perimetrales que incluyan: certificado SSL/TLS, protección WAF básica y reglas de mitigación contra DDoS, evidenciando el resultado con métricas de bloqueo y mitigación registradas en la herramienta.

7. Evaluar la efectividad de los mecanismos de seguridad implementados mediante pruebas de validación técnica, con el fin de verificar el cumplimiento de los principios de confidencialidad, integridad y disponibilidad (CIA) del sistema.

1.7 Metodologías a utilizar

El desarrollo del proyecto se condujo bajo el marco ágil Scrum, elegido por su capacidad de adaptación frente a cambios en los requerimientos y por fomentar entregas parciales y verificables en cada iteración. Antes de iniciar el primer sprint, se llevaron a cabo actividades de ideación y diseño temprano, que incluyeron el uso de herramientas de prototipado (Figma) y dinámicas participativas con el patrocinador, con el propósito de validar la usabilidad y el flujo de interacción. En lo relativo a la seguridad, este trabajo se sustenta en una metodología de investigación aplicada, la cual integra tres componentes:

- Revisión bibliográfica y estado del arte (algoritmos de hashing, modelos de control de acceso, medidas OWASP).

- Implementación práctica de los controles de seguridad en la plataforma FlotUp.

- Pruebas y validación mediante simulaciones de ataques y uso de herramientas de análisis.

1.8 Plan de trabajo

Sprint 1 – Fundamentos de Seguridad y Autenticación:

- Integración y prueba de algoritmos de hashing para contraseñas, evaluando al menos 3 algoritmos (PBKDF2, bcrypt, Argon2id) y comparando tiempos de hashing.
- Implementación inicial del módulo de autenticación con JWT, asegurando que el 100% de los endpoints críticos requieran token válido para su acceso.
- Configuración básica de dependencias y parámetros de seguridad en Django, verificando que el 100 % de las configuraciones sensibles (SECRET_KEY, DEBUG, CORS, CSRF) estén correctamente deshabilitadas o ajustadas para los diferentes entornos (desarrollo, producción).

Sprint 2 – Controles de Acceso y Protección contra Ataques:

- Aplicación de rate-limiting en el endpoint de inicio de sesión, validando que el sistema bloquee solicitudes después de 5 intentos fallidos por minuto por IP.
- Configuración y documentación de medidas contra los principales ataques OWASP Top 10 (CSRF, XSS, inyección, etc).
- Definición y aplicación de un esquema de control de acceso basado en roles (administrador, conductor, funcionario), verificando que el 100 % de las vistas protegidas respondan correctamente al modelo RBAC definido.

Sprint 3 – Seguridad Perimetral, Infraestructura y Pruebas:

- Configuración de certificados SSL/TLS.
- Activación de reglas WAF y medidas de mitigación de DDoS.
- Ejecución de pruebas de validación de seguridad mediante simulaciones controladas, evaluando que las medidas implementadas detecten, mitiguen o bloqueen adecuadamente los intentos de vulneración en la plataforma.

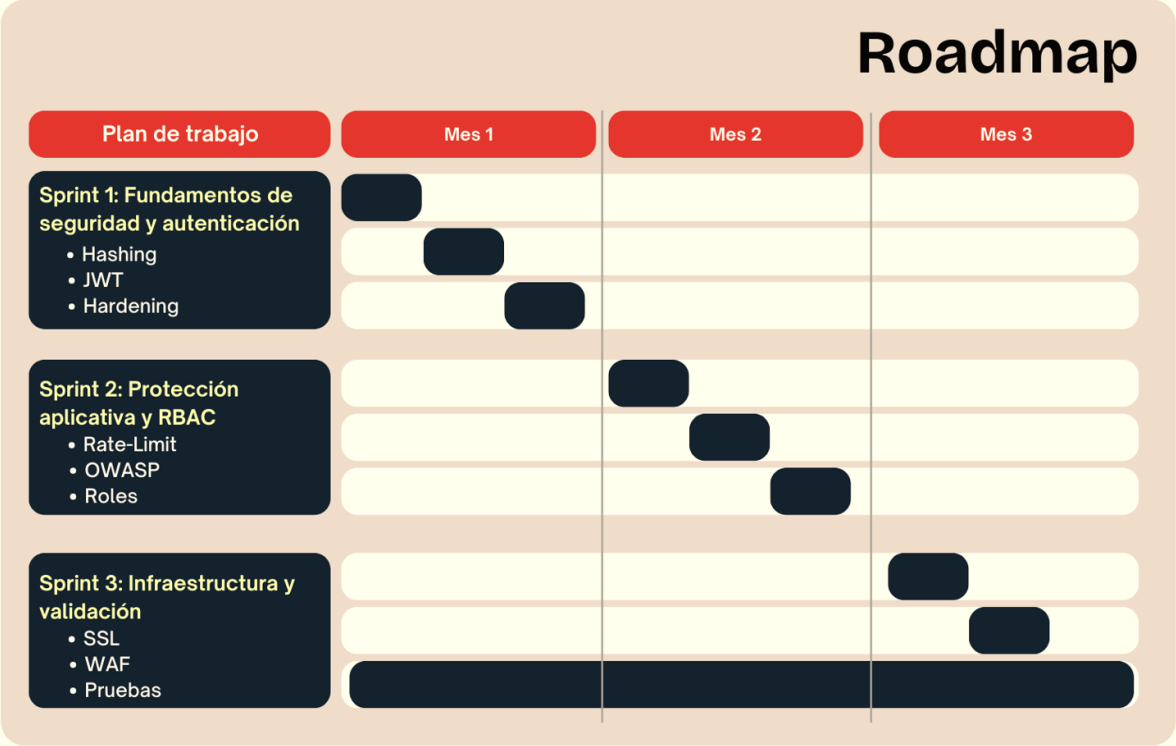

Figura 1-2. Roadmap de implementación de los mecanismos de seguridad en FlotUp
Fuente: Elaboración propia

2 Marco Conceptual

La seguridad en aplicaciones web bajo el modelo Software como Servicio (SaaS) constituye un elemento crítico, ya que estas plataformas concentran múltiples usuarios y organizaciones que operan de manera simultánea en entornos compartidos. En este contexto, proteger identidades, credenciales y datos sensibles resulta fundamental para garantizar la continuidad y confiabilidad del servicio. Asimismo, la naturaleza distribuida

del modelo SaaS las expone a vulnerabilidades propias de las aplicaciones web modernas, lo que demanda mecanismos adecuados de autenticación, control de acceso, criptografía y mitigación frente a ataques comunes.

Bajo este marco, el presente apartado introduce los conceptos esenciales que sustentan las medidas de seguridad implementadas en FlotUp, abordando temas como hashing de contraseñas, autenticación mediante tokens, modelos de control de acceso, lineamientos OWASP y elementos de protección en infraestructura. Estos fundamentos permiten contextualizar técnicamente las decisiones adoptadas en el desarrollo de la solución y dar a conocer en detalle los componentes que se presentan en las secciones posteriores.

2.1 Hashing de contraseñas

El almacenamiento seguro de contraseñas depende del uso de algoritmos de hashing robustos con técnicas de salting e iteraciones múltiples. En la literatura se identifican enfoques tradicionales como PBKDF2, de amplia adopción en frameworks, y alternativas más recientes como bcrypt y Argon2, siendo este último, seleccionado como el algoritmo recomendado en el Password Hashing Competition por su resistencia frente a ataques con hardware especializado y sus configuraciones de memoria y tiempo ajustables [2].

2.2 Autenticación con tokens JWT

El uso de JSON Web Tokens (JWT) se ha consolidado como un mecanismo habitual de autenticación y autorización en arquitecturas web y móviles [3]. Esto permite sesiones sin estado (stateless), facilitando la escalabilidad en entornos distribuidos. El token contiene un payload firmado digitalmente, que integra datos de identificación del usuario y una fecha de expiración. No obstante, su adopción requiere prácticas seguras como expiraciones cortas, uso de refresh tokens y mecanismos de invalidación (blacklisting) para evitar reutilización indebida.

2.3 Modelos de control de acceso

La literatura distingue diversos enfoques de control de acceso:

- RBAC (Role-Based Access Control) [4]: asigna permisos a roles definidos (ej. administrador, conductor, funcionario), simplificando la gestión en entornos organizacionales.
- ABAC (Attribute-Based Access Control) [4]: incorpora atributos dinámicos (ej. horario, ubicación, estado del recurso), permitiendo reglas más granulares, aunque a mayor complejidad.

En contextos institucionales como los municipales, el uso de RBAC es frecuente por su claridad y alineación con estructuras jerárquicas.

2.4 Medidas contra ataques comunes

El OWASP Top 10 identifica vulnerabilidades críticas recurrentes en aplicaciones web, tales como inyección de código, fallas de autenticación, exposición de datos sensibles, XSS (Cross-Site Scripting) o CSRF (Cross-Site Request Forgery). Estas guías constituyen una referencia internacional para diseñar estrategias de protección que deben integrarse desde las etapas tempranas de desarrollo. [5]

2.5 Herramientas de protección en infraestructura

En el plano de la infraestructura, soluciones perimetrales como firewalls de aplicaciones web (WAF), mitigación de ataques de denegación de servicio (DDoS) y el uso obligatorio de TLS/SSL se consideran prácticas estándar. Proveedores como Cloudflare o Amazon CloudFront ofrecen servicios que incorporan estas funciones de forma integrada, contribuyendo a la defensa en capas de las aplicaciones SaaS [6].

3 Estado del Arte

El desarrollo de FlotUp se inserta en un ecosistema tecnológico donde las plataformas de gestión de movilidad y flotas no solo buscan optimizar la asignación de recursos, sino también garantizar seguridad y confianza en el intercambio de información y la operación de los sistemas. La protección de los datos personales, la autenticación segura de los usuarios y la integridad de las comunicaciones son hoy pilares esenciales en las soluciones de movilidad inteligente, especialmente en contextos donde convergen múltiples actores y datos sensibles.

3.1 Seguridad en plataformas de viajes compartidos

Empresas como Lyft Line y Uber Pool han implementado sistemas de asignación dinámica y optimización de rutas para conectar pasajeros con trayectos compatibles, reduciendo costos y mejorando la eficiencia operativa. Ambas plataformas también integran medidas de seguridad digital orientadas a proteger la información del usuario y la integridad del servicio.

Lyft, por ejemplo, aplica cifrado de extremo a extremo y verificación en tiempo real de identidad para conductores y pasajeros, asegurando que las transacciones de viaje se realicen en entornos confiables [7].

De forma similar, Uber ha fortalecido su modelo de seguridad implementando autenticación multifactor, revisión automatizada de actividad sospechosa y almacenamiento cifrado de datos de geolocalización, minimizando el riesgo de accesos indebidos o manipulación de información [8].

3.2 Seguridad en sistemas de gestión de flotas

En el ámbito corporativo, la seguridad de la información y la integridad operacional son componentes esenciales de las plataformas de gestión de flota. Soluciones como Geotab y Samsara destacan por integrar capas de protección tanto a nivel de software como de infraestructura.

Geotab incorpora un enfoque de seguridad por diseño, con certificaciones ISO/IEC 27001 y cumplimiento del marco GDPR, garantizando la protección de datos de telemetría, ubicación y comportamiento de conducción [9].

Por su parte, Samsara combina el uso de IoT seguro, comunicaciones cifradas y autenticación mediante tokens para proteger la información transmitida entre dispositivos y servidores, reduciendo la exposición ante ataques o intrusiones externas [10].

Estas experiencias internacionales demuestran que la incorporación de controles de seguridad integrados no solo mejora la confiabilidad de las plataformas, sino que también fortalece la transparencia y la trazabilidad de los procesos, principios que FlotUp adopta para su implementación en el contexto público e institucional.

4 Desarrollo, resultados y validación de la solución

Antes de detallar el desarrollo de cada uno de los mecanismos de seguridad implementados, es necesario contextualizar la arquitectura general sobre la cual opera FlotUp.

En la siguiente figura se presenta la arquitectura completa (general) de la solución SaaS.

