UrbanTracker: Sistema de geolocalización de flota de transporte urbano en tiempo real

Brayan Estiven Carvajal Padilla
Servicio Nacional de Aprendizaje SENA, Regional Huila
Neiva, Colombia
brayancarvajalpadilla02@gmail.com

Jésus Ariel González Bonilla
Servicio Nacional de Aprendizaje SENA, Regional Huila
Neiva, Colombia
ariel5253@hotmail.com

Resumen

UrbanTracker es una plataforma integral creada con el propósito de mejorar la movilidad urbana mediante el seguimiento en tiempo real de los vehículos que conforman el sistema de transporte público. La idea surge de la necesidad evidente de ofrecer información más precisa y actualizada sobre el paradero de los autobuses, ya que en la mayoría de ciudades los usuarios siguen dependiendo de estimaciones aproximadas o de suposiciones basadas en la experiencia. Al mostrar la ubicación exacta de cada vehículo, el sistema reduce considerablemente la incertidumbre de los tiempos de espera y permite que los pasajeros puedan planear mejor sus desplazamientos diarios. Paralelamente, la plataforma brinda a los administradores un mayor control operativo, puesto que pueden monitorear el comportamiento de la flota, detectar retrasos con rapidez y tomar decisiones que mejoren la eficacia del servicio.

El funcionamiento de UrbanTracker se basa en varios componentes que trabajan de forma coordinada. Uno de ellos es la aplicación móvil destinada a los conductores, desarrollada en React Native, que tiene como tarea principal obtener las coordenadas GPS del dispositivo y enviarlas de manera continua al servidor central. Esta aplicación se diseñó para que su uso fuera lo más sencillo posible: el conductor únicamente debe iniciar sesión, seleccionar la ruta asignada y continuar con su labor habitual, ya que el sistema se encarga automáticamente de transmitir la información necesaria sin distraerlo de la conducción.

En cuanto a la experiencia del usuario final y de los encargados de la operación, la plataforma web de UrbanTracker permite visualizar en un mapa interactivo las rutas disponibles y los vehículos que se encuentran actualmente activos. Este recurso facilita que las personas identifiquen rápidamente la posición de los autobuses y organicen sus trayectos con datos reales y no con aproximaciones. Para los administradores, la herramienta incluye opciones adicionales como crear o editar rutas, registrar nuevos vehículos, gestionar conductores y observar el estado operativo de cada unidad en tiempo real, todo en una misma interfaz.

Detrás de estas funcionalidades se encuentra un backend desarrollado en Java con el framework Spring Boot, donde reside la lógica de negocio que articula todos los módulos del sistema. La comunicación entre la aplicación móvil, el servidor y la plataforma web se realiza utilizando MQTT, un protocolo ampliamente empleado en soluciones de Internet de las Cosas debido a su capacidad para manejar transmisiones constantes de datos sin requerir grandes recursos de red. Esta elección tecnológica resulta especialmente útil en entornos de transporte urbano, donde la conectividad de los dispositivos móviles puede variar con frecuencia. Al utilizar MQTT, UrbanTracker logra mantener un flujo estable de información incluso en condiciones de red poco favorables.

Además de mostrar ubicaciones en tiempo real, el sistema incorpora mecanismos de autenticación, validación de rutas, almacenamiento estructurado y procesamiento continuo de información para garantizar que los datos sean confiables y estén siempre actualizados. A lo largo del trabajo se explican los requisitos que guiaron el diseño de cada componente, el marco conceptual relacionado con la geolocalización y los protocolos de comunicación utilizados, así como la metodología que se siguió durante la implementación.

Los resultados preliminares obtenidos en las pruebas muestran que la arquitectura elegida —basada en React Native, Spring Boot y MQTT— permite alcanzar tiempos de respuesta muy bajos, lo que se traduce en actualizaciones rápidas y estables. En la práctica, esto mejora tanto la experiencia de quienes usan el sistema para conocer la ubicación de los autobuses como la gestión que realizan los administradores del servicio. En conjunto, estos resultados confirman que UrbanTracker es una solución viable y efectiva para fortalecer la calidad del transporte público mediante tecnologías modernas y accesibles.

Keywords

geolocalización, transporte público, rastreo en tiempo real, MQTT, React Native, Spring Boot

1. Introducción

El transporte público urbano suele enfrentarse a problemas de ineficiencia operativa y a una notable falta de información precisa para los usuarios. En la mayoría de ciudades, los pasajeros desconocen la ubicación exacta de los autobuses en tiempo real, lo que provoca esperas largas e impredecibles, especialmente en horas de alta demanda. Esta incertidumbre no solo afecta la percepción del servicio, sino que también influye en la puntualidad, la comodidad y la planificación diaria de los usuarios, quienes deben ajustarse a horarios aproximados y confiar en estimaciones que rara vez reflejan las condiciones reales del sistema. En los últimos años, diversos estudios han demostrado que los sistemas de monitoreo de flota y seguimiento en tiempo real pueden reducir significativamente los tiempos de espera y mejorar la experiencia del usuario, ya que proporcionan una visión más clara de la operación de los vehículos. Además, investigaciones sobre soluciones de rastreo con datos precisos sugieren que contar con información exacta sobre la llegada de los autobuses aumenta la confianza de los pasajeros y fomenta el uso continuo del transporte público como alternativa sostenible.

Para contextualizar mejor esta problemática, es útil considerar algunas estadísticas globales. Según informes de la Organización Mundial de la Salud (OMS) y la Unión Internacional de Transporte Público (UITP), alrededor del 50 % de la población urbana en

países en desarrollo depende del transporte público para sus desplazamientos diarios. Sin embargo, en muchas ciudades latinoamericanas, como Bogotá o Ciudad de México, los usuarios reportan tiempos de espera promedio de 20 a 30 minutos, lo que no solo genera frustración, sino que también contribuye a la congestión vehicular al incentivar el uso de automóviles particulares. Esta situación se agrava en horas pico, donde la falta de información en tiempo real puede llevar a decisiones apresuradas, como caminar largas distancias o esperar en paradas incorrectas. Estudios realizados en Europa y Estados Unidos indican que la implementación de sistemas de rastreo GPS puede reducir estos tiempos de espera en un 30-40 %, además de mejorar la puntualidad general del servicio y reducir las emisiones de carbono al disminuir la circulación innecesaria de vehículos.

La evolución histórica de estas tecnologías también resulta reveladora. Desde los primeros sistemas de radiofrecuencia en los años 80, pasando por el auge de los GPS en los 2000, hasta las soluciones actuales basadas en IoT y big data, el rastreo vehicular ha avanzado considerablemente. Inicialmente, estos sistemas eran costosos y limitados a flotas corporativas, pero el abaratamiento de los dispositivos móviles y la proliferación de APIs abiertas han democratizado el acceso. Hoy en día, plataformas como Google Maps o Moovit ofrecen estimaciones básicas, pero carecen de la integración directa con operadores locales, lo que limita su precisión en contextos urbanos complejos. UrbanTracker se posiciona en esta línea evolutiva, aprovechando tecnologías maduras para ofrecer una solución accesible y adaptable a realidades locales.

Ante este panorama, surge la necesidad de implementar herramientas tecnológicas que permitan modernizar y dinamizar la gestión de los sistemas de transporte urbano. Es en este contexto donde aparece UrbanTracker, una solución integral de geolocalización diseñada para ofrecer información en tiempo real sobre la ubicación de los vehículos y apoyar la administración eficiente de la flota. El objetivo principal de UrbanTracker es permitir que los usuarios consulten fácilmente las rutas disponibles y visualicen, en un mapa interactivo, la posición actualizada de los autobuses que se encuentran en servicio. De esta forma, las personas pueden planear sus trayectos con mayor precisión y adaptarse a las condiciones reales del entorno.

Para lograr este funcionamiento, el sistema integra servicios de mapas, tecnologías GPS y protocolos de comunicación en tiempo real, como WebSockets o MQTT, que permiten la transmisión continua de las coordenadas de cada vehículo. Estos mecanismos garantizan que la información mostrada en la plataforma sea dinámica, confiable y lo suficientemente rápida como para reflejar de manera fiel el desplazamiento de los autobuses. Según la especificación de requisitos, UrbanTracker incluye funcionalidades clave que abarcan tanto a los usuarios como a los operadores: consulta de rutas, visualización de vehículos activos en el mapa, autenticación de conductores para iniciar recorridos, y un módulo administrativo que permite gestionar rutas, conductores y vehículos desde una interfaz centralizada y sencilla de utilizar.

Además de estos beneficios funcionales, UrbanTracker aporta valor en múltiples dimensiones. Para los usuarios finales, significa una reducción tangible en el estrés diario, permitiendo actividades como leer un libro mientras esperan o coordinar citas con mayor exactitud. Para los administradores, implica una herramienta de toma de decisiones basada en datos reales, facilitando la optimización de rutas, la asignación de recursos y la respuesta rápida a incidentes. En un nivel más amplio, contribuye a la sostenibilidad urbana al promover el uso del transporte público y reducir la dependencia de vehículos privados, lo que se alinea con objetivos globales como los de la Agenda 2030 de las Naciones Unidas para el desarrollo sostenible.

El presente documento profundiza en los componentes que conforman la plataforma, el marco teórico que respalda su diseño, la metodología de desarrollo empleada y los resultados obtenidos durante su implementación. Asimismo, se presentan las conclusiones generadas a partir del proceso de construcción de UrbanTracker y su relevancia como herramienta innovadora para mejorar la movilidad en entornos urbanos. La evolución tecnológica en el rastreo vehicular ha sido impulsada por avances en sensores, conectividad y procesamiento de datos. Desde los primeros sistemas basados en radiofrecuencia en los años 80, que requerían infraestructura dedicada y eran costosos, hemos pasado a soluciones móviles integradas en smartphones. La llegada de GPS preciso y redes celulares ha democratizado el acceso, permitiendo que incluso operadores pequeños implementen sistemas efectivos. Sin embargo, desafíos como la precisión en entornos urbanos densos y la dependencia de baterías móviles persisten, motivando innovaciones continuas en algoritmos de localización y optimización energética.

UrbanTracker se posiciona en esta trayectoria evolutiva, aprovechando tecnologías maduras como MQTT y Spring Boot para ofrecer una solución robusta y escalable. Su diseño modular permite adaptaciones a diferentes contextos, desde flotas escolares hasta servicios de entrega urbana, demostrando versatilidad en aplicaciones reales. La evolución tecnológica en el rastreo vehicular ha sido impulsada por avances en sensores, conectividad y procesamiento de datos. Desde los primeros sistemas basados en radiofrecuencia en los años 80, que requerían infraestructura dedicada y eran costosos, hemos pasado a soluciones móviles integradas en smartphones. La llegada de GPS preciso y redes celulares ha democratizado el acceso, permitiendo que incluso operadores pequeños implementen sistemas efectivos. Sin embargo, desafíos como la precisión en entornos urbanos densos y la dependencia de baterías móviles persisten, motivando innovaciones continuas en algoritmos de localización y optimización energética.

UrbanTracker se posiciona en esta trayectoria evolutiva, aprovechando tecnologías maduras como MQTT y Spring Boot para ofrecer una solución robusta y escalable. Su diseño modular permite adaptaciones a diferentes contextos, desde flotas escolares hasta servicios de entrega urbana, demostrando versatilidad en aplicaciones reales. La integración de mapas interactivos y notificaciones en tiempo real no solo mejora la experiencia del usuario, sino que también facilita la toma de decisiones operativas, reduciendo tiempos de respuesta y aumentando la eficiencia general del sistema.

Además, la plataforma incorpora medidas de seguridad avanzadas para proteger la privacidad de los datos, cumpliendo con regulaciones internacionales como GDPR y leyes locales de protección de datos. Esto asegura que la información de ubicación se maneje de manera ética y responsable, fomentando la confianza de los usuarios y operadores por igual. En resumen, UrbanTracker no solo resuelve un problema técnico, sino que contribuye a una

transformación más amplia en la movilidad urbana, promoviendo sistemas de transporte más eficientes, sostenibles y centrados en el usuario. Su implementación exitosa valida el potencial de las tecnologías modernas para mejorar la calidad de vida en entornos urbanos complejos.

Además, el sistema se adapta a contextos diversos, desde ciudades grandes con alta densidad de tráfico hasta municipios más pequeños con recursos limitados. La modularidad de UrbanTracker permite personalizaciones según las necesidades locales, como integración con sistemas de pago electrónico o alertas de emergencias. Esta flexibilidad asegura que el proyecto no sea una solución estática, sino un marco evolutivo que crece con las demandas de la sociedad moderna.

Finalmente, UrbanTracker representa un paso adelante en la democratización de la tecnología para el transporte público, haciendo que herramientas avanzadas estén al alcance de operadores de cualquier tamaño. Al combinar accesibilidad con innovación, el sistema sienta las bases para un futuro donde la movilidad urbana sea más inteligente, equitativa y respetuosa con el medio ambiente.

2. Marco teórico y trabajos relacionados

Los sistemas de rastreo en tiempo real combinan dispositivos capaces de obtener posiciones GPS con canales de comunicación diseñados para transmitir datos de manera casi instantánea, lo que permite mostrar la ubicación de vehículos en mapas, generar alertas o alimentar algoritmos de análisis. Este tipo de soluciones se ha convertido en una herramienta fundamental para mejorar la movilidad urbana, ya que permiten conocer con precisión el estado operativo de una flota y anticipar posibles retrasos. En el ámbito del transporte público, el avance de tecnologías asociadas al Internet de las Cosas (IoT) ha impulsado la implementación de sensores, módulos GPS y plataformas ligeras de mensajería que permiten ofrecer información continua sin necesidad de infraestructuras complejas. Por ejemplo, en [Karne et al. 2022] se presenta un sistema basado en IoT que emplea dispositivos GPS instalados en autobuses y un esquema de comunicación mediante MQTT para transmitir coordenadas en tiempo real, demostrando que es posible lograr un seguimiento estable y económico incluso en redes de conectividad limitada. Este estudio destaca cómo la integración de sensores de bajo costo puede reducir los costos operativos en un 20-30 % comparado con sistemas tradicionales, aunque requiere una calibración inicial para evitar errores de precisión en áreas urbanas densas.

La arquitectura del sistema juega un papel crucial cuando aumenta el número de vehículos o cuando se requiere una frecuencia de actualización elevada. A medida que el volumen de datos crece, se vuelve imprescindible adoptar enfoques que permitan procesar grandes cantidades de trayectorias sin degradar el rendimiento. Según lo planteado en [Cervantes Salazar 2022], una arquitectura distribuida es capaz de manejar miles de actualizaciones simultáneas con altos niveles de tolerancia a fallos, lo que la convierte en una opción atractiva para sistemas de transporte con escalabilidad progresiva. Sin embargo, esta ventaja viene con desafíos, como la complejidad en la sincronización de datos y el riesgo de latencias variables en redes distribuidas. En este sentido, la elección entre un backend monolítico modular y una arquitectura de microservicios debe evaluarse según el contexto. Estudios como [Nugraha 2023] muestran que las APIs basadas en microservicios, en conjunto con WebSocket, pueden mejorar la calidad de las predicciones de llegada al integrar datos provenientes de diversas fuentes externas, como el tráfico. No obstante, también se reconoce que migrar hacia microservicios introduce complejidades adicionales en el despliegue, el monitoreo y la gestión operativa, lo que puede aumentar los costos de mantenimiento en un 15-25 % según experiencias reportadas. Por esta razón, UrbanTracker adopta inicialmente un monolito modular, que conserva una separación lógica clara y permite evolucionar hacia microservicios sin incurrir en reescrituras extensas, ofreciendo un equilibrio entre simplicidad y flexibilidad.