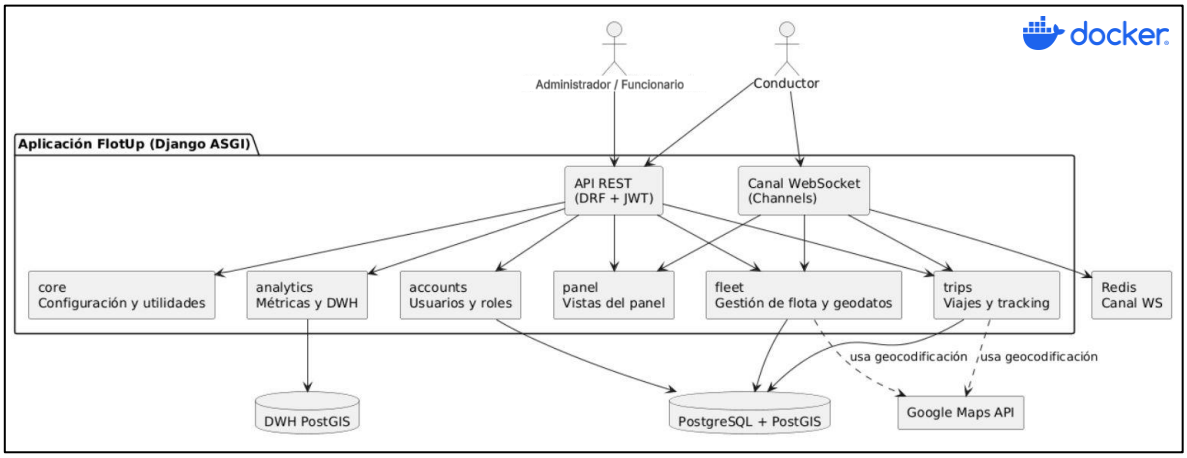

Figura 4-1. Arquitectura general de la plataforma FlotUp
Fuente: Elaboración propia

En la siguiente figura se muestra la arquitectura particular de seguridad implementada en este trabajo de título, la cual permite visualizar los componentes, sus responsabilidades y las interacciones que sustentan los mecanismos de seguridad descritos en los apartados posteriores.

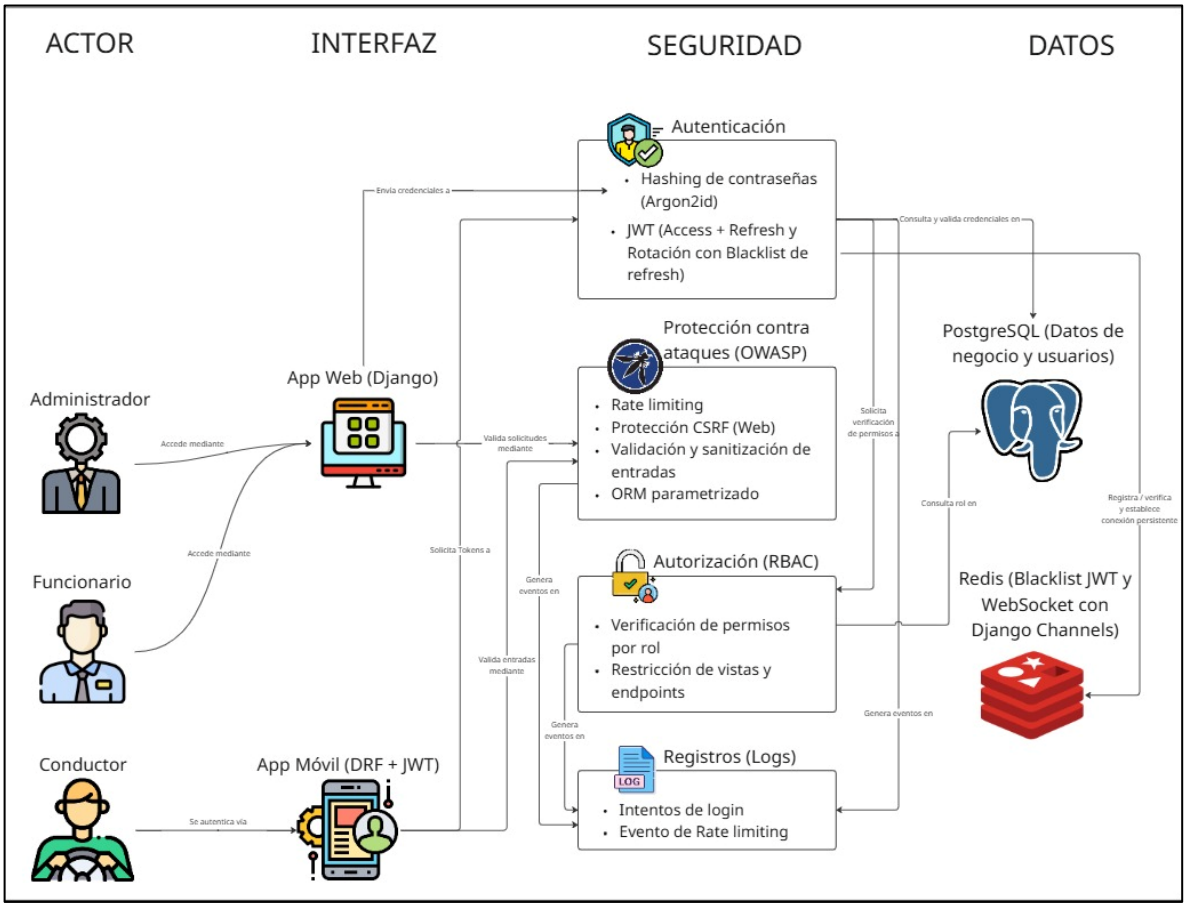

Figura 4-2. Arquitectura particular de los mecanismos de seguridad de FlotUp
Fuente: Elaboración propia

4.1 Objetivo 1 — Análisis e implementación del algoritmo de hashing

4.1.1 Desarrollo

Definición de hashing

El hashing de contraseñas es un proceso criptográfico que transforma una contraseña en una representación irreversible mediante una función matemática unidireccional. En lugar de almacenar la contraseña en texto plano, el sistema almacena su hash, así, durante el inicio de sesión, la contraseña ingresada se vuelve a hashear y se compara con el valor registrado. Este enfoque impide que un atacante que obtenga la base de datos pueda leer las contraseñas directamente, y permite aplicar mecanismos de seguridad adicionales.

Origen y evolución

El hashing de contraseñas surge en los primeros sistemas multiusuario, donde almacenar contraseñas en texto plano representaba un riesgo directo. Con la evolución de los ataques, especialmente los offline tras filtraciones de bases de datos, surgió la necesidad de algoritmos más robustos que incorporaran mecanismos de ralentización y resistencia a hardware paralelo.

Tipos de algoritmos actuales:

Los algoritmos de derivación de claves utilizados para proteger contraseñas pueden agruparse en cuatro categorías principales:

1. CPU-bound: aumentan el costo computacional mediante múltiples iteraciones. Su eficacia es limitada frente a GPUs y ASICs debido a la paralelización masiva.
2. Compute-hard: diseñados para ser lentos, pero no requieren grandes cantidades de memoria, lo que permite que hardware paralelo acelere ataques.
3. Memory-hard: obligan a consumir grandes cantidades de RAM por intento, lo que reduce significativamente la eficiencia de GPUs y vuelve económicamente inviable el uso de ASICs.
4. Modelos avanzados asistidos por hardware o protocolos modernos: aún en investigación y poco integrados en frameworks tradicionales.

Análisis comparativo:

Para esta comparativa la selección de los tres algoritmos siguientes no es arbitraria, corresponden a los tres estándares más representativos y utilizados actualmente para el hashing seguro de contraseñas, cubriendo tres generaciones distintas de evolución en este ámbito.

1. PBKDF2 (SHA-256): CPU-bound, estándar, amplio soporte, fácil de configurar en Django.
2. bcrypt: buena resistencia a fuerza bruta, limita el largo efectivo de la contraseña (72 bytes), costo de memoria bajo.
3. Argon2id: memory-hard, algoritmo moderno recomendado por OWASP, híbrido (resiste ataques de canal lateral y GPU) y con soporte nativo en Django vía "argon2-cffi".

Un aspecto crítico en la selección de algoritmos de hashing para contraseñas es la resistencia frente a ataques offline, los cuales se producen cuando un atacante obtiene los hashes almacenados en la base de datos y ejecuta técnicas de fuerza bruta sin restricciones del sistema original (ej: rate-limiting, IP Block, MFA).

En este escenario, resulta habitual el uso de hardware altamente paralelo, como:

- GPUs (tarjeta gráfica, las cuales tienen miles de núcleos pequeños diseñados para ejecutar operaciones simples en paralelo) capaces de calcular millones de hashes por segundo
- ASICs (Application-Specific Integrated Circuit, chips diseñado para hacer solo una cosa, pero extremadamente rápido) diseñados específicamente para acelerar algoritmos simples.

Frente a este tipo de amenazas, los algoritmos memory-hard ofrecen una protección significativamente superior, ya que requieren grandes cantidades de memoria por intento, reduciendo drásticamente la eficiencia de las GPUs y haciendo económicamente inviable la fabricación de ASICs específicos para romperlos.

Un esquema de protección de contraseñas requiere mecanismos complementarios que refuercen su resistencia frente a ataques offline. Entre estos mecanismos se encuentran:

- Salt: cadena aleatoria única generada por usuario, añadida antes del proceso de hashing. Su función es evitar que dos contraseñas iguales produzcan el mismo hash y, al mismo tiempo, neutralizar el uso de tablas precomputadas como rainbow tables.

- Pepper: valor secreto definido a nivel de aplicación y almacenado fuera de la base de datos. Se concatena a la contraseña antes del hashing, de modo que incluso si un atacante obtiene la base de datos completa, no pueda realizar ataques offline sin conocer este valor externo.

Estos mecanismos aplican a cualquiera de los algoritmos comparados y constituyen una capa adicional de defensa, especialmente relevante para sistemas multiusuario como FlotUp.

CRITERIO / ALGORITMO	ALGORITMO 1 PBKDF2 (SHA-256)	ALGORITMO 2 bcrypt	ALGORITMO 3 Argon2id
Tipo de algoritmo	CPU-bound	Compute-hard	Memory-hard
Resistencia a GPU/ASIC	Baja, Altamente paralelizable	Media, GPU acelera ataques	Alta, Requiere RAM por intento, inhibe paralelismo
Consumo de memoria	Bajo	Bajo	Alto (configurable)
Límite de longitud	Sin límite práctico	~ 72 bytes	Sin límite práctico
Soporte en Django	Nativo	Nativo	Paquete argon2-cffi (soporte completo)
Ajuste de parámetros	Iteraciones	Cost (work factor)	Memoria, tiempo, paralelismo
Robustez actual	Buena, pero superada	Media, vulnerable a GPUs modernas	Excelente, recomendado por OWASP y PHC
Velocidad de hashing	Rápido (provoca GPU attacks)	Moderado	Controlada (memory-hard)
Adecuado para SaaS	Aceptable, pero menos seguro	Aceptable, pero no óptimo	Para entornos multiusuario y despliegues cloud
Complejidad de ataque offline	Baja, Millones de hashes en GPU	Media, Cientos de miles/s en GPU	Muy alta, GPU necesita cientos de MB por hilo

Tabla 4-1. Comparativa de características entre algoritmos de hashing
Fuente: Elaboración propia

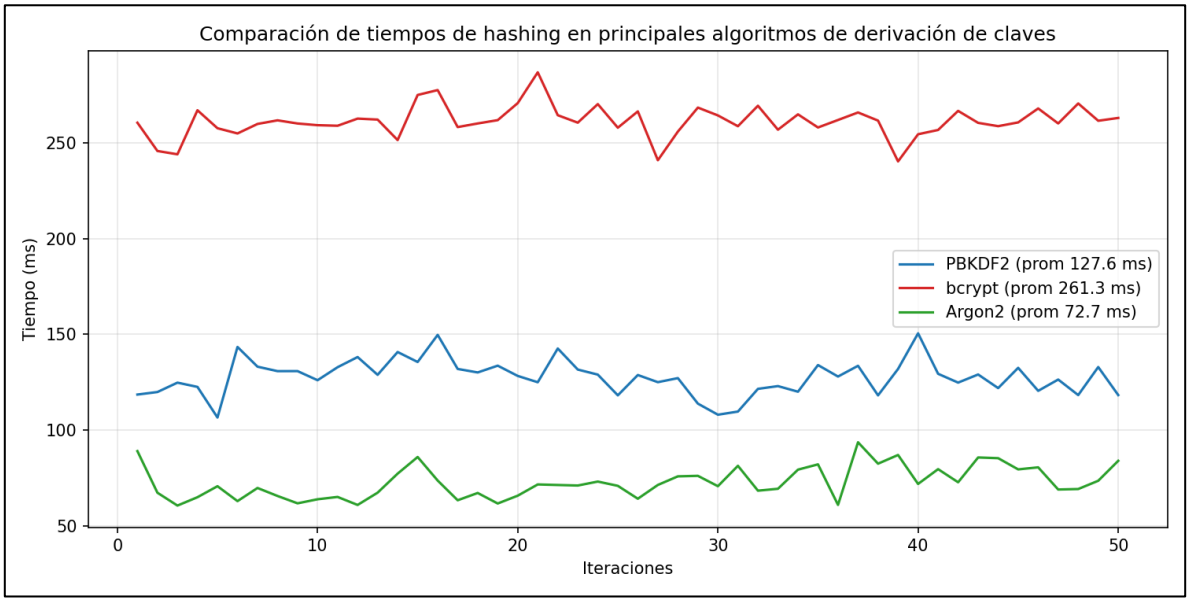

Figura 4-3. Comparación de tiempos de hashing en algoritmos
Fuente: Elaboración propia

Justificación técnica de la elección de Argon2id para FlotUp:

A partir de los datos presentados anteriormente, es posible sustentar técnicamente la elección de Argon2id como algoritmo principal de hashing para FlotUp.