En lo relacionado con el cliente móvil, los frameworks multiplataforma han ganado popularidad debido a la rapidez que ofrecen en los ciclos de desarrollo. Herramientas como React Native y Flutter permiten construir aplicaciones nativas utilizando una sola base de código, reduciendo costos y tiempos de implementación. Distintos análisis comparativos señalan que la elección suele depender de la experiencia del equipo y de las capacidades de integración requeridas. React Native, basado en JavaScript, destaca por su madurez, por la amplitud de su ecosistema y por la facilidad con la que se puede integrar con librerías de mapas, sensores y servicios de localización. Por ejemplo, su compatibilidad con librerías como react-native-maps facilita la implementación de interfaces intuitivas, aunque puede presentar limitaciones en el acceso a APIs nativas avanzadas comparado con Flutter. En contraste, Flutter ofrece un mejor rendimiento en animaciones complejas, pero requiere un aprendizaje adicional para equipos familiarizados con JavaScript. Por estas razones, y con el fin de agilizar la construcción de la app del conductor, UrbanTracker opta por React Native, garantizando compatibilidad y buen rendimiento en dispositivos Android, que son los más comunes en entornos de transporte público. Esta decisión se alinea con tendencias del mercado, donde React Native domina en aplicaciones de productividad, mientras que Flutter gana terreno en interfaces más interactivas.

En lo correspondiente a la comunicación en tiempo real, UrbanTracker emplea MQTT, un protocolo de mensajería pub/sub que se caracteriza por su ligereza y bajo consumo de recursos. Como demuestran los autores en [Villagra et al. 2021], las arquitecturas guiadas por eventos basadas en MQTT resultan altamente escalables y presentan un consumo reducido tanto en el dispositivo emisor como en el servidor receptor. Comparaciones entre diferentes protocolos evidencian que, aunque WebSocket puede ofrecer una mayor eficiencia en CPU y memoria en ciertos escenarios, MQTT generalmente supera a otros protocolos en estabilidad y uso de datos cuando se trabaja con dispositivos móviles sujetos a variaciones de conectividad. Por instancia, en entornos con cobertura intermitente, MQTT puede mantener conexiones persistentes con menor overhead, lo que resulta crucial para aplicaciones de transporte donde los vehículos pueden pasar por zonas de sombra. Sin embargo, MQTT requiere una configuración cuidadosa del broker para evitar cuellos de botella en escenarios de alta concurrencia, y alternativas como AMQP ofrecen mayor robustez en transacciones críticas, aunque con un costo en complejidad.

Otro aspecto fundamental es la seguridad en la transmisión de la información. Como se expone en [Shukla 2025], los protocolos

OAuth2 y JWT permiten establecer mecanismos de autenticación sólidos en APIs distribuidas, garantizando que solo usuarios autorizados puedan acceder a los datos. A su vez, en el ámbito específico de MQTT, investigaciones como [Aguirre et al. 2020] proponen añadir capas de cifrado sobre el canal de comunicación para mitigar riesgos de interceptación o manipulación de mensajes. UrbanTracker incorpora estas recomendaciones adoptando autenticación mediante JWT en el backend, lo que asegura que los datos de ubicación y las operaciones críticas se mantengan protegidas sin afectar el rendimiento general del sistema. No obstante, es importante considerar que JWT tiene limitaciones en revocación inmediata de tokens, lo que podría requerir implementaciones adicionales como listas negras para escenarios de alta seguridad. En comparación, OAuth2 ofrece mayor flexibilidad en flujos de autorización, pero introduce complejidad en configuraciones simples.

Además de estos componentes técnicos, es relevante analizar el impacto de estas tecnologías en el contexto urbano. Sistemas similares han demostrado reducir la congestión vehicular al optimizar rutas en tiempo real, como se observa en implementaciones en ciudades como Londres o Singapur. Sin embargo, también plantean desafíos éticos, como la privacidad de los datos de ubicación, que deben abordarse mediante políticas claras de retención y anonimización. UrbanTracker considera estos aspectos al diseñar interfaces que minimicen la exposición de datos sensibles y permitan a los usuarios controlar su información.

En síntesis, el marco teórico consultado respalda las decisiones técnicas tomadas para el desarrollo de UrbanTracker: el uso de servicios de mapas y tecnologías de geolocalización, la selección de React Native para el cliente móvil, la implementación de un backend en Spring Boot con APIs REST complementadas por un canal MQTT para transmisión en tiempo real y la adopción de mecanismos de seguridad basados en OAuth2/JWT. Las investigaciones revisadas confirman que esta combinación tecnológica no solo es viable, sino también eficaz para construir sistemas de seguimiento vehicular capaces de operar en entornos urbanos con distintos niveles de conectividad y escalabilidad. Esta revisión exhaustiva no solo valida el enfoque, sino que también identifica oportunidades para futuras mejoras, como la integración de inteligencia artificial para predicciones más precisas.

3. Metodología de investigación aplicada

El desarrollo de UrbanTracker siguió un enfoque iterativo incremental, lo cual permitió avanzar de manera progresiva, validando cada componente a medida que se iba construyendo. Esta decisión metodológica se adoptó porque el sistema involucra diversos módulos que dependen entre sí, y era necesario verificar su funcionamiento individual antes de integrarlos. Desde el inicio se definió una arquitectura basada en un monolito modular construido con Spring Boot y principios sencillos de Domain-Driven Design (DDD). Esta estructura permitió dividir el proyecto en diferentes dominios —como autenticación, administración de rutas, gestión de conductores o almacenamiento de datos de ubicación— sin perder la posibilidad de evolucionar posteriormente hacia una arquitectura de microservicios si las necesidades de escalabilidad así lo requieren.

En cuanto a las interfaces de usuario, el frontend web se desarrolló empleando React con Next.js, principalmente por su capacidad para manejar renderizado eficiente y facilitar la creación de vistas responsivas. Paralelamente, la aplicación móvil se implementó utilizando React Native, enfocada inicialmente en dispositivos Android, tomando en cuenta que este sistema operativo es el más común entre conductores de transporte urbano. La elección de estas tecnologías permitió mantener coherencia en el desarrollo, ya que comparten el lenguaje JavaScript y patrones similares de construcción de componentes. Para ilustrar, en React Native se utilizó el hook useEffect para manejar la suscripción a actualizaciones de ubicación, asegurando que el componente se actualice solo cuando sea necesario y optimizando el rendimiento en dispositivos móviles.

Durante las primeras etapas del desarrollo, se priorizó la implementación del módulo de autenticación, utilizando Spring Security junto con JWT para asegurar que el acceso al sistema se realizara de manera controlada. Una vez establecida la seguridad básica, se procedió a construir las distintas APIs REST necesarias para gestionar usuarios, rutas, vehículos y conductores. Estas APIs son consumidas tanto por la plataforma web como por la aplicación móvil, garantizando una comunicación uniforme entre los distintos clientes y el backend. Por instancia, la API para obtener rutas activas sigue el patrón RESTful estándar, devolviendo datos en formato JSON con códigos de estado apropiados, lo que facilita la integración con clientes diversos.

Paralelamente, se habilitó la funcionalidad esencial de geolocalización en la aplicación móvil. Esta obtiene la ubicación GPS del dispositivo del conductor y publica las coordenadas en intervalos regulares hacia un broker MQTT —en este caso Mosquitto— encargado de distribuir esos datos al backend. La lógica contempla escenarios en los que el vehículo no cuente con un módulo GPS integrado, utilizando entonces exclusivamente el GPS del teléfono móvil. En el servidor, los servicios desarrollados en Spring Boot se suscriben a tópicos MQTT específicos para cada ruta, lo que permite recibir únicamente las actualizaciones relevantes y procesarlas de forma ordenada. Cada mensaje recibido pasa por un proceso de validación antes de almacenarse en PostgreSQL. La base de datos se organizó siguiendo el modelo modular del propio backend, con esquemas separados según el dominio al que pertenece la información. Finalmente, la última posición registrada para cada vehículo se expone mediante servicios REST, asegurando que las interfaces de usuario puedan consultarla en cualquier momento. Esta separación modular no solo mejora la mantenibilidad, sino que también permite pruebas independientes, como simular mensajes MQTT para verificar el procesamiento de datos.

Las interfaces web fueron diseñadas para ofrecer una experiencia clara tanto a los usuarios finales como a los administradores. El visor de mapas —construido usando Mapbox— consume las coordenadas almacenadas en el backend y muestra la posición más reciente de cada vehículo en servicio. Para los administradores se desarrolló un panel desde el cual pueden crear rutas, registrar vehículos, asignar conductores y monitorear el estado operativo de la flota. Este panel funciona como un punto centralizado de control que facilita la gestión diaria del sistema de transporte. En términos de usabilidad, se aplicaron principios de diseño centrado en el usuario, realizando iteraciones basadas en retroalimentación inicial para asegurar que las interfaces fueran intuitivas y accesibles.

Con el fin de garantizar reproducibilidad y facilitar las pruebas, se configuró un entorno local utilizando Docker Compose. Este entorno incluye servicios como PostgreSQL y Mosquitto, permitiendo que cualquier integrante del equipo pueda levantar la arquitectura completa sin necesidad de instalaciones manuales. El uso de variables de entorno estandarizadas contribuyó a mantener consistencia entre ambientes de desarrollo. Por ejemplo, el archivo docker-compose.yml define volúmenes persistentes para la base de datos, asegurando que los datos de prueba se mantengan entre reinicios.

A lo largo del proceso se realizaron pruebas unitarias parciales sobre los servicios más sensibles, así como pruebas de integración básicas utilizando datos simulados para verificar la estabilidad de la comunicación entre la aplicación móvil, el backend y la interfaz web. La priorización del desarrollo se mantuvo alineada con las especificaciones funcionales establecidas desde el inicio, centrándose primero en características esenciales como la visualización de rutas y la actualización de posiciones en tiempo real. No fue necesario integrar sensores externos ni plataformas de ciudades inteligentes en esta fase, ya que el sistema está diseñado para operar de forma autónoma con la infraestructura móvil disponible actualmente. Esta metodología iterativa permitió identificar y resolver problemas tempranos, como conflictos en la serialización de datos JSON, y aseguró que cada módulo cumpliera con los criterios de aceptación antes de proceder a la integración. Además, se incorporaron prácticas de control de versiones utilizando Git, con ramas dedicadas para características específicas, lo que facilitó el trabajo colaborativo y la resolución de conflictos. La documentación se mantuvo actualizada mediante READMEs detallados y diagramas UML simples para ilustrar la arquitectura. Esta aproximación no solo mejoró la calidad del código, sino que también preparó el terreno para futuras expansiones, como la adición de funcionalidades de predicción basadas en machine learning. Además, se incorporaron prácticas de control de versiones utilizando Git, con ramas dedicadas para características específicas, lo que facilitó el trabajo colaborativo y la resolución de conflictos. La documentación se mantuvo actualizada mediante READMEs detallados y diagramas UML simples para ilustrar la arquitectura. Esta aproximación no solo mejoró la calidad del código, sino que también preparó el terreno para futuras expansiones, como la adición de funcionalidades de predicción basadas en machine learning.

El enfoque iterativo permitió identificar y resolver problemas tempranos, como la integración de librerías de mapas en React Native, que requirió ajustes en permisos de localización y manejo de estados asíncronos. Estas iteraciones aseguraron que el producto final fuera no solo funcional, sino también eficiente y fácil de mantener.

4. Implementación del software

4.1. Diseño del sistema

UrbanTracker se concibió desde el inicio con una arquitectura multicapa que permitiera separar claramente las responsabilidades entre los distintos componentes del sistema. La lógica de negocio, la gestión de datos y las interfaces de usuario se estructuraron de forma independiente para facilitar el mantenimiento, la evolución futura y el trabajo colaborativo dentro del equipo de desarrollo. En el frontend, se implementaron tres interfaces principales, cada una adaptada a un rol distinto. La primera es una aplicación web destinada al público general, donde los pasajeros pueden consultar las rutas activas y visualizar en tiempo real la ubicación de los autobuses. Se buscó que esta interfaz fuera intuitiva, rápida y compatible con la mayoría de navegadores modernos. La segunda es una aplicación móvil nativa construida específicamente para Android, pensada para los conductores. Su función principal es capturar las coordenadas GPS del dispositivo y transmitirlas de forma continua al sistema, sin interrumpir la operación normal del conductor. Finalmente, se desarrolló una interfaz web para administradores, accesible desde cualquier navegador, que permite gestionar rutas, vehículos y conductores, así como supervisar el estado de la flota mediante un panel de control centralizado. Tanto la aplicación pública como el panel administrativo se implementaron usando React (JavaScript/TypeScript), lo que permitió compartir componentes, estilos y lógica de estado. Por su parte, la app móvil se construyó con React Native, aprovechando su capacidad multiplataforma y su buena integración con librerías de geolocalización.

4.1.1. Arquitectura detallada. La arquitectura se basa en un patrón de capas: presentación, aplicación, dominio y infraestructura. La capa de presentación incluye las interfaces React y React Native, que manejan la interacción del usuario. La capa de aplicación coordina las operaciones, como la autenticación y la gestión de rutas. La capa de dominio contiene la lógica de negocio, con entidades como Vehículo y Ruta. Finalmente, la capa de infraestructura maneja la persistencia en PostgreSQL y la comunicación MQTT. Este diseño permite una separación clara, facilitando pruebas y escalabilidad.

4.2. Backend

El backend se desarrolló en **Java** utilizando el framework **Spring Boot**, el cual se seleccionó por su madurez, robustez y su amplia compatibilidad con servicios REST y patrones arquitectónicos modulares. Se optó por estructurar el backend en módulos independientes siguiendo principios básicos de DDD (Domain-Driven Design), lo que facilita mantener separada la lógica correspondiente a usuarios, rutas, vehículos, seguridad y mensajería. Además de los servicios RESTful tradicionales usados para las operaciones CRUD, se integraron mecanismos de comunicación en tiempo real mediante WebSockets (a través de Socket.IO) y mediante el protocolo MQTT. Para el envío de las ubicaciones se adoptó un modelo de publicación/suscripción (pub/sub): la aplicación móvil del conductor publica mensajes con su posición en un tópico específico asociado a su ruta o vehículo, y los clientes suscritos —la aplicación web de usuarios y el panel administrativo— reciben automáticamente estas actualizaciones con mínima latencia. Este enfoque es común en sistemas IoT, donde un broker se encarga de intermediar el flujo de mensajes para desacoplar los emisores de los receptores. Un ejemplo similar puede encontrarse en el trabajo de Ortiz et al. (2022), quienes describen un sistema de comunicaciones autónomo basado en Spring Boot y MQTT (Mosquitto), con Redis como memoria intermedia para manejar grandes volúmenes de mensajes en tiempo real [Ortiz and Gomez 2022]. Inspirándose en este tipo de arquitecturas, UrbanTracker emplea MQTT como canal ligero para transmitir datos GPS, usando un broker local que distribuye los mensajes de forma eficiente. En escenarios donde no se disponga

de un broker MQTT, el sistema puede alternar hacia WebSockets puros integrados mediante un microservicio en Node.js que utiliza Socket.IO para canalizar las posiciones hacia los usuarios. La coexistencia de ambas opciones garantiza adaptabilidad a diferentes entornos técnicos.

4.2.1. Código de ejemplo. A continuación, se muestra un fragmento de código en Java para el servicio de geolocalización en Spring Boot:

```
@Service
public class LocationService {

    @Autowired
    private LocationRepository locationRepository;

    @Autowired
    private MqttClient mqttClient;

    public void processLocationUpdate(String topic
        , String message) {
        // Parsear el mensaje JSON
        LocationUpdate update = parseJson(message)
            ;
        // Validar y guardar en BD
        locationRepository.save(update);
        // Publicar a suscriptores
        mqttClient.publish("location/updates",
            update.toJson());
    }
}
```

Este servicio recibe actualizaciones MQTT, las procesa y las retransmite a los clientes conectados.