En primer lugar, el criterio de alta resistencia frente a ataques offline se ve reflejado explícitamente en la tabla comparativa, mientras PBKDF2 y bcrypt pueden ser procesados de forma eficiente por GPU (millones y cientos de miles de hashes por segundo, respectivamente), Argon2id incorpora un enfoque memory-hard que obliga a reservar grandes cantidades de memoria por intento, inhibiendo la paralelización masiva y elevando significativamente el costo computacional para un atacante.

En segundo lugar, se toma en consideración el criterio de compatibilidad directa con Django y despliegues en contenedores, ya que Argon2id cuenta con soporte completo mediante el paquete argon2-cffi en Django y permite ajustar sus parámetros para entornos como Docker o Cloud Run sin afectar la portabilidad ni la estabilidad del servicio.

En tercer lugar, se toma en cuenta el tiempo de hashing, la figura asociada muestra que Argon2id mantiene tiempos promedios cercanos a 70–80ms por operación, muy inferiores a bcrypt y considerablemente más eficientes que configuraciones equivalentes de PBKDF2. Esto permite que el inicio de sesión permanezca ágil para el usuario final, al mismo tiempo que el algoritmo impone un costo suficiente para dificultar ataques de fuerza bruta offline.

Finalmente, la naturaleza parametrizable de Argon2id permite implementar una política de defensa reforzada mediante salt por usuario, pepper aplicacional y ajuste fino de memoria, tiempo y paralelismo, cumpliendo así con el enfoque de defensa en profundidad propuesto para FlotUp. La combinación de estos factores evidencia, de forma coherente con los datos obtenidos, que Argon2id es la alternativa más robusta, eficiente y alineada con los requisitos operativos y de seguridad de la plataforma.

Decisión:

En base a los criterios e información presentada anteriormente, se implementará Argon2id.

Implementación:

El algoritmo Argon2id opera dentro del backend Django, el cual ese ejecuta en contenedores Docker que utilizan las capacidades de CPU asignadas por el entorno de ejecución.

Para la correcta implementación, se deben realizar los siguientes pasos:

1. Activar Argon2id como hasher principal y parametrizarlo:

   Se selecciona Argon2id y se establece la configuración de parámetros por defecto del algoritmo con django, que equilibran seguridad con rendimiento en contenedores (infraestructura base de FlotUp).

   Este algoritmo permite ajustar tres parámetros principales que determinan su nivel de seguridad y su costo computacional. Estos parámetros permiten adaptar el algoritmo al hardware disponible, aumentando la resistencia frente a ataques de fuerza bruta, especialmente ataques offline basados en GPU o ASICs.

   A continuación, se describen los parámetros utilizados y sus justificaciones para la infraestructura de FlotUp.

   time_cost (número de iteraciones del algoritmo)

   Representa la cantidad de veces que el algoritmo ejecuta el proceso de hashing completo.
   Cuanto mayor es este valor, más tiempo toma generar un hash.

   - Incrementa el tiempo necesario para verificar una contraseña.
   - Dificulta ataques offline, pues cada intento requiere más tiempo.

   Se selecciona time_cost = 2, porque, con 2 iteraciones y un parámetro acorde de 'memory_cost', el tiempo de hashing se ubica entre 50 y 150 ms, aceptable para un login real. En contenedores (Cloud Run o Docker), tiempos mayores pueden generar latencia innecesaria y es el equilibrio perfecto entre usabilidad (inicio de sesión rápido) y seguridad (tiempo significativo para impedir millones de intentos por segundo).

   memory_cost (memoria utilizada en MiB)

   Define la cantidad de memoria RAM que el algoritmo debe reservar para procesar el hash. Este es el parámetro más importante debido a que Argon es 'memory-hard', ósea que obliga a los atacantes a usar grandes cantidades de RAM, lo cual limita la eficiencia de GPUs y hardware especializado.

   - Subir la memoria aumenta exponencialmente el costo de un ataque.
   - Reduce la viabilidad de ataques paralelos.

Se seleccionó memory_cost = 102400 (≈100 MiB) porque, los contenedores usados en desarrollo y producción cuentan con al menos 2 GB de RAM disponibles, consume <5 % de la memoria por operación, sin comprometer estabilidad del backend, representa un costo extremadamente alto para ataques GPU (una GPU moderna de 8 a 12 GB podría ejecutar solo 80–120 hashes simultáneos, reduciendo drásticamente su ventaja paralela).

parallelism (cantidad de hilos paralelos)

Define cuántos hilos de CPU se usarán durante el hashing, el algoritmo divide internamente la memoria en "lanes" (memoria reservada para generar el hash) paralelos, una por cada hilo.

- Mejora el rendimiento aprovechando múltiples núcleos.
- No reduce la seguridad (Argon2 está diseñado para ello).

Se fijó parallelism = 8 porque, las máquinas donde corre FlotUp (entornos Docker locales en debug y Cloud Run en despliegue) suelen tener 4 a 8 vCPUs disponibles, permite que el hashing se ejecute de manera eficiente sin saturar la CPU, no afecta la usabilidad ya que el proceso es muy corto y paralelo permitiendo un uso eficiente del hardware disponible y evitando latencias innecesarias.

Configuraciones particulares:

```
# Hash de contraseñas: Argon2 como principal, con fallbacks para
migración de contraseñas legadas
PASSWORD_HASHERS = [
    "apps.accounts.utils.hashers.PepperArgon2PasswordHasher",    # Argon2
con pepper (personalizado)
    "django.contrib.auth.hashers.Argon2PasswordHasher",          #
Fallback Argon2 sin pepper (contraseñas legadas)
]
```

Figura 4-4. Configuración de gestores de contraseñas en Django
Fuente: Elaboración propia

Configuraciones extendidas de django framework:

```
class Argon2PasswordHasher(BasePasswordHasher):
    """
    Secure password hashing using the argon2 algorithm.

    This is the winner of the Password Hashing Competition 2013-2015
    (https://password-hashing.net). It requires the argon2-cffi library
which
    depends on native C code and might cause portability issues.
    """
    algorithm = "argon2"
    library = "argon2"

    time_cost = 2
    memory_cost = 102400
    parallelism = 8
```

Figura 4-5. Parámetros técnicos del algoritmo Argon2id
Fuente: Elaboración propia

2. Incorporar "pepper" aplicacional

Además del salt aleatorio que maneja Django, se concatena el "pepper" secreto (extraído desde las variables de entorno) antes del hashing. Esto agrega una capa adicional en caso de exposición de la base de datos.

Configuraciones particulares:

```
class PepperArgon2PasswordHasher(Argon2PasswordHasher):
    """
    Argon2id + pepper desde settings.PASSWORD_PEPPER.
    Se Concatena un "pepper" secreto (desde variables de entorno) antes
del hashing
    """

    algorithm = "argon2"

    def _pepperize(self, password: str) -> str:
        pepper = getattr(settings, "PASSWORD_PEPPER", "")
        return f"{password}{pepper}"

    def encode(self, password):
        # Siempre codificar con pepper
        return super().encode(
            self._pepperize(password),
        )
```

```python
def verify(self, password, encoded):
    # 1) Intento con pepper
    if super().verify(self._pepperize(password), encoded):
        return True
```

Figura 4-6. Implementación de clase para hashing con pepper aplicacional
Fuente: Elaboración propia

4.1.2 Validación

La validación de la configuración de hashing se realizó a través de pruebas orientadas a verificar que el algoritmo Argon2id se encuentre efectivamente activo, que las contraseñas no se almacenen en texto plano y que los parámetros definidos no afecten de forma negativa la experiencia de uso en FlotUp.

En primer lugar, se creó un conjunto de usuarios de prueba en FlotUp.

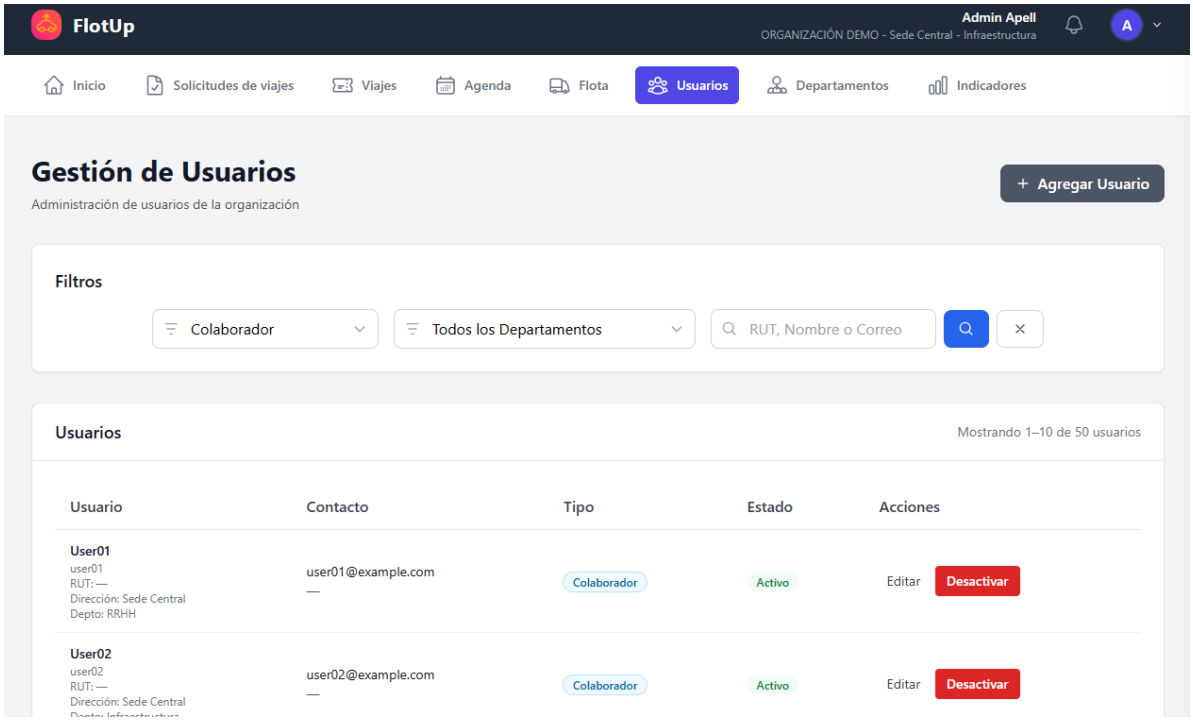

Figura 4-7. Interfaz de gestión de usuarios de FlotUp
Fuente: FlotUp

Posteriormente, se inspeccionó directamente la base de datos PostgreSQL para revisar los valores asociados al campo de contraseña en la tabla de usuarios.

```
C:\Users\sando\Escritorio\INF\OTROS\REPOSITORIOS\FLOTUP\feriasoftware2025>docker exec -it bd_postgis psql -U postgres -d flota -c "SELECT username, pass
word FROM auth_user WHERE username LIKE 'user%' ORDER BY username ASC LIMIT 10;"
 username |                                                                                                                password
----------+---------------------------------------------------------------------------------------------------------------------------------------
 user01   | argon2$argon2id$v=19$m=102400,t=2,p=8$UVZaSHRaSXVsZU14RFBFbU1tM3NNYg$qBKj3fKgNDzFRyQtI8A4msRHKx9B0Py4ephg5g8T5cg
 user02   | argon2$argon2id$v=19$m=102400,t=2,p=8$WTRGSUplYldPdlpoazM5TngzVExyMw$ykOZ8ymjAHwOBsUdvpCXqiWW4mdc2YEWN8x5nDIbgKE
 user03   | argon2$argon2id$v=19$m=102400,t=2,p=8$UDlHZDIyYXMzeVQzcGtzaGVOblJwZQ$y0JP5b0hA+2ijpWmVoM1kHWQRSGn94bJeySW4pKMRwc
 user04   | argon2$argon2id$v=19$m=102400,t=2,p=8$aEhEZ0FDZ0gwWFRKZFNBSU9jS1VzMA$M0Hzs1jGGxg2BOZ9lGMDlgOAMvd3nn3Lg9N4gxZ57WM
 user05   | argon2$argon2id$v=19$m=102400,t=2,p=8$b0RYYkJBT1hFanBMMXhvd0ZHbFk5cQ$5uQ3bjpK9GFLs6irHhxH6xVDtioDQ4yfOQ59hkq1BpA
 user06   | argon2$argon2id$v=19$m=102400,t=2,p=8$clZxTU5waVJyOEpVQnJaZU5rVTIxTg$jVsdmpKwNdaYK3Efgs5bP2NmhsnYZPfIEwd6A2sTRes
 user07   | argon2$argon2id$v=19$m=102400,t=2,p=8$aWFIeWZsWDVmNVdOQU1FWmMzekpkTw$oZPYdjZE/RCwZlqJlHiMW/AwRJvbrN4GuBO09S+Uv1c
 user08   | argon2$argon2id$v=19$m=102400,t=2,p=8$a0Y3bnlKM1hPSVlSbjBJTVJ1ZDBaMQ$HwAGPI4LzZbmSBSjU7TZkzwvEB6BQLypyZxlgsIECdQ
 user09   | argon2$argon2id$v=19$m=102400,t=2,p=8$Y0ZMelZIZDdzSnIxZGxYRXJFWG9xRA$TKNuaZ8GpZoJBy7ybS6Rto9dlYVstLpktxluc+2+oh4
 user10   | argon2$argon2id$v=19$m=102400,t=2,p=8$T25UMnhld3dCNG01Y2xoWXNsTUxpcg$jDoFvcgmIxE/wooOUSUJSRqgporeDehyZ8Q5p64GJuQ
(10 rows)
```

Figura 4-8. Inspección de hashes de contraseñas en base de datos PostgreSQL
Fuente: Elaboración propia

A partir de la inspección, se verificó que:

- Ninguna contraseña se almacenaba en texto plano.
- Todos los registros contenían hashes con el prefijo "argon2$argon2id", correspondiente al algoritmo configurado.
- Las nuevas cuentas creadas utilizaban Argon2id como hasher principal, confirmando que la configuración en PASSWORD_HASHERS estaba activa.