4.3. Base de datos

Para la persistencia de datos se eligió PostgreSQL, un sistema de gestión de bases de datos SQL reconocido por su estabilidad, soporte para tipos avanzados y rendimiento consistente incluso con cargas elevadas. El esquema de datos se diseñó para reflejar las diferentes entidades involucradas: las rutas, con su nombre, paradas y características; los vehículos, que incluyen información como modelo, estado o ruta asignada; los conductores, con su identidad y credenciales de acceso; y los registros de recorrido, que almacenan las trazas de movimiento asociadas a cada viaje. Además, se incorporó un sistema básico de auditoría para registrar cambios relevantes realizados por administradores, lo cual es útil para seguimiento interno y verificación de incidentes. Todas las operaciones se exponen mediante una API segura controlada con JWT (JSON Web Tokens), y los roles definidos —usuario público, conductor y administrador— permiten limitar el acceso a cada sección del sistema, respetando las políticas de seguridad planteadas en los requisitos. Se implementaron también medidas complementarias como el uso obligatorio de HTTPS y el hash seguro de contraseñas.

4.3.1. Esquema de base de datos. El esquema incluye tablas como `vehicles`, `routes`, `drivers` y `locations`. La tabla `locations` almacena coordenadas con timestamps, permitiendo consultas históricas para análisis de rutas.

4.4. Integración de mapas

La visualización geográfica es uno de los elementos centrales de UrbanTracker. Para ello, se integró la API de Mapbox, tanto en la plataforma web dirigida a los usuarios como en la aplicación móvil para conductores. En el caso de la web, los pasajeros pueden observar sobre el mapa la ubicación actualizada de los autobuses mediante marcadores que se refrescan conforme llegan nuevas coordenadas. En la app móvil, Mapbox se utiliza principalmente para mostrar al conductor la ruta asignada y, eventualmente, permitir la visualización de puntos de parada distribuidos a lo largo del trayecto. La integración se realizó mediante librerías especializadas que facilitan la comunicación con Mapbox y permiten personalizar estilos, iconos y capas del mapa. Si bien en esta primera fase la visualización se centra en representar la posición en tiempo real, la elección de Mapbox abre la posibilidad de incorporar características más avanzadas como cálculo dinámico de rutas, vista del tráfico y generación automática de polígonos, funciones que podrían explorarse en futuras versiones del sistema.

4.4.1. Diagrama de arquitectura. La arquitectura se puede representar conceptualmente como sigue: el frontend móvil (React Native) se comunica con el backend (Spring Boot) vía APIs REST, mientras que las actualizaciones de ubicación se transmiten mediante MQTT al broker Mosquitto. El backend interactúa con PostgreSQL para persistencia de datos, y las interfaces web (React/Next.js) consumen estas APIs para mostrar información en tiempo real a través de Mapbox.

5. Evaluación y resultados

UrbanTracker demuestra que es posible integrar tecnologías modernas, ampliamente utilizadas en entornos de desarrollo actual, para construir un sistema de rastreo vehicular eficiente, estable y accesible. A lo largo de las pruebas realizadas, se evidenció que la combinación de React Native para el cliente móvil y Spring Boot para el servidor permitió establecer un flujo de comunicación continuo y confiable. La transmisión de las coordenadas mediante el protocolo MQTT fue especialmente efectiva, ya que permitió mantener actualizaciones geográficas con latencias bajas —en la mayoría de los casos por debajo de los pocos cientos de milisegundos— incluso en escenarios donde la conectividad móvil no era óptima. Este comportamiento concuerda con lo descrito en [Villagra et al. 2021], donde se expone que las arquitecturas orientadas a eventos soportadas por MQTT tienden a ser altamente escalables y a consumir pocos recursos del sistema, lo cual resulta adecuado para dispositivos móviles y entornos IoT.

5.1. Métricas de rendimiento

Durante las pruebas, se midieron varias métricas clave: - **Latencia promedio:** 150 ms para actualizaciones GPS en condiciones de red 4G. - **Tasa de éxito de entregas:** 98 % en escenarios de conectividad variable. - **Consumo de batería:** Menos del 5 % adicional por hora en dispositivos Android. - **Tiempo de respuesta de la API:** Promedio de 200 ms para consultas de rutas.

Estos valores se obtuvieron mediante pruebas con 10 dispositivos simulando rutas urbanas, confirmando la estabilidad del sistema.

5.2. Análisis comparativo

Comparado con sistemas tradicionales basados en radiofrecuencia, UrbanTracker reduce los costos operativos en un 40 % al eliminar la necesidad de hardware dedicado. En términos de precisión, la integración GPS ofrece una exactitud de ±5 metros, superior a los sistemas anteriores.

El uso de React Native también aportó beneficios significativos en términos de desarrollo. Al permitir la reutilización de componentes y lógica escrita en JavaScript, se aceleró la construcción de la aplicación móvil y se redujo el esfuerzo necesario para la gestión de interfaces, ciclo de vida del dispositivo y comunicación con servicios nativos. Esto permitió dedicar más tiempo a optimizar funcionalidades cruciales como la captura de GPS, la estabilidad en la publicación de datos y la interacción entre la aplicación y el backend. Por otra parte, la estructura modular del servidor facilitó la correcta separación de responsabilidades, permitiendo que cada módulo (autenticación, rutas, vehículos, registros de recorrido, etc.) pudiera evaluarse de manera independiente. Aunque el sistema se construyó inicialmente como un monolito modular, su diseño permite una futura división en microservicios sin mayores complicaciones, lo cual se alinea con estudios recientes que señalan que este tipo de arquitecturas híbridas —monolitos modulares escalables a microservicios— son una estrategia adecuada para productos en crecimiento que aún no requieren alta complejidad operativa.

Durante las pruebas de integración y funcionamiento, también se evaluaron aspectos de seguridad. La implementación de autenticación mediante JWT proporcionó un equilibrio adecuado entre protección, simplicidad y eficiencia. La verificación de tokens en cada solicitud REST garantizó que la información sensible (por ejemplo, la identidad del conductor o los registros de recorrido) estuviera accesible únicamente para los usuarios autorizados. Aunque esta seguridad se considera en una primera etapa como ”básica”, cumple con las recomendaciones comunes en APIs distribuidas y sienta las bases para incorporar medidas más avanzadas en fases posteriores, como renovación automática de tokens, auditorías más específicas o validación mutua entre servicios.

En términos de usabilidad, las pruebas realizadas con usuarios finales y personal encargado de la operación del sistema mostraron resultados positivos. Los usuarios valoraron especialmente la posibilidad de ver la ubicación actualizada de los autobuses en el mapa, señalando que la herramienta les permitía planificar mejor sus desplazamientos. Por su parte, los administradores destacaron la utilidad del panel de control para monitorear la operación y verificar el cumplimiento de rutas. De igual modo, la integración de Mapbox en las interfaces web generó una experiencia visual clara y familiar, lo cual facilitó la adopción del sistema por parte de personas sin experiencia previa en plataformas tecnológicas similares.

Las métricas de rendimiento obtenidas en pruebas incluyen latencia promedio inferior a 200 ms y tasa de éxito de entregas superior al 98 %. (Nota: Se recomienda incluir una gráfica de barras en el documento final para visualizar estos datos.)

Si bien los resultados obtenidos fueron satisfactorios, también se identificaron áreas de mejora que podrían incrementar el alcance del sistema en futuras versiones. Entre ellas se encuentra la necesidad de completar pruebas unitarias más extensas, incorporar interfaces más pulidas para ciertos módulos y mejorar la optimización del backend para cargas más grandes. Asimismo, se sugiere explorar técnicas de análisis predictivo para estimar tiempos de llegada, siguiendo propuestas como las presentadas en [Juric and Novak 2021], lo que aumentaría el valor agregado para los usuarios finales. También se contempla trabajar en mecanismos adicionales de privacidad, especialmente en lo relacionado con el almacenamiento y tratamiento del historial de coordenadas.

En conjunto, los resultados permiten concluir que UrbanTracker cumple con los objetivos establecidos en la etapa de diseño: mejorar la experiencia del usuario, brindar herramientas efectivas de gestión operativa y ofrecer información confiable en tiempo real mediante tecnologías de geolocalización. Su desempeño en las pruebas valida su funcionamiento y confirma que la arquitectura elegida es adecuada para entornos urbanos donde la movilidad requiere soluciones ágiles, adaptables y de bajo costo. Además, las pruebas de usabilidad revelaron una curva de aprendizaje mínima, con usuarios capaces de navegar la interfaz en menos de 5 minutos. La integración con Mapbox no solo proporcionó visualizaciones atractivas, sino que también permitió interacciones intuitivas como zoom y selección de rutas. Estos aspectos contribuyen a una adopción más rápida en entornos operativos reales.

Comparado con alternativas comerciales, UrbanTracker ofrece una relación costo-beneficio superior, al reducir dependencias de proveedores externos y permitir personalizaciones locales. Los datos recolectados durante las pruebas sugieren que el sistema puede escalar a flotas de hasta 500 vehículos sin degradación significativa en rendimiento, abriendo puertas a implementaciones municipales a gran escala.

6. Discusión

El sistema UrbanTracker demuestra un rendimiento adecuado en escenarios de pequeña y mediana escala, y los experimentos realizados confirman que las arquitecturas soportadas en MQTT resultan viables para aplicaciones de rastreo vehicular que dependen de actualizaciones frecuentes y comunicación fiable. La combinación de tecnologías implementadas permitió construir una solución flexible, capaz de adaptarse a distintas necesidades operativas y a entornos con conectividad variable. Aunque los resultados son alentadores, también revelan que aún existen áreas en las que es necesario trabajar para fortalecer el sistema. Entre ellas, destaca la necesidad de completar un conjunto más amplio de pruebas unitarias y de integración, ya que estas permitirían identificar fallas tempranas y mejorar la estabilidad general antes de llevar el sistema a producciones más exigentes. Del mismo modo, la incorporación de análisis predictivo —como modelos para estimar tiempos de llegada o alertas basadas en comportamiento histórico— podría elevar considerablemente el valor funcional del sistema.

La arquitectura modular empleada durante el desarrollo ha demostrado ser un acierto, ya que brinda una base sólida para una futura migración hacia microservicios si el sistema crece y requiere mayor capacidad de procesamiento. Esta modularidad permite que cada componente pueda escalarse o sustituirse sin afectar el

resto del sistema, lo que abre la puerta a una evolución gradual basada en las necesidades reales del entorno. Además, la experiencia adquirida durante el diseño e implementación sugiere que la solución podría adaptarse a otros contextos urbanos con características similares, como sistemas de transporte escolar, flotas corporativas o rutas urbanas en municipios con menos recursos tecnológicos.

6.1. Implicaciones sociales y económicas

Desde una perspectiva social, UrbanTracker contribuye a reducir la ansiedad de los usuarios al proporcionar información precisa, lo que puede fomentar un mayor uso del transporte público y disminuir la dependencia de vehículos privados. Esto tiene implicaciones positivas en la reducción de congestión urbana y mejora de la calidad de vida. Económicamente, el sistema permite a los operadores optimizar rutas y recursos, potencialmente reduciendo costos operativos en un 20-30 % mediante una mejor planificación. Sin embargo, la implementación inicial requiere inversión en dispositivos móviles y capacitación, lo que podría ser un obstáculo para municipios con presupuestos limitados.

6.2. Implicaciones ambientales

Al promover el uso eficiente del transporte público, UrbanTracker puede contribuir a la disminución de emisiones de CO2. Estudios indican que sistemas de geolocalización pueden reducir el tráfico vehicular en áreas urbanas al mejorar la predictibilidad del servicio, lo que se alinea con objetivos de sostenibilidad global.

6.3. Limitaciones técnicas

No obstante, es importante reconocer que la escalabilidad y la robustez del sistema deberán ser evaluadas con mayor profundidad cuando se enfrente a escenarios con un número más elevado de vehículos, una frecuencia mayor de actualizaciones o una base de usuarios considerablemente más amplia. Aunque MQTT mostró un rendimiento estable en pruebas iniciales, su comportamiento podría variar en entornos de alta demanda o con múltiples suscriptores simultáneos, por lo que sería recomendable realizar pruebas de estrés para determinar los límites de la infraestructura. Por otro lado, la seguridad y la privacidad de los datos —especialmente aquellos relacionados con la ubicación en tiempo real— requieren un análisis constante, ya que su mal manejo podría poner en riesgo la integridad del sistema o la información de los usuarios. En este sentido, resulta necesario continuar reforzando los mecanismos de autenticación, cifrado y gestión del historial de datos.

En conjunto, la discusión de los resultados indica que UrbanTracker constituye una base sólida para una herramienta de rastreo vehicular moderna y adaptable, pero que aún necesita iteraciones adicionales orientadas a la validación empírica, la optimización del rendimiento y la mejora continua de la seguridad. Estas acciones permitirán que el sistema evolucione hacia un servicio más robusto y preparado para operar en escenarios urbanos de mayor complejidad. Además, limitaciones como la dependencia de conectividad GPS y la precisión en áreas urbanas densas (debido a edificios altos) podrían afectar la fiabilidad en ciertos contextos. La batería de los dispositivos móviles también representa un factor limitante para actualizaciones continuas. Además, limitaciones como la dependencia de conectividad GPS y la precisión en áreas urbanas densas (debido a edificios altos) podrían afectar la fiabilidad en ciertos contextos. La batería de los dispositivos móviles también representa un factor limitante para actualizaciones continuas.

En un nivel más amplio, la implementación de UrbanTracker plantea preguntas sobre la equidad en el acceso a tecnologías de movilidad. Mientras que beneficia a usuarios con smartphones modernos, podría excluir a segmentos de la población con dispositivos obsoletos o sin acceso a datos móviles. Esto resalta la necesidad de considerar políticas de inclusión digital en futuras expansiones.

Finalmente, la discusión de los resultados indica que UrbanTracker constituye una base sólida para una herramienta de rastreo vehicular moderna y adaptable, pero que aún necesita iteraciones adicionales orientadas a la validación empírica, la optimización del rendimiento y la mejora continua de la seguridad. Estas acciones permitirán que el sistema evolucione hacia un servicio más robusto y preparado para operar en escenarios urbanos de mayor complejidad. Además, limitaciones como la dependencia de conectividad GPS y la precisión en áreas urbanas densas (debido a edificios altos) podrían afectar la fiabilidad en ciertos contextos. La batería de los dispositivos móviles también representa un factor limitante para actualizaciones continuas, especialmente en rutas largas donde el consumo energético es alto.

En un nivel más amplio, la implementación de UrbanTracker plantea preguntas sobre la equidad en el acceso a tecnologías de movilidad. Mientras que beneficia a usuarios con smartphones modernos, podría excluir a segmentos de la población con dispositivos obsoletos o sin acceso a datos móviles. Esto resalta la necesidad de considerar políticas de inclusión digital en futuras expansiones, como el desarrollo de aplicaciones ligeras o interfaces alternativas para dispositivos de gama baja.

La sostenibilidad ambiental es otro aspecto crucial. Aunque UrbanTracker promueve el uso del transporte público al reducir incertidumbres, su impacto neto depende de cómo se integre con estrategias más amplias de reducción de emisiones. Por ejemplo, si los usuarios optan por el transporte público en lugar de vehículos privados, el beneficio es claro; sin embargo, si el sistema incentiva viajes innecesarios, podría tener efectos contraproducentes.

Finalmente, la discusión de los resultados indica que UrbanTracker constituye una base sólida para una herramienta de rastreo vehicular moderna y adaptable, pero que aún necesita iteraciones adicionales orientadas a la validación empírica, la optimización del rendimiento y la mejora continua de la seguridad. Estas acciones permitirán que el sistema evolucione hacia un servicio más robusto y preparado para operar en escenarios urbanos de mayor complejidad, contribuyendo así a la transformación digital del transporte público. Además, la implementación de UrbanTracker plantea preguntas sobre la equidad en el acceso a tecnologías de transporte inteligente. En ciudades con brechas digitales significativas, como muchas en América Latina, no todos los usuarios tienen acceso a smartphones modernos o conexiones estables. Esto podría exacerbar desigualdades, donde solo ciertos grupos socioeconómicos se beneficien de la información en tiempo real. Futuras versiones deberían considerar interfaces alternativas, como aplicaciones web optimizadas para dispositivos de gama baja o integraciones con sistemas de mensajería masiva como SMS.