Adicionalmente, se comprobó el funcionamiento correcto del proceso de verificación de contraseñas mediante pruebas de inicio de sesión exitosas y fallidas, validando que:

- Una contraseña válida permitía la autenticación y generaba el flujo normal de acceso.

- Contraseñas incorrectas eran rechazadas sin entregar información adicional (mensajes de error genéricos), evitando filtración accidental de metadatos o condiciones del sistema.

Finalmente, se validó el uso del pepper aplicacional modificando temporalmente el valor definido en las variables de entorno. Tras este cambio, todas las contraseñas previamente almacenadas dejaron de ser válidas, confirmando que dicho valor participa efectivamente en el proceso de hashing. Luego de restaurar el valor original, los usuarios de prueba pudieron autenticarse nuevamente sin inconvenientes, cerrando así la validación completa del mecanismo.

Conformidad con estándares de seguridad:

La configuración de hashing implementada en FlotUp cumple con las directrices de la NIST SP 800-63B, las cuales indican que las contraseñas deben almacenarse mediante funciones de derivación de clave unidireccional + salt, resistentes a ataques offline. [11]. Además, coincide con las recomendaciones del OWASP Password Storage Cheat Sheet, que sugiere el uso de algoritmos memory-hard como Argon2id para maximizar la seguridad frente a ataques por fuerza bruta con hardware especializado. [12]

La implementación satisface requisitos de seguridad aceptados internacionalmente, reduciendo significativamente la probabilidad de que se comprometan las credenciales en caso de filtración de los datos.

4.1.3 Impactos

La adopción de Argon2id como algoritmo principal de hashing genera un impacto directo y significativo en la seguridad del manejo de credenciales dentro de FlotUp. Al tratarse de un algoritmo memory-hard, incrementa de manera considerable el costo computacional asociado a ataques offline, reduciendo la viabilidad de intentos de fuerza bruta basados en hardware.

La aplicación combinada de salt por usuario y pepper aplicacional fortalece aún más la protección frente a filtraciones, mitigando riesgos asociados a tablas precomputadas y evitando que un atacante pueda verificar contraseñas sin conocer el valor secreto externo al sistema.

Adicionalmente, los parámetros seleccionados permiten mantener una experiencia de uso fluida, ya que los tiempos de hashing permanecen dentro de rangos aceptables para un entorno web sin comprometer el rendimiento general del sistema. Esto permite que la plataforma mantenga un equilibrio adecuado entre seguridad y usabilidad, requisito fundamental para su futura operación en diferentes entornos institucionales.

En conjunto, estas medidas elevan de forma sustancial el nivel de protección de las credenciales y contribuyen a establecer una base robusta para el resto de los controles de seguridad de la plataforma.

4.2 Objetivo 2 — Implementación del módulo de autenticación con tokens

4.2.1 Desarrollo

La autenticación

La autenticación es un componente esencial en cualquier plataforma multiusuario, ya que permite verificar la identidad de los usuarios y restringir el acceso a funcionalidades críticas. En sistemas SaaS como FlotUp, donde múltiples actores interactúan desde distintos dispositivos (administradores, funcionarios y conductores), es necesario combinar mecanismos de autenticación que sean seguros y, al mismo tiempo, adecuados al patrón de uso de cada tipo de cliente (web y móvil).

Tipos de autenticación existentes en la actualidad

Las principales opciones de autenticación para aplicaciones web modernas son:

1. Sesiones tradicionales (cookies + sesión en servidor): El servidor mantiene un identificador de sesión asociado al usuario autenticado, mientras que el navegador conserva una cookie que permite recuperar dicha sesión en cada solicitud.
2. API Keys estáticas: El servidor genera un valor único que el cliente envía en cada petición para identificarse y acceder a la API.
3. OAuth2 / OpenID Connect: Protocolos de autorización y federación de identidad en los que un proveedor externo autentica al usuario y emite tokens para acceder a recursos protegidos.
4. JSON Web Tokens (JWT): Mecanismo de autenticación stateless (sin estado) donde, tras verificar las credenciales, el servidor genera un token firmado que contiene información del usuario y que el cliente envía en cada solicitud posterior.

MECANISMO DE AUTENTICACIÓN	VENTAJAS	DESVENTAJAS
Sesiones tradicionales (cookies + sesión en servidor)	Sencillas de implementar, integradas en Django, seguras si se usan correctamente (cookies httpOnly).	Requieren almacenamiento de estado, dificultan el escalado horizontal, no funcionan bien en contenedores o serverless, complejas de usar con WebSockets.
API Keys estáticas	Muy fáciles de implementar, bajo costo de computación, ampliamente soportadas.	No representan a un usuario real, no expiran de forma natural, alto riesgo si se filtran, permisos poco granulares.
OAuth2 / OpenID Connect	Estándar moderno, federación con Google/Azure AD, soporte avanzado de identidades.	Excesiva complejidad para un SaaS interno, requiere un IdP o proveedor de identidad externo (como Google Identity Platform o Microsoft Azure Active Directory), sobrecarga innecesaria para el caso de uso.
JSON Web Tokens (JWT)	Stateless, funciona bien en APIs REST, ideal para clientes móviles, permite incluir claims (rol, ID), escalable y ligero.	Requiere protección adecuada de refresh tokens y definición cuidadosa de tiempos de expiración.

Tabla 4-2. Comparativa de mecanismos de autenticación
Fuente: Elaboración propia

Decisión

En FlotUp se adopta un esquema híbrido de autenticación, alineado con las necesidades de cada tipo de cliente:

- La interfaz web utilizada por administradores y funcionarios mantiene el modelo de sesiones tradicionales de Django, aprovechando el middleware existente, la protección CSRF y los flujos ya implementados sin requerir cambios en formularios ni plantillas.

- La aplicación móvil de conductores, en cambio, consume exclusivamente la API y opera en entornos potencialmente menos confiables, por lo que requiere un mecanismo de autenticación sin estado, ligero y fácil de renovar. Para este caso se

selecciona JWT con expiración corta, refresh tokens y soporte de blacklist, asegurando que el 100 % de los endpoints críticos consumidos por este cliente solo puedan ser accedidos por usuarios autenticados.

JWT permite la asignación de diferentes parámetros para ajustar la estrategia de seguridad. A continuación, se describen los parámetros utilizados y sus justificaciones para la infraestructura de FlotUp.

1. **ACCESS_TOKEN_LIFETIME**

   Define la duración del access token, es decir, el tiempo durante el cual el token es válido para autenticar solicitudes.
   Se establece en 15 minutos, lo cual reduce el impacto de una eventual filtración, ya que un token comprometido solo podría ser usado por un tiempo muy limitado

2. **REFRESH_TOKEN_LIFETIME**

   Indica cuánto tiempo permanece válido el refresh token, que permite obtener nuevos access tokens sin volver a introducir credenciales.
   Se establece en 7 días, esto significa que el conductor puede mantener una sesión activa durante 7 días, siempre que el refresh token no haya sido revocado. Es un tiempo adecuado para aplicaciones móviles que requieren sesiones prolongadas sin relogin frecuente.

3. **ROTATE_REFRESH_TOKENS**

   Controla si, al solicitar un nuevo access token, se debe generar también un nuevo refresh token.
   Se configura en "True" (opción habilitada), lo que implica que cada vez que el cliente renueva el token, se reemplaza el refresh token anterior por uno nuevo. Esto añade seguridad, ya que un token antiguo no puede seguir siendo reutilizado.

4. **BLACKLIST_AFTER_ROTATION**

   Determina si el refresh token anterior debe marcarse como inválido después de la rotación.
   Se configura en "True" (opción habilitada), por lo que, el refresh token antiguo se revoca automáticamente al generar uno nuevo. De esta forma, si un atacante obtuviera un token previamente emitido, no podría usarlo para extender la sesión del usuario, mitigando ataques de "token replay".

5. **AUTH_HEADER_TYPES**

   Especifica el tipo de valor esperado en la cabecera "HTTP Authorization".
   Lo configuramos en "Bearer", esto indica que el token debe enviarse en la forma estandarizada: "Authorization: Bearer <access_token>", esto garantiza compatibilidad con prácticas comunes en APIs REST y evita ambigüedades en la interpretación del encabezado.

Implementación

1. Configuración de autenticación y JWT (en settings de la aplicación Django)
   Se habilitan simultáneamente autenticación por sesión y JWT, y se exige autenticación en DRF por defecto.

```
# ================================================================
# Configuración de Django REST Framework y JWT
# ================================================================
REST_FRAMEWORK = {
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.SessionAuthentication",
        "rest_framework_simplejwt.authentication.JWTAuthentication",
    ),
    "DEFAULT_PERMISSION_CLASSES":
("rest_framework.permissions.IsAuthenticated",),
}

SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=15),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=7),
    "ROTATE_REFRESH_TOKENS": True,
    "BLACKLIST_AFTER_ROTATION": True,
    "AUTH_HEADER_TYPES": ("Bearer",),
}
```

Figura 4-9. Parámetros de configuración para Django REST Framework y Simple JWT
Fuente: Elaboración propia

2. Serializer personalizado para emisión de tokens sólo a conductores (acepta como ID el username o email) se valida la contraseña, el rol DRIVER y emite el refresh y access.

```
class DriverTokenObtainPairSerializer(TokenObtainPairSerializer):
    def validate(self, attrs):
        identifier = (
            self.initial_data.get("identifier")
            or self.initial_data.get("username")
            or self.initial_data.get("email")
        )
        password = self.initial_data.get("password")
        # ...validaciones...
        role = get_user_role(user)
        if role != RoleChoices.DRIVER:
            raise exceptions.AuthenticationFailed(
                "Solo conductores pueden iniciar sesión en esta app."
```

```
            )
        refresh = self.get_token(user)
        return {
            "refresh": str(refresh),
            "access": str(refresh.access_token),
            "role": "driver",
            "user_id": user.id,
        }
```

Figura 4-10. Serializador personalizado para validación de roles y emisión de tokens
Fuente: Elaboración propia

3. Vista para obtener el par de tokens

```
class DriverTokenObtainPairView(TokenObtainPairView):
    serializer_class = DriverTokenObtainPairSerializer
```

Figura 4-11. Vista de API para la generación del par de tokens del conductor
Fuente: Elaboración propia

4. Protección de endpoint de información del conductor. Requiere autenticación (JWT). Para la app móvil, el access token se envía en Authorization: Bearer.