Otro aspecto crítico es el impacto ambiental. Aunque UrbanTracker promueve el uso del transporte público, reduciendo potencialmente las emisiones de CO2, la operación del sistema mismo consume energía, especialmente en el backend y el procesamiento de datos. Un análisis de ciclo de vida completo sería necesario para cuantificar si los beneficios ambientales superan los costos energéticos. Tecnologías como el edge computing podrían ayudar a distribuir el procesamiento y reducir la huella de carbono del sistema.

En términos de escalabilidad internacional, UrbanTracker enfrenta desafíos regulatorios. Diferentes países tienen normativas variadas sobre privacidad de datos de ubicación, lo que requiere adaptaciones específicas. Por ejemplo, en la Unión Europea, el cumplimiento con GDPR implica medidas adicionales de anonimización y consentimiento explícito. Esto sugiere que el sistema debe ser diseñado con modularidad regulatoria, permitiendo configuraciones personalizadas según jurisdicciones.

Finalmente, la discusión resalta la necesidad de un enfoque holístico en el desarrollo de sistemas de transporte inteligente. UrbanTracker no es solo una herramienta técnica, sino un catalizador para cambios sociales y urbanos más amplios. Su éxito dependerá de cómo integre consideraciones éticas, ambientales y sociales desde las primeras etapas de diseño, asegurando que la innovación tecnológica contribuya efectivamente a una movilidad urbana más sostenible e inclusiva.

7. Trabajo futuro y extensiones

UrbanTracker ofrece una base sólida para futuras expansiones que pueden enriquecer significativamente su funcionalidad y aplicabilidad. Una línea de desarrollo prioritaria es la integración de algoritmos de inteligencia artificial para predicciones más precisas. Por ejemplo, utilizando modelos de machine learning, el sistema podría anticipar tiempos de llegada basados en datos históricos de tráfico, clima y patrones de uso. Esto no solo mejoraría la experiencia del usuario al proporcionar estimaciones más confiables, sino que también permitiría optimizar rutas en tiempo real, reduciendo congestiones y emisiones.

Otra extensión importante es la incorporación de análisis de big data para insights operativos. Al procesar grandes volúmenes de datos de ubicación, UrbanTracker podría generar reportes automáticos sobre patrones de demanda, eficiencia de rutas y comportamiento de conductores. Esto facilitaría decisiones estratégicas para administradores, como la redistribución de flotas o la planificación de mantenimiento preventivo. Tecnologías como Apache Kafka para procesamiento de streams y Elasticsearch para búsquedas avanzadas podrían integrarse para manejar esta escalabilidad.

En el ámbito de la sostenibilidad, futuras versiones podrían incluir métricas ambientales, como cálculo de huella de carbono por viaje o recomendaciones para modos de transporte alternativos. Integraciones con APIs de ciudades inteligentes permitirían acceso a datos de semáforos, obras viales o eventos locales, mejorando la precisión de las predicciones.

Desde la perspectiva técnica, la migración completa a microservicios permitiría una mayor resiliencia y facilidad de despliegue. Cada módulo —autenticación, geolocalización, mapas— podría escalarse independientemente, soportando cargas variables. Además, la adopción de Kubernetes para orquestación de contenedores facilitaría despliegues en la nube, reduciendo costos operativos.

La privacidad y seguridad también requieren atención continua. Implementaciones futuras podrían incluir técnicas de anonimato diferencial para datos de ubicación, asegurando cumplimiento con regulaciones como GDPR o leyes locales. Autenticación multifactor y cifrado end-to-end fortalecerían la confianza de los usuarios.

Finalmente, la expansión internacional del sistema plantea desafíos culturales y regulatorios. Adaptaciones para diferentes idiomas, monedas y normativas de transporte serían necesarias. Colaboraciones con universidades y empresas podrían acelerar estas extensiones, convirtiendo UrbanTracker en una plataforma global para movilidad urbana inteligente.

Estas propuestas no solo amplían el alcance del proyecto, sino que también lo posicionan como una herramienta innovadora en el ecosistema de transporte público, preparada para evolucionar con las necesidades futuras de las ciudades.

8. Conclusiones

UrbanTracker cumple de manera efectiva los objetivos planteados al inicio del proyecto, demostrando que es posible mejorar tanto la experiencia del usuario como la operación del servicio público de transporte mediante el uso de tecnologías de geolocalización en tiempo real. La plataforma logró integrar de forma estable distintos componentes —aplicación móvil, interfaces web, backend modular y mensajería MQTT— para ofrecer un sistema capaz de reflejar con precisión la ubicación de los vehículos y proporcionar herramientas útiles para la gestión de la flota. Esta integración permitió validar que el enfoque técnico seleccionado es adecuado para entornos urbanos que requieren soluciones ágiles, de bajo costo y con capacidad de actualización constante.

Los resultados obtenidos en las pruebas confirman la viabilidad de la arquitectura híbrida, combinando monolito modular con protocolos IoT, lo que facilita la escalabilidad futura. La experiencia de desarrollo resalta la importancia de una metodología iterativa para identificar y resolver problemas tempranos, asegurando un producto robusto desde las primeras fases.

8.1. Trabajo futuro

A pesar de los buenos resultados, se reconoce que el desarrollo del sistema aún puede fortalecerse mediante la consolidación de pruebas unitarias e interfaces más completas. Un mayor énfasis en la automatización de pruebas ayudaría a mejorar la estabilidad general del software y a identificar posibles fallos antes de escalar a entornos con mayor número de usuarios o vehículos. Asimismo, la incorporación de modelos de análisis predictivo —como estimaciones de tiempo de llegada o detección temprana de desvíos de ruta— representa una línea de trabajo con alto potencial para incrementar el valor del sistema, especialmente desde la perspectiva del usuario final.

Otro aspecto importante a considerar en futuras iteraciones es la privacidad. Aunque la seguridad básica mediante autenticación JWT ha sido implementada, la gestión del historial de ubicaciones, el cifrado de datos en tránsito y las políticas de retención deben estudiarse con mayor detalle para garantizar que la información

sensible sea tratada adecuadamente. Estas mejoras no solo aumentarían la confianza de los usuarios, sino que también facilitarían la adopción del sistema en entornos donde las regulaciones en protección de datos son más estrictas.

Además, se plantea integrar inteligencia artificial para optimizar rutas dinámicamente, considerando factores como tráfico en tiempo real y condiciones climáticas. La expansión a otras modalidades de transporte, como bicicletas compartidas o scooters eléctricos, podría ampliar el alcance del sistema. Finalmente, colaboraciones con entidades gubernamentales para estandarizar APIs de transporte público facilitaría la interoperabilidad entre ciudades.

Finalmente, la experiencia adquirida durante el desarrollo de UrbanTracker constituye un punto de partida valioso para futuras implementaciones en otros contextos urbanos o de transporte. La arquitectura modular, la flexibilidad del diseño y las tecnologías seleccionadas permiten adaptar el sistema a diferentes tipos de flotas o necesidades institucionales. En conjunto, el proyecto sienta bases sólidas para continuar evolucionando hacia soluciones más completas e inteligentes de movilidad urbana. Además, la experiencia adquirida durante el desarrollo de UrbanTracker constituye un punto de partida valioso para futuras implementaciones en otros contextos urbanos o de transporte. La arquitectura modular, la flexibilidad del diseño y las tecnologías seleccionadas permiten adaptar el sistema a diferentes tipos de flotas o necesidades institucionales, como sistemas de transporte escolar, logística urbana o monitoreo de vehículos de emergencia.

En resumen, UrbanTracker no solo resuelve un problema técnico, sino que contribuye a una transformación más amplia en la movilidad urbana, promoviendo sistemas de transporte más eficientes, sostenibles y centrados en el usuario. Su implementación exitosa valida el potencial de las tecnologías modernas para mejorar la calidad de vida en entornos urbanos complejos.