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
@authentication_classes([JWTAuthentication, SessionAuthentication,
BasicAuthentication])
def get_user_driver_info(request):
    # ...
    serializer = UserDriverSerializer(data)
    return Response(serializer.data)
```

Figura 4-12. Definición de vista protegida mediante autenticación JWT y de sesión
Fuente: Elaboración propia

5. Registro de URL del endpoint de tokens. para exponer la vista:

```
# --- RUTA DE API JWT ---
path("api/token/", TokenObtainPairView.as_view(),
name="token_obtain_pair"),
# Endpoint exclusivo para conductores (acepta username o email)
path("api/token/driver/", DriverTokenObtainPairView.as_view(),
name="driver_token_obtain_pair"),
```

```
path("api/token/refresh/", TokenRefreshView.as_view(),
name="token_refresh"),
```

Figura 4-13. Configuración de rutas para los puntos de acceso de tokens JWT
Fuente: Elaboración propia

4.2.2 Validación

La validación del módulo de autenticación mediante JWT se realizó a través de pruebas orientadas a confirmar que el sistema emitiera tokens válidos, respetara los tiempos de expiración configurados y restringiera el acceso a todos los endpoints críticos de la API móvil de FlotUp.

En primer lugar, se envió una solicitud al endpoint exclusivo para conductores "/api/token/driver/", entregando credenciales válidas. La respuesta incluyó correctamente el par de tokens (access y refresh), en el formato estándar Bearer, validando así la configuración de firma y emisión:

```
RESPONSE 200 OK
{
    "refresh":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV
4cCI6MTc2NDQ4NjA4NiwiaWF0IjoxNzYzODgxMjg2LCJqdGkiOiJkMzU5MjNjMTQ0OWQ0ZWQ4
ODAzMzhmNjVmNDJkNmE0YiIsInVzZXJfaWQiOjN9.eKIIHyBTNvFWmQdC40KcSyxb9dRRUKKq
7s2cxmMo5ws",
    "access":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXh
wIjoxNzYzODgyMTg2LCJpYXQiOjE3NjM4ODEyODYsImp0aSI6ImY4NjI5NWZjMzkxYjRmMjQ4
ZWFiNTU1MGQwYTM0YjgyIiwidXNlcl9pZCI6M30.OmUyEjNmw6MQU3iA8RsgfIj_dFL2S4RcX
Zzcz6KoXuQ",
    "role": "driver",
    "user_id": 3
}
```

Figura 4-14. Respuesta exitosa con emisión de par de tokens JWT para conductores
Fuente: Elaboración propia

Esto confirma que el sistema genera tokens firmados, con claims válidos y en el formato esperado (Authorization: Bearer <token>).

Posteriormente, se probó el mecanismo de renovación. A partir de un refresh token inicial, el sistema emitió un nuevo access token y un refresh token rotado, validando el funcionamiento de ROTATE_REFRESH_TOKENS y BLACKLIST_AFTER_ROTATION:

```
Nuevo access token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhw
IjoxNzYzODgwODQ5LCJpYXQiOjE3NjM4Nzk5NDksImp0aSI6IjU2ZDExMDg4ZmExNzQxYzhiZ
GQ0ODJiOGZkNTk0ZWNiIiwidXNlcl9pZCI6M30.ljz76F4eeMyHcWP6tHTgwShrJ-
I05wnz92IA1G6uIZk
```

```
Nuevo refresh token (rotado):
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4
cCI6MTc2NDQ4NDc0OSwiaWF0IjoxNzYzODc5OTQ5LCJqdGkiOiIyNGZlMWQxMzdkZjI0MjY0Y
WFlYjVhOTUxZDk4MTI2NSIsInVzZXJfaWQiOjN9.j_8zDKE-O07AZJgw-
VY3MSeHc3ttr7blX8Sf5Dyc2XY
```

Figura 4-15. Resultados de la validación del mecanismo de rotación y renovación de tokens
Fuente: Elaboración propia

Con el refresh token antiguo, se verificó que el servidor lo rechazara correctamente, evidenciando que fue incorporado a la lista negra (blacklist):

```
Respuesta esperada (fallo): 401
{"detail":"Token is blacklisted","code":"token_not_valid"}
```

Figura 4-16. Validación de denegación de acceso por token en lista negra
Fuente: Elaboración propia

Finalmente, se confirmó que los endpoints críticos protegidos requieren un token válido para acceder, las solicitudes sin token devolvieron:

```
401 Unauthorized - Authentication credentials were not provided.
```

Figura 4-17. Validación de rechazo de acceso para solicitudes no autenticadas
Fuente: Elaboración propia

Mientras que solicitudes con un access token válido permitieron el acceso normal al recurso protegido:

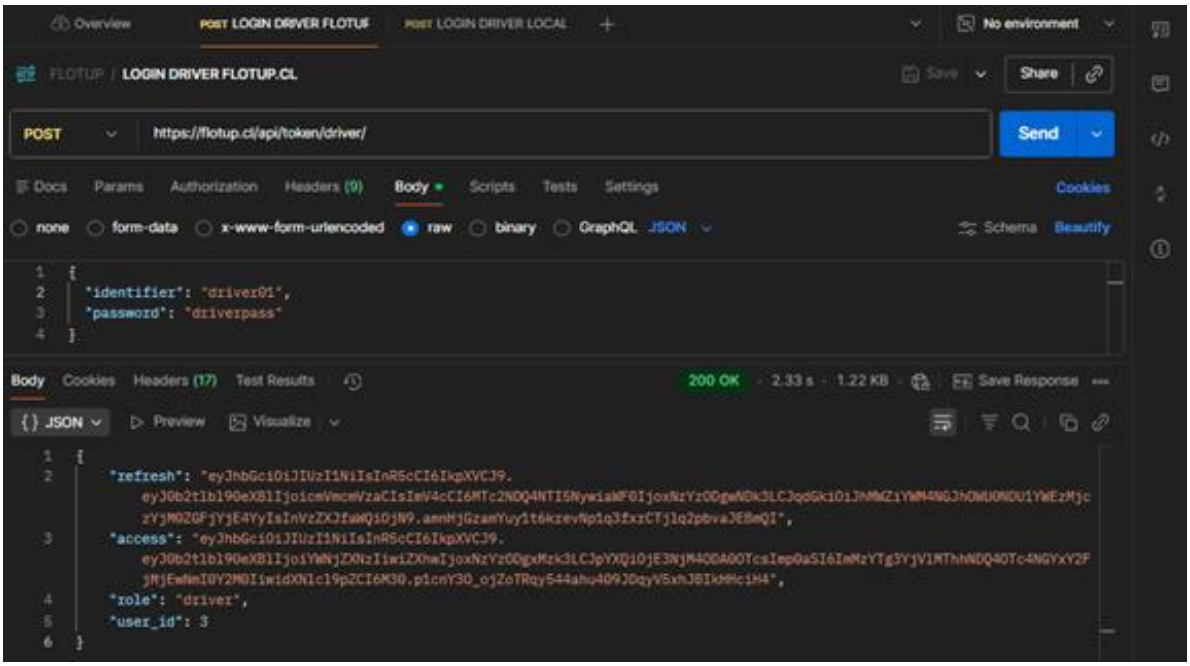

Figura 4-18. Validación de acceso autorizado a recursos protegidos
Fuente: Elaboración propia

Las pruebas demostraron que:

- El módulo de autenticación basado en tokens opera correctamente.
- Los tokens poseen expiración y renovación controlada.
- Los refresh tokens rotados se invalidan según lo definido.
- Los endpoints críticos consumidos por la aplicación móvil requieren autenticación válida.

Conformidad con estándares de seguridad:

La autenticación basada en JSON Web Tokens (JWT) implementada se fundamenta en estándares técnicos ampliamente reconocidos para tokens seguros. El estándar RFC 7519 define los JWT como un formato compacto y seguro para transmitir "claims" (datos sobre el usuario) entre partes, con firma criptográfica que protege la integridad de los tokens y reduce el riesgo de manipulación maliciosa [3]. Además, las directrices de identidad digital y autenticación de NIST SP 800-63B especifican que los mecanismos de autenticación deben garantizar la validez y protección de los tokens emitidos tras la autenticación, incluyendo prácticas de expiración y manejo de ciclo de vida que se aplican directamente al uso de access y refresh tokens en FlotUp. [11]

En conjunto, la implementación cumple con estas referencias internacionales, y alinea el proceso de emisión y validación de JWT con buenas prácticas de seguridad aceptadas, lo que confirma que no solo funciona técnicamente, sino que también satisface criterios de seguridad exigidos en sistemas modernos de autenticación.

4.2.3 Impactos

La incorporación de un mecanismo de autenticación basado en tokens aporta mejoras directas a la seguridad y escalabilidad de FlotUp. En primer lugar, el uso de JWT con expiración corta reduce significativamente el impacto de un eventual compromiso de credenciales, ya que un token filtrado sólo sería utilizable durante un periodo limitado. Asimismo, la rotación automática de refresh tokens y su registro en blacklist fortalece la protección frente a ataques de reutilización o secuestro de sesiones, un riesgo frecuente en aplicaciones móviles.

Desde el punto de vista operativo, el esquema stateless permite que la API funcione eficientemente en entornos desplegados sobre contenedores o servicios serverless, sin necesidad de mantener sesiones en memoria o sincronización entre instancias. Esto ayudará a garantizar un comportamiento consistente durante el futuro despliegue en Google Cloud Platform.

Finalmente, la verificación obligatoria de tokens en todos los endpoints críticos consumidos por los conductores asegura que sólo usuarios autenticados puedan acceder a información sensible, reforzando la trazabilidad, la integridad del sistema y la protección de los datos operativos de la flota.

4.3 Objetivo 3 — Sistema de rate-limiting para el inicio de sesión

4.3.1 Desarrollo

El endpoint de inicio de sesión es uno de los puntos más sensibles dentro de cualquier plataforma, pues constituye un vector común para ataques de fuerza bruta y enumeración de credenciales. Para mitigar estos riesgos se implementó un mecanismo de rate-limiting por dirección IP que limita la cantidad de solicitudes POST en un intervalo definido. Este control se aplica mediante un decorador sobre la vista de autenticación antes de procesar las credenciales, reduciendo la posibilidad de automatizar grandes volúmenes de intentos. El límite configurado es de 5 solicitudes por cada minuto por IP (5/m), alineado con buenas prácticas que recomiendan incorporar bloqueos temporales.

Implementación

1. Uso de la librería "django-ratelimit" en requirements.txt:

```
django-ratelimit==4.1.0    # Limitar intentos de login (anti fuerza bruta)
```

Figura 4-19. Declaración de dependencia para el control de tasa de solicitudes
Fuente: Elaboración propia

2. Decorador aplicado sobre la vista de login (plataforma web)

```
@ratelimit(key='ip', rate='5/m', method='POST', block=False)
def login_view(request):
    """
```

```
    Vista de inicio de sesión.
    """
    # Si ya esta logueado, redirige según su rol
```

Figura 4-20. Aplicación de decorador para control de tasa de peticiones en inicio de sesión
Fuente: Elaboración propia

Comportamiento:

- rate='5/m': máximo 5 peticiones POST por cada minuto por IP.
- block=False: la librería no bloquea automáticamente; expone request.limited = True cuando se supera el umbral.
- La vista detecta limited y devuelve mensaje de error sin ejecutar la autenticación.
- El conteo incluye cualquier POST (fallido o exitoso).

4.3.2 Validación

1. Se realizaron múltiples intentos fallidos consecutivos (POST) al endpoint de login desde la misma IP (localhost / proxy interno 172.21.0.1).
2. Se repitió la prueba desde cliente externo (móvil vía túnel Cloudflare) verificando captura de IP pública (IPv6 y luego IPv4).
3. Se creó un archivo de log "login_attempts.log" (por el logger auth.login) para la visualización de los intentos de inicio de sesión (exitosos y fallidos)

   La línea que menciona 'RATE_LIMIT' confirma que tras exceder el umbral configurado (rate='5/m') la vista detectó request.limited y bloqueó el procesamiento normal.

```
2025-11-23 14:30:29,154 INFO LOGIN_FAILURE ip=172.21.0.1 user_input='a'
2025-11-23 14:30:39,685 INFO LOGIN_FAILURE ip=172.21.0.1 user_input='a'
2025-11-23 14:30:45,469 INFO LOGIN_FAILURE ip=172.21.0.1 user_input='a'
2025-11-23 14:30:48,946 INFO LOGIN_FAILURE ip=172.21.0.1 user_input='a'
2025-11-23 14:30:51,737 INFO LOGIN_FAILURE ip=172.21.0.1 user_input='a'
2025-11-23 14:30:56,203 INFO RATE_LIMIT ip=172.21.0.1 user_input='a'
```

Figura 4-21. Registros de auditoría evidenciando el bloqueo por Rate Limit
Fuente: Elaboración propia

Luego, al transcurrir el tiempo de bloqueo, se puede realizar nuevamente el intento de inicio de sesión.

```
2025-11-23 14:32:49,560 INFO LOGIN_FAILURE ip=172.21.0.1 user_input='as'
2025-11-23 14:33:16,660 INFO LOGIN_FAILURE ip=172.21.0.1 user_input='pe'
```

Figura 4-22. Registro de auditoría mostrando la liberación del bloqueo tras el tiempo de espera
Fuente: Elaboración propia

La implementación de rate-limiting para el inicio de sesión se alinea con las mejores prácticas de seguridad de aplicaciones web, que recomiendan limitar la frecuencia de solicitudes para prevenir ataques de fuerza bruta y abuso automatizado de credenciales. En particular, guías de seguridad de aplicaciones señalan que aplicar límites por dirección IP o usuario es una defensa efectiva contra ataques de enumeración de credenciales y explotación de autenticación. [13]

Adicionalmente, en el entorno legal de Chile, la Ley Marco de Ciberseguridad N° 21.663 establece un marco para "prevenir, detectar y responder ante amenazas digitales", lo que incluye adoptar controles técnicos que reduzcan el riesgo de incidentes de seguridad, como mecanismos que mitiguen intentos maliciosos repetidos sobre servicios de autenticación. [14]

El sistema de rate-limiting configurado responde a criterios de seguridad respaldados por estándares de la industria y marcos legales de ciberseguridad, lo que otorga un sustento técnico y normativo al control aplicado.

4.3.3 Impactos

La incorporación del mecanismo de rate-limiting en el endpoint de inicio de sesión genera un impacto directo en la reducción de riesgos asociados a ataques de fuerza bruta y automatización de intentos de autenticación. Al limitar a cinco solicitudes por minuto por dirección IP, se incrementa significativamente el costo temporal de un ataque, desincentivando la exploración sistemática de credenciales y la enumeración de usuarios. Además, el registro explícito de eventos de inicio de sesión permite detectar comportamientos anómalos y facilita la trazabilidad ante incidentes de seguridad.

En conjunto, la medida contribuye a fortalecer la superficie de defensa del sistema, mitigando uno de los vectores de ataque más comunes en aplicaciones web y asegurando que el flujo de autenticación se mantenga protegido frente a consultas repetitivas o automatizadas.

4.4 Objetivo 4 — Medidas contra ataques OWASP Top 10

4.4.1 Desarrollo

OWASP Top 10

La protección frente a vulnerabilidades comunes es un componente esencial en aplicaciones web modernas, especialmente en plataformas SaaS donde múltiples usuarios interactúan de forma simultánea y desde distintos dispositivos. Para estructurar la mitigación de riesgos en FlotUp, se adoptaron como referencia las categorías del OWASP Top 10, un estándar internacionalmente reconocido que reúne las vulnerabilidades más críticas y frecuentes en aplicaciones web.

La elección de OWASP Top 10 responde a tres razones principales.

En primer lugar, constituye un marco ampliamente utilizado en auditorías, evaluaciones de madurez y programas de cumplimiento, lo que asegura que las medidas implementadas estén alineadas con prácticas aceptadas por la industria.

En segundo lugar, su enfoque es práctico y priorizado, permitiendo abordar vulnerabilidades transversales de alto impacto sin ampliar el alcance de la solución hacia catálogos más extensos como CWE o CVE (donde se documentan cientos de vulnerabilidades).

Finalmente, varias de sus categorías están directamente relacionadas con el stack actual de FlotUp, por lo que su adopción permite concentrarse en medidas aplicables a la realidad del proyecto.

ID	NOMBRE	DESCRIPCIÓN
A01	Broken Access Control	Fallas en los mecanismos de control de acceso que permiten a un usuario realizar acciones sin los permisos necesarios.
A02	Cryptographic Failures	Uso incorrecto de criptografía o exposición de datos sensibles en tránsito o en reposo.
A03	Injection	Inclusión de datos maliciosos en consultas SQL, comandos u otros intérpretes.
A04	Insecure Design	Debilidades estructurales en arquitecturas, flujos y decisiones de diseño que generan riesgos inherentes.
A05	Security Misconfiguration	Configuraciones inseguras en servidores, frameworks, contenedores o dependencias.
A06	Vulnerable and Outdated Components	Uso de dependencias, librerías o paquetes con vulnerabilidades conocidas o sin mantenimiento.
A07	Identification and Authentication Failures	Fallas en autenticación, gestión de sesiones o manejo de credenciales que permiten acceso no autorizado.
A08	Software and Data Integrity Failures	Procesos vulnerables que afectan la integridad de código, pipelines de despliegue o datos críticos.
A09	Security Logging and Monitoring Failures	Falta de registro, monitoreo o alertas oportunas ante comportamientos sospechosos o incidentes.
A10	Server-Side Request Forgery (SSRF)	Vulnerabilidad que permite a un atacante forzar al servidor a realizar solicitudes a recursos internos o externos.

Tabla 4-3. Categorías de riesgo del OWASP Top 10
Fuente: OWASP 2021 [5]

Implementación

Evidencia directa de implementación por categoría de riesgo

1. A01 - Broken Access Control

   Para esta categoría se incluyen diferentes protecciones:

   - Decoradores y permisos restringen las vistas (ejemplo admin_required).

```
from .utils import admin_required
@admin_required
def accounts_manager(request: HttpRequest) -> HttpResponse:
    return render(request, 'accounts/manager.html', data)
```

Figura 4-23. Restricción de acceso mediante decorador personalizado para roles administrativos
Fuente: Elaboración propia

   - Endpoints JWT protegidos para el conductor (con validación de rol).

```
path("api/token/driver/", DriverTokenObtainPairView.as_view(),
name="driver_token_obtain_pair"),

def is_driver(user) -> bool:
    role = get_user_role(user)
    return role == RoleChoices.DRIVER
```

Figura 4-24. Configuración de endpoint con validación específica de rol para conductores
Fuente: Elaboración propia

   - Protecciones para CSRF (Cross-Site Request Forgery), tipo de ataque donde un sitio externo induce al navegador del usuario autenticado a enviar peticiones maliciosas a la aplicación (aprovechando su cookie de sesión). Esto se mitiga exigiendo un token único (csrfmiddlewaretoken) en formularios o requests mutantes (POST, PUT, DELETE, etc.).

     La implementación es:

     - Middleware activo: django.middleware.csrf.CsrfViewMiddleware en MIDDLEWARE intercepta las solicitudes y valida el token.
     - CSRF_TRUSTED_ORIGINS en las variables de entorno y settings de Django definen los dominios desde los que se aceptan peticiones que incluyen encabezados seguros.

     - Cuando DEBUG=False se fuerza CSRF_COOKIE_SECURE=True (solo por HTTPS), aumentando protección en producción.

- CORS incluye el header x-csrftoken en CORS_ALLOW_HEADERS para permitir envíos AJAX desde orígenes permitidos.

```
MIDDLEWARE = [
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    # ...
]
CSRF_TRUSTED_ORIGINS = env.list("CSRF_TRUSTED_ORIGINS")
```

Figura 4-25. Configuración de middleware y orígenes confiables para protección CSRF
Fuente: Elaboración propia

Con esto, solo los formularios/clientes legítimos que incluyen el token CSRF emitido por el servidor pueden ejecutar acciones que cambian estado, las solicitudes forzadas desde terceros sin ese token son rechazadas.

2. A02 - Cryptographic Failures

   - Uso de Argon2id (hashing robusto) referenciado en ítem 4.1 de este informe.

   - Uso de JWT con expiraciones y rotación (mitiga exposición prolongada) referenciado en ítem 4.2 de este informe.

3. A03 - Injection

   Para abordar esta categoría, se utilizan las capacidades nativas de Django, el cual incorpora un ORM (Object-Relational Mapper) que construye consultas SQL parametrizadas, evitando la necesidad de concatenar cadenas manualmente y reduciendo el riesgo de inyección de forma significativa.

   Todas las operaciones de lectura y búsqueda sobre usuarios, vehículos u otros modelos se realizan mediante métodos del ORM, lo que asegura que los valores provistos por el usuario sean tratados como parámetros y no como parte de la instrucción SQL.

   Un ejemplo concreto es la búsqueda de usuario por identificador durante el proceso de autenticación, donde el filtro se realiza mediante una consulta indirecta:

```
user_obj = User.objects.get(email__iexact=identificador)
```

Figura 4-26. Consulta parametrizada mediante el ORM de Django para mitigar inyecciones
Fuente: Elaboración propia

En esta instrucción, el ORM traduce internamente el filtro email__iexact en una consulta SQL segura, aplicando parameter binding y evitando que el contenido del campo identificador pueda alterar la estructura de la consulta.

En FlotUp no existe construcción manual de SQL ni concatenación de strings para formar sentencias dinámicas. Todas las consultas se gestionan mediante las abstracciones de Django. Siendo una medida sólida contra inyecciones SQL en un sistema que opera sobre PostgreSQL.

Esta práctica se refuerza con la validación y sanitización de entradas implementada previamente, lo que contribuye a que los datos recibidos desde formularios o APIs sean filtrados antes de ser procesados por el backend.

4. A04 - Insecure Design

   Para esta categoría se incluyen diferentes consideraciones:

   - Limitación de intentos de login (defensa fuerza bruta) refleja decisión de diseño seguro (ítem 4.3 de este informe)

   - El rol de conductor cuenta con endpoint exclusivo (por lo que se reduce la superficie de riesgo).

5. A05 - Security Misconfiguration

   Para esta categoría se incluyen diferentes consideraciones:

   - Autenticación y permisos por defecto en DRF (todos requieren autenticación).

```
REST_FRAMEWORK = {
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.SessionAuthentication",
        "rest_framework_simplejwt.authentication.JWTAuthentication",
    ),
    "DEFAULT_PERMISSION_CLASSES":
("rest_framework.permissions.IsAuthenticated",),
}
```

Figura 4-27. Definición de políticas de autenticación y permisos globales en DRF
Fuente: Elaboración propia

   - Registro de intentos de inicio de sesión en archivo de logging configurado en ítem 4.3 de este documento.

6. A06 - Vulnerable and Outdated Components

   Para el control del entorno y componentes obsoletos se fijan las versiones de todas las dependencias en el archivo "requirements" del proyecto, con el fin de evitar que actualizaciones automáticas introduzcan vulnerabilidades, cambios incompatibles o diferencias entre entornos de ejecución y así mantener un sistema consistente y seguro.

```
…
# --- Seguridad ---
argon2-cffi==23.1.0             # Hashing robusto para contraseñas
django-ratelimit==4.1.0         # Limitar intentos de login (anti fuerza
bruta)
django-cors-headers==4.4.0      # Control de orígenes permitidos (CORS)

# --- Base de datos ---
…
```

Figura 4-28. Control de versiones fijas para mitigar riesgos por componentes obsoletos
Fuente: Elaboración propia

7. A07 - Identification and Authentication Failures

   Para esta categoría se incluyen diferentes protecciones:

   - Login tradicional con 'authenticate' + 'rate-limit' (visto en ítem 4.3 de este documento).

```
user = authenticate(request, username=identificador, password=password)
```

Figura 4-29. Uso de función nativa para validación de identidad y credenciales
Fuente: Elaboración propia

   - JWT emisión y refresh controlado, con rotación + blacklist (referenciado en ítem 4.2 de este documento).

   - Validación de rol antes de emitir tokens de conductor.

```
role = get_user_role(user)
if role != RoleChoices.DRIVER:
    raise exceptions.AuthenticationFailed(
        "Solo conductores pueden iniciar sesión en esta app."
    )
```

Figura 4-30. Control lógico de autorización basado en roles de usuario
Fuente: Elaboración propia

8. A08 - Software and Data Integrity Failures

   Rotación y blacklist de refresh tokens, evitando la reutilización maliciosa de estos (apartado documentado en ítem 4.2 de este documento).

9. A09 - Security Logging and Monitoring Failures

   Se genera un registro de intentos de inicio de sesión y eventos de rate-limit facilitando la auditoría.