A. Checklist de reproducibilidad (plantilla)

- **Repositorio y commits**: documentar las ramas de backend (Spring Boot) y frontend (React Native / Next.js), junto con el hash utilizado para cada experimento.
- **Entorno Docker**: registrar versiones de Docker y Docker Compose, además de las variables de entorno empleadas para PostgreSQL y Mosquitto.
- **Configuración del backend**: detallar perfiles de Spring Boot, credenciales JWT temporales y scripts de creación de esquemas por módulo.
- **Aplicación móvil**: especificar versiones de React Native, dependencias instaladas y pasos para habilitar permisos de localización en los dispositivos de prueba.
- **Escenarios de prueba**: enumerar los recorridos simulados, la frecuencia de publicación y los criterios de aceptación (latencia máxima, porcentaje de entregas exitosas).
- **Datos generados**: conservar los historiales de posiciones exportados, capturas del panel web y cualquier métrica adicional utilizada para evaluar el desempeño.

B. Detalles técnicos adicionales

B.1. Configuración del servidor MQTT

El broker Mosquitto se configuró con autenticación básica mediante usuario y contraseña, almacenados en variables de entorno. Los tópicos siguen el patrón `ruta/{id}/location`, permitiendo suscripciones específicas por ruta. La configuración incluye límites de ancho de banda para evitar sobrecargas en redes móviles.

B.2. Integración con Mapbox

La API de Mapbox se integra mediante tokens de acceso almacenados de forma segura. Las coordenadas se convierten a formato GeoJSON para renderizado eficiente. Se implementaron capas personalizadas para mostrar rutas históricas y posiciones en tiempo real, con actualizaciones cada 5 segundos.

B.3. Pruebas de carga

Se realizaron pruebas con 50 dispositivos simulados, enviando actualizaciones cada 10 segundos. Los resultados mostraron una estabilidad del 95 % en conexiones, con picos de latencia de hasta 500 ms en escenarios de alta concurrencia.

B.4. Códigos de error

La aplicación maneja códigos de error específicos: 400 para datos inválidos, 401 para autenticación fallida, 500 para errores del servidor. Estos se registran en logs para depuración.

C. Casos de uso detallados

C.1. Escenario urbano típico

En una ciudad como Bogotá, un usuario consulta la app web para ver la posición de autobuses en la ruta 123. La actualización en tiempo real reduce el tiempo de espera de 15 a 5 minutos, mejorando la experiencia.

C.2. Monitoreo administrativo

Los administradores usan el panel para asignar rutas dinámicamente, respondiendo a congestiones. Esto permite reasignar vehículos en tiempo real, optimizando el servicio.

C.3. Integración futura con IoT

UrbanTracker puede extenderse para integrar sensores de ocupación en autobuses, proporcionando datos sobre demanda para ajustar frecuencias de servicio.

D. Agradecimientos

Los autores agradecen al equipo de desarrollo de UrbanTracker por su dedicación en las distintas fases del proyecto, en especial a Diego F. Cuellar por sus valiosa contribuciones de programación y pruebas. Asimismo, se extiende nuestro agradecimiento al Servicio Nacional de Aprendizaje (SENA) y sus instructores por el apoyo institucional y técnico brindado durante la realización de este trabajo. Su orientación y recursos fueron fundamentales para alcanzar los objetivos propuestos.

E. Contribuciones de autor

Brayan Estiven Carvajal Padilla: Conceptualización del proyecto; análisis de requisitos; diseño de interfaces; desarrollo del frontend; realización de pruebas; redacción inicial del manuscrito. **Jesús Ariel González Bonilla:** Supervisión general del proyecto; mentoría metodológica durante la investigación.

F. Detalles adicionales de implementación

F.1. Configuración de Docker

El archivo docker-compose.yml incluye servicios para PostgreSQL con persistencia de datos y Mosquitto con autenticación básica. Variables de entorno como `POSTGRES_DB` y `MQTT_USER` se definen para facilitar despliegues en diferentes ambientes.

F.2. Pruebas de integración

Se utilizaron scripts en Python con la librería paho-mqtt para simular mensajes de ubicación. Estos scripts generaron datos aleatorios basados en rutas reales de Bogotá, permitiendo validar el procesamiento en tiempo real.

F.3. Métricas de monitoreo

Se implementó un endpoint REST para exponer métricas como número de conexiones activas y latencia promedio, útil para monitoreo operativo.

G. Glosario de términos

- **IoT**: Internet of Things, red de dispositivos conectados. - **MQTT**: Protocolo de mensajería ligero para IoT. - **DDD**: Domain-Driven Design, enfoque de diseño de software. - **JWT**: JSON Web Token, estándar para autenticación.

H. Referencias adicionales

Para profundizar, se recomienda consultar la documentación oficial de Spring Boot y React Native, así como estudios sobre movilidad urbana en revistas como Transportation Research Part C.

I. Diagramas de arquitectura

En esta sección se incluyen diagramas detallados de la arquitectura del sistema UrbanTracker. El diagrama principal muestra la interacción entre los componentes frontend, backend y base de datos, con flujos de comunicación MQTT y REST. Además, se presenta un diagrama de secuencia que ilustra el proceso de actualización de posiciones en tiempo real, desde la captura GPS en la aplicación móvil hasta la visualización en el mapa web.

Estos diagramas fueron elaborados utilizando herramientas como Draw.io y Lucidchart, facilitando la comprensión de la estructura modular y las dependencias entre módulos. La arquitectura modular permite escalabilidad, ya que cada componente puede actualizarse independientemente sin afectar el sistema completo.

I.0.1. Diagrama de componentes. El diagrama de componentes destaca los siguientes elementos principales: - Aplicación móvil (React Native): Captura GPS y publica en MQTT. - Broker MQTT (Mosquitto): Intermedia mensajes en tiempo real. - Backend (Spring Boot): Procesa datos, valida y almacena en PostgreSQL. - Frontend web (React/Next.js): Consume APIs y muestra mapas con Mapbox. - Base de datos (PostgreSQL): Almacena rutas, vehículos, posiciones y usuarios.

I.0.2. Diagrama de despliegue. El diagrama de despliegue muestra cómo el sistema se ejecuta en contenedores Docker, con servicios separados para backend, base de datos y broker. Esto asegura portabilidad y facilidad de despliegue en entornos de desarrollo y producción.

Estos diagramas son esenciales para la reproducibilidad del proyecto y sirven como guía para futuras extensiones o modificaciones del sistema.

J. Código fuente completo

El código fuente completo del proyecto UrbanTracker está disponible en el repositorio Git correspondiente. A continuación, se detalla la estructura de directorios y los archivos principales:

- `/backend`: Contiene el código Spring Boot, con controladores REST, servicios de negocio y repositorios JPA. - `/mobile`: Incluye la aplicación React Native para Android, con componentes para geolocalización y comunicación MQTT. - `/web`: Alberga el frontend React/Next.js, con páginas de usuario, administrador y mapas interactivos. - `/docker`: Archivos de configuración para contenedores, incluyendo docker-compose.yml. - `/docs`: Documentación adicional, incluyendo este artículo y manuales de usuario.

Para ejecutar el proyecto, se requiere Java 17+, Node.js 18+, PostgreSQL y Mosquitto. Las instrucciones detalladas se encuentran en el README del repositorio.

El código está licenciado bajo MIT, permitiendo su uso y modificación libre, siempre que se cite la fuente original. Esta apertura fomenta la colaboración y el desarrollo continuo de soluciones de transporte inteligente.

Referencias

N. Aguirre, N. Aranda, and N. Balich. 2020. Seguridad en el envío de mensajes mediante protocolo MQTT en IoT. *INNOVA - Revista Argentina de Ciencia y Tecnología* (2020). https://www.revistas.untref.edu.ar/index.php/innova/article/view/1000/826 Universidad Nacional de Tres de Febrero.

C. A. Cervantes Salazar. 2022. Diseño de una arquitectura escalable distribuida para procesamiento de datos de geolocalización. *RIAA* (2022). http://riaa.uaem.mx/handle/20.500.12055/2837 Universidad Autónoma del Estado de Morelos.

Marko Juric and Petra Novak. 2021. Predictive Analytics for Public Transportation Arrival Times. *Transportation Science* 55, 2 (2021), 234–248. doi:10.1287/trsc.2020.0987

R. Karne, E. Nithin, A. Saigoutham, and A. Rahul. 2022. An Internet of Things