```
2025-11-23 14:30:56,203 INFO RATE_LIMIT ip=172.21.0.1 user_input='a'
```

Figura 4-31. Trazabilidad de eventos de seguridad para monitoreo de intentos de acceso
Fuente: Elaboración propia

10. A10 - Server-Side Request Forgery (SSRF)

    El servidor de FlotUp nunca hace peticiones HTTP externas basadas en parámetros de usuario, por lo que hoy es imposible que ocurra un ataque SSRF.

    El backend actualmente no usa requests(), ni httpx, ni urllib, tampoco toma parámetros del usuario para enviarlos a un servicio externo (otros servidores), por lo que, un atacante no tiene ninguna forma de obligar al servidor a conectarse a otros servicios o direcciones internas, esto significa que no existe la vulnerabilidad SSRF en el estado actual del sistema.

4.4.2 Validación

La validación asociada a este objetivo se realizó de manera documental y técnica.

No obstante, las medidas implementadas fueron verificadas de manera indirecta a través de los mecanismos ya validados en otros apartados del documento:

- La mitigación frente a inyección (A03) se comprobó mediante el uso exclusivo del ORM de Django, ausencia de SQL dinámico y validación previa de entradas.

- La protección CSRF (A01/A07) se verificó revisando la presencia del middleware correspondiente, la inclusión del token en formularios y la configuración de dominios de confianza.

- Los controles de Identification and Authentication Failures (A07) fueron validados en la implementación del hashing y el módulo de autenticación JWT (secciones 4.1 y 4.2).

- La protección frente a Security Misconfiguration (A05) se confirmó revisando la configuración segura del framework, permisos por defecto en DRF, encabezados permitidos y parámetros deshabilitados en producción.

- Los registros asociados a Security Logging and Monitoring Failures (A09) fueron verificados mediante el archivo de auditoría generado para intentos de login y eventos de rate-limit (sección 4.3).

- La categoría SSRF (A10) se considera mitigada por diseño, ya que el servidor no ejecuta peticiones HTTP externas basadas en parámetros controlados por el usuario.

Estas protecciones están ampliamente reconocidas como prácticas de referencia para asegurar aplicaciones web modernas. El OWASP Top 10 es un documento de concienciación en seguridad respaldado por una comunidad global de expertos, que identifica los riesgos de seguridad más críticos a los que se enfrentan aplicaciones web

y móviles y orienta a los desarrolladores sobre cómo mitigarlos. El uso de este estándar como base para diseñar y evaluar las protecciones implementadas en FlotUp permite demostrar que las medidas de seguridad están alineadas con buenas prácticas internacionales ampliamente aceptadas en la industria. Este enfoque coincide con guías complementarias como el OWASP Application Security Verification Standard (ASVS), que proporciona criterios de verificación para evaluar los controles técnicos de seguridad en aplicaciones web, reforzando que la mitigación de estas vulnerabilidades cumple con parámetros estructurados para la evaluación técnica de seguridad. [15]

4.4.3 Impactos

La incorporación de medidas alineadas al OWASP Top 10 fortalece de manera significativa la postura de seguridad de FlotUp, ya que permite abordar las vulnerabilidades más frecuentes y de mayor severidad en aplicaciones web modernas.

El uso de protecciones como CSRF, sanitización de entradas, validación de datos y el ORM seguro, reduce de forma efectiva la probabilidad de ataques comunes como inyección, XSS y usos indebidos de sesiones. Asimismo, la definición explícita de controles de acceso, configuraciones seguras y políticas consistentes entre los distintos componentes del framework disminuye el riesgo de fallas derivadas de configuraciones incorrectas.

Adicionalmente, el reforzamiento del registro de eventos y el manejo adecuado de autenticación contribuyen a mejorar la trazabilidad y la capacidad de detección temprana de comportamientos anómalos. En conjunto, estas medidas permiten que la plataforma avance hacia un nivel de seguridad sólido y coherente con su operación como solución SaaS destinada a cualquier tipo de institución (sea pública o privada).

4.5 Objetivo 5 — Control de acceso basado en roles

4.5.1 Desarrollo

Control de acceso

El control de acceso es un componente fundamental en cualquier sistema multiusuario, ya que determina qué operaciones puede realizar cada actor dentro de la plataforma. En el contexto de FlotUp, donde conviven distintos perfiles operativos (funcionarios, conductores y administradores de flota) resulta necesario aplicar un modelo que permita definir permisos de forma clara, consistente y alineada a la estructura organizacional de cualquier institución que adopte la plataforma.

Tipos

Existen principalmente dos modelos utilizados en la industria para la gestión de permisos:

- RBAC (Role-Based Access Control): Las autorizaciones se determinan según el rol asignado al usuario

- ABAC (Attribute-Based Access Control): Las decisiones de acceso se basan en atributos dinámicos como ubicación, horario, estado del recurso, nivel de riesgo o cualquier metadato relevante.

Aunque ABAC ofrece una mayor granularidad, su implementación y mantenimiento son más complejos, especialmente en sistemas con requerimientos administrativos tradicionales.

CRITERIO	RBAC	ABAC
Principio base	Los permisos se asignan a roles predefinidos.	Las decisiones se basan en atributos del usuario, del recurso o del contexto.
Modelo de decisión	"El usuario de rol X puede hacer Y".	"Si usuario cumple atributo A, el recurso cumple atributo B y el contexto cumple atributo C, se permite/denega acceso".
Complejidad	Baja, fácil de implementar y mantener.	Alta, requiere políticas detalladas y motor de evaluación de reglas.
Escalabilidad de permisos	Alta, se puede ampliar a medida que los roles aumenten.	Alta, se puede ampliar a medida que las variables consideradas aumenten.
Flexibilidad	Deficiente, limitada a roles estáticos.	Alta, permite decisiones granulares y contextuales.
Facilidad de administración	Simple, se gestionan roles en lugar de permisos individuales.	Compleja, múltiples atributos deben mantenerse actualizados.
Casos de uso ideales	ideal para soluciones jerárquicas donde se cuenten con roles predefinidos.	Soluciones con condiciones cambiantes (IoT, sistemas bancarios, control basado en contexto).
Sobrecarga operacional	Baja, reglas simples y predecibles.	Alta, requiere evaluación continua de atributos.
Riesgos si se implementa mal	Delegación excesiva o roles demasiado amplios.	Regla mal definida puede permitir acceso no deseado o bloquear usuarios legítimos.
Ejemplos de atributos usados	Rol del usuario.	Rol, ubicación, horario, dispositivo, clasificación del recurso, estado del proceso, etc.

Tabla 4-4. Análisis comparativo entre los modelos de control de acceso
Fuente: Elaboración propia

Decisión

En FlotUp se adopta RBAC, ya que se adapta directamente a la estructura jerárquica de la solución:

- Administrador: supervisa la flota, accede a vistas de gestión, aprueba solicitudes de viaje (en caso de organizaciones con la política de "tipo de asignación de conductor" de forma manual).
- Funcionario: crea solicitudes y revisa sus viajes asignados.
- Conductor: lleva a cabo los viajes, operando únicamente con la información relacionada con su viaje activo.

El modelo RBAC permite definir reglas estáticas y explícitas, facilita la auditoría y reduce la superficie de error al evitar la proliferación de reglas basadas en atributos complejos.

Implementación

La implementación del control de acceso se realiza en dos niveles:

1. Nivel: Enumeración de roles usados en la base de datos y lógica de permisos.

```
class RoleChoices(models.IntegerChoices):
    COLLABORATOR = 1, "Funcionario"
    DRIVER       = 2, "Conductor"
    ADMIN        = 3, "Administrador"

def get_user_role(user):
    profile = getattr(user, "profile", None)
    return getattr(profile, "role", None)
```

Figura 4-32. Definición de roles y lógica de obtención de perfil de usuario
Fuente: Elaboración propia

2. Nivel: Permisos para API (DRF). Clases de permiso por rol, permiten acceso solo si el usuario está autenticado y su perfil coincide.

```
class IsAdmin(_RolePermission):
    required_role = _get_rolechoices().ADMIN

class IsCollaborator(_RolePermission):
    required_role = _get_rolechoices().COLLABORATOR

class IsDriver(_RolePermission):
    required_role = _get_rolechoices().DRIVER
```

Figura 4-33. Clases de permisos personalizadas para el control de acceso basado en roles (RBAC)
Fuente: Elaboración propia

Todas las vistas DRF (Django REST Framework) exigen autenticación, luego se añade permiso de rol según necesidad:

```
REST_FRAMEWORK = {
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.SessionAuthentication",
        "rest_framework_simplejwt.authentication.JWTAuthentication",
    ),
    "DEFAULT_PERMISSION_CLASSES":
("rest_framework.permissions.IsAuthenticated",),
}
```

Figura 4-34. Configuración de políticas de autenticación y permisos predeterminados en DRF
Fuente: Elaboración propia

Decoradores para vistas web (RBAC en capa de interfaz), se realiza la verificación del rol antes de renderizar la vista.

- Ejemplo: restricción para vista de administración de usuarios de la organización (solo disponible para administrador):

```
@permission_classes([IsAdmin])
def accounts_manager(request):
```

Figura 4-35. Implementación de decorador de permisos para la gestión de cuentas administrativas
Fuente: Elaboración propia

- Ejemplo: restricción en endpoint para la construcción del perfil del conductor (solo disponible para conductores)

```
@api_view(['GET'])
@permission_classes([IsDriver])
@authentication_classes([JWTAuthentication, SessionAuthentication,
BasicAuthentication])
def get_user_driver_info(request):
```

Figura 4-36. Implementación de decorador de permisos para la gestión de cuentas de conductores
Fuente: Elaboración propia

- Ejemplo: restricción para vista de crear una solicitud de viaje (solo disponible para funcionarios)

```
@require_http_methods(["GET", "POST"])
@permission_classes([IsCollaborator])
def create_travel_request(request):
    """
    Vista para crear una solicitud de viaje.
    """
```

Figura 4-37. Implementación de decorador de permisos para la gestión de cuentas de funcionarios
Fuente: Elaboración propia

4.5.2 Validación

La validación del esquema de control de acceso basado en roles se realizó comprobando que las vistas y endpoints protegidos respondan correctamente según el rol del usuario autenticado. Para ello, se generaron usuarios de prueba con los tres roles definidos en la plataforma (administrador, funcionario y conductor) y se efectuaron solicitudes a las rutas asociadas a cada perfil.

En primer lugar, se verificó que un usuario con rol "Administrador" pudiera acceder a vistas exclusivas de gestión, como el home de administrador (donde puede ver la flota en tiempo real, información general de viajes, etc.). La solicitud se realizó exitosamente, confirmando que el permiso "IsAdmin" permite acceso únicamente a los perfiles autorizados:

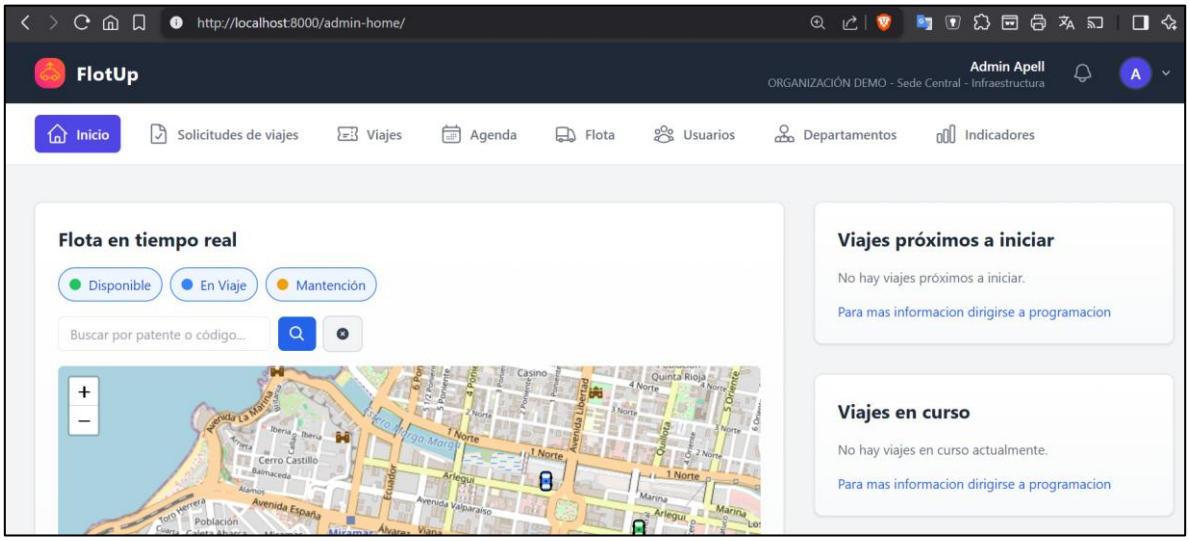

Figura 4-38. Interfaz del panel de administración accesible para usuarios con rol adecuado
Fuente: Elaboración propia

Posteriormente, se intentó acceder a la misma vista utilizando un usuario con un rol diferente (Funcionario). El sistema devolvió:

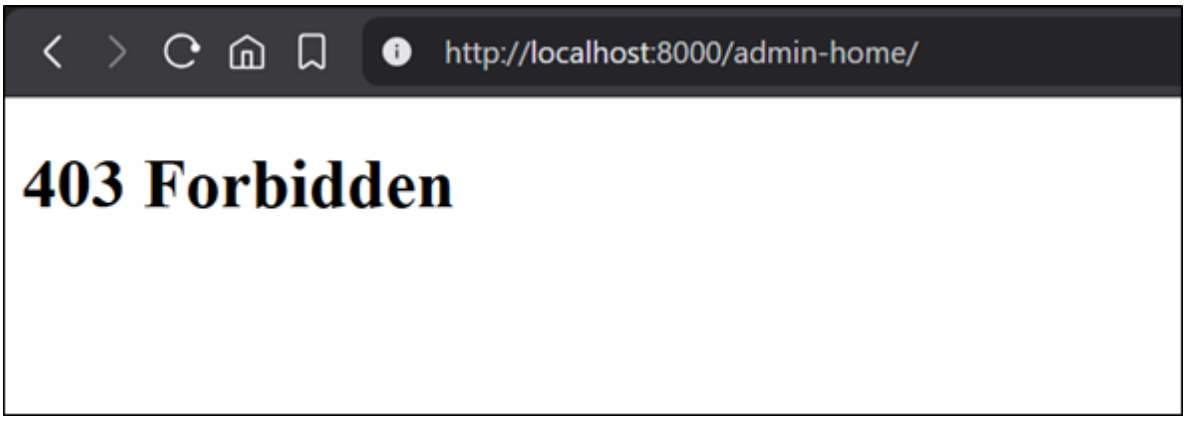

Figura 4-39. Interfaz del panel de administración inaccesible para usuarios con rol inadecuados
Fuente: Elaboración propia

lo que reafirma que las restricciones definidas en los decoradores y clases de permiso funcionan correctamente.

Según OWASP, RBAC permite que los permisos no se asignen directamente a cada usuario, sino que se asocien a roles predefinidos, y los usuarios heredan esos permisos mediante su rol, promoviendo el principio de mínimo privilegio y reduciendo la superficie de ataque [16]. Además, el modelo RBAC ha sido estandarizado y promovido por organizaciones como NIST, que reconoce su eficacia para controlar y limitar el acceso a sistemas y datos en entornos de seguridad TI, reforzando su uso como control de acceso efectivo en soluciones empresariales. [17]
En conjunto, estas pruebas confirman que el modelo RBAC implementado en FlotUp aplica de forma consistente las restricciones de acceso para cada rol, cumpliendo con el objetivo planteado y asegurando que cada actor del sistema interactúe únicamente con las funcionalidades que le corresponden.

4.5.3 Impactos

La incorporación de un modelo de control de acceso basado en roles permite que cada usuario de FlotUp interactúe únicamente con las funcionalidades asociadas a sus responsabilidades operativas, reduciendo de manera significativa el riesgo de accesos indebidos o manipulaciones no autorizadas dentro del sistema.

El uso de permisos explícitos y verificaciones consistentes tanto en la API como en las vistas web fortalece la trazabilidad y facilita auditorías internas, ya que las acciones quedan claramente asociadas al rol que las ejecuta. Además, la estructura jerárquica y estática del modelo RBAC simplifica la administración de permisos y evita errores comunes derivados de configuraciones manuales o reglas complejas.

En conjunto, este esquema mejora la seguridad, la gobernanza y la continuidad operativa de la plataforma, asegurando que FlotUp pueda adaptarse a instituciones con estructuras organizacionales diversas.

4.6 Objetivo 6 — Medidas perimetrales

El presente objetivo considera la planificación y definición técnica de medidas perimetrales para la protección del entorno de red. Aunque su implementación requiere un despliegue en infraestructura Cloud, se definen a continuación las buenas prácticas de implementación a futuro:

- Certificados SSL/TLS:
  Implementar protocolos TLS 1.2 o superior, desactivando versiones obsoletas (SSLv3, TLS 1.0) para evitar ataques de degradación. Se sugiere el uso de certificados emitidos por autoridades reconocidas (como Let's Encrypt o AWS ACM) con renovación automática para evitar la interrupción del servicio.

- Reglas WAF:
  Configurar un conjunto de reglas que mitiguen los riesgos del OWASP Top 10, específicamente contra Inyecciones SQL (SQLi) y Cross-Site Scripting (XSS). Se recomienda el uso de un firewall de aplicaciones en modo "bloqueo" para filtrar tráfico malicioso antes de que llegue al servidor de Django.

- Mecanismos de mitigación de DDoS:
  Utilizar redes de entrega de contenido (CDN) como Cloudflare o AWS CloudFront para absorber picos de tráfico anómalo. Se deben configurar límites de tasa (rate-limiting) a nivel de red y escudos de protección de capa 7 para garantizar la disponibilidad del servicio ante ataques de denegación de servicio.

Debido a que la plataforma FlotUp se encuentra actualmente en ejecución local y no dispone de la capa de infraestructura Cloud necesaria (servicios gestionados y puntos de entrada públicos), estas medidas no podrán ser validadas técnicamente en esta etapa del proyecto. No obstante, su incorporación conceptual asegura una hoja de ruta clara para el despliegue en producción.

4.7 Objetivo 7 — Pruebas de validación de seguridad

El objetivo de validar la efectividad de las medidas de seguridad implementadas se abordó de manera transversal a lo largo de los objetivos anteriores, donde se ejecutaron pruebas específicas para cada control aplicado:

- Hashing de credenciales con Argon2id
- Autenticación mediante tokens JWT con expiración y rotación controlada
- Limitación de intentos de inicio de sesión con rate-limiting
- Mitigación de vulnerabilidades del top 10 de OWASP
- Verificación del control de acceso (RBAC).

Comprobando que los mecanismos respondieran conforme a los principios de confidencialidad, integridad y disponibilidad definidos en el modelo CIA. En conjunto, los resultados obtenidos demuestran que las medidas aplicadas se ejecutan correctamente y que el sistema mantiene un comportamiento seguro frente a los escenarios evaluados, cumpliendo así el objetivo de validación establecido.

4.8 Análisis preliminar de costos y viabilidad

Si bien el presente trabajo se centra en el diseño e implementación de mecanismos de seguridad, resulta pertinente realizar una estimación preliminar de costos asociados al despliegue e infraestructura requerida.

Considerando su implementación en la nube (Google Cloud Platform) y una carga estimada de operación concurrente potencial de aprox. 60 vehículos y 10 usuarios administrativos, los principales costos corresponden a los siguientes servicios:

SERVICIO	DETALLE Y CARGA ESTIMADA	COSTO MENSUAL (USD)
Cloud Run (Backend Django + WebSockets)	App ASGI (Django + DRF + Channels). 6–12 req/s promedio. Autoescala con 1–2 instancias activas (1 vCPU / 2 GB).	250 – 350
Cloud SQL (PostgreSQL + PostGIS)	Base principal: escrituras GPS (batch cada 10 s) y lecturas de panel. 1 instancia db-custom-2-2048, 30 GB SSD.	80 – 150
Cloud SQL (DWH)	Cargas ETL periódicas (cada hora). Instancia pequeña (1 vCPU / 2 GB RAM).	40 – 70
Memorystore (Redis)	Cache + canal layer para websockets (≈ 300 conexiones simultáneas). Tier Basic 1 GB.	25 – 40
Cloud Storage + CDN	Archivos estáticos + media (~10 GB totales, 100 GB egress/mes).	10 – 30
Load Balancing (HTTP/S)	Balanceo global + SSL + soporte WebSockets.	20 – 30
Secret Manager	6–10 secretos gestionados (.env, DB, JWT, etc.).	2 – 5
Cloud Scheduler + Run Jobs	3–5 tareas cron diarias (ETL, migraciones, limpieza).	5 – 10
Cloud Build + Artifact Registry (CI/CD)	2–3 builds semanales.	5 – 10
Cloud Logging + Monitoring	Observabilidad y métricas básicas.	5 – 10
DNS / Dominio / Certificados	Cloud DNS + certificados SSL automáticos.	5 – 10
API Google Maps	Localizador de ubicaciones de origen, destino e intermedias	80 – 100
Costo mensual estimado		USD 527 – 825

Tabla 4-5. Estimación preliminar de costos de despliegue de infraestructura de FlotUp
Fuente: Elaboración propia

El costo mensual proyectado para una organización se sitúa en un rango aproximado de 500 a 800 USD mensuales, dependiendo de la configuración final y el consumo efectivo de recursos.

Este análisis preliminar permite concluir que la solución es técnicamente viable y económicamente abordable para una empresa o institución pública de tamaño medio, especialmente considerando los beneficios asociados a la trazabilidad, seguridad y control de flota que se adquieren con la solución.

5 Conclusiones

El desarrollo de las medidas de seguridad para la plataforma FlotUp permitió consolidar una base tecnológica robusta para un sistema SaaS que será utilizado en entornos institucionales, donde la protección de credenciales, datos sensibles y accesos es un requisito esencial. El trabajo realizado no solo fortaleció los mecanismos internos del sistema, sino que también evidenció la importancia de integrar prácticas de seguridad desde las primeras etapas del diseño, especialmente en soluciones destinadas a organizaciones públicas o privadas con requerimientos estrictos de trazabilidad y control.

Entre los aprendizajes más relevantes se encuentra la correcta selección de algoritmos criptográficos, destacando la evaluación comparativa entre PBKDF2, bcrypt y Argon2id, lo que permitió identificar el enfoque más adecuado frente a ataques. Asimismo, la implementación de autenticación basada en tokens con expiración y renovación controlada demostró ser una solución eficiente para clientes móviles, mientras que la integración de rate-limiting, medidas contra vulnerabilidades del OWASP Top 10 y un modelo RBAC claro y verificable reforzó la superficie de protección del sistema. Estas tareas contribuyeron al fortalecimiento de habilidades relacionadas con el análisis de riesgos, la configuración segura de frameworks, la validación práctica de controles y la documentación técnica alineada con estándares de la industria.

El proyecto permitió, además, evidenciar cómo decisiones arquitectónicas aparentemente simples (como la separación de roles, el uso del ORM para prevenir inyecciones, o el registro sistemático de eventos sensibles) tienen un impacto directo en la confianza, la integridad operativa y la capacidad de auditoría del sistema. En conjunto, la solución presentada cumple con los objetivos planteados y sienta una base sólida para la futura etapa de despliegue en infraestructura cloud, donde controles perimetrales como SSL/TLS, WAF y mitigación DDoS podrán integrarse para completar el modelo de defensa en capas.

Como trabajo futuro, se recomienda profundizar en estrategias de seguridad avanzadas, tales como autenticación multifactor (MFA), análisis automatizado de vulnerabilidades, integración con herramientas SIEM para monitoreo centralizado de los eventos de seguridad del sistema y la aplicación de hardening (endurecimiento de configuraciones para reducir riesgos) específico para entornos containerizados. Estas mejoras permitirían seguir elevando el nivel de protección de FlotUp y asegurar su sostenibilidad en escenarios de mayor escala operativa.

6 Bibliografía

[1] AWS (Amazon Web Services), «What is SaaS?,» 2025. [En línea]. Available: https://aws.amazon.com/es/what-is/saas/.

[2] Password Hashing Competition, «Argon2: The memory-hard function for password hashing and other applications,» 2015. [En línea]. Available: https://password-hashing.net/. [Último acceso: 2025].

[3] «RFC 7519: JSON Web Token (JWT),» 2015. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc7519. [Último acceso: 2025].

[4] V. C. Hu, D. F. Ferraiolo, D. R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller y K. Scarfone, «Guide to Attribute Based Access Control (ABAC) Definition and Considerations,» 2014. [En línea]. Available: https://doi.org/10.6028/NIST.SP.800-162. [Último acceso: Octubre 2025].

[5] Open Web Application Security Project (OWASP), «OWASP Top Ten: The Ten Most Critical Web Application Security Risks,» 2021. [En línea]. Available: https://owasp.org/Top10/es/. [Último acceso: Octubre 2025].

[6] Cloudflare, Inc., «What is a Web Application Firewall (WAF)? Cloudflare Learning Center,» 2024. [En línea]. Available: https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/. [Último acceso: Octubre 2025].

[7] Lyft, Inc., «How Lyft Protects Your Data. Lyft Safety Center,» 2024. [En línea]. Available: https://www.lyft.com/safety. [Último acceso: Octubre 2025].

[8] Uber Technologies, Inc., «Security and Privacy Practices at Uber. Uber Safety & Privacy,» 2024. [En línea]. Available: https://www.uber.com/us/en/safety/. [Último acceso: Octubre 2025].

[9] Geotab Inc., «Geotab Security Practices: Protecting Fleet Data. Geotab Resources,» 2024. [En línea]. Available: https://www.geotab.com/security/. [Último acceso: Octubre 2025].

[10] Samsara Inc., «Samsara Security Overview. Samsara Documentation,» 2024. [En línea]. Available: https://www.samsara.com/legal/security. [Último acceso: Octubre 2025].

[11] «NIST SP 800-63B: Digital Identity Guidelines — Authentication and Lifecycle Management,» 2025. [En línea]. Available: https://pages.nist.gov/800-63-4/sp800-63b.html. [Último acceso: 2025].

[12] «OWASP Password Storage Cheat Sheet,» 2025. [En línea]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html. [Último acceso: 2025].

[13] «Rate limiting best practices,» 2025. [En línea]. Available: https://developers.cloudflare.com/waf/rate-limiting-rules/best-practices/. [Último acceso: 2025].

[14] «Ley Marco de Ciberseguridad N° 21.663,» 2025. [En línea]. Available: https://preyproject.com/es/blog/ley-21663-marco-de-ciberseguridad-en-chile-2025/. [Último acceso: 2025].

[15] «OWASP Application Security Verification Standard (ASVS),» 2025. [En línea]. Available: https://owasp.org/www-project-application-security-verification-standard/. [Último acceso: 2025].

[16] «Authorization Cheat Sheet,» 2025. [En línea]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html. [Último acceso: 2025].

[17] «NIST Role-Based Access Control Standard Reference,» 2025. [En línea]. Available: https://csrc.nist.gov/projects/role-based-access-control. [Último acceso: 2025].

[18] Fortinet, «¿Qué es la tríada CIA?,» 2025. [En línea]. Available: https://www.fortinet.com/lat/resources/cyberglossary/cia-triad. [Último acceso: 5 Octubre 2025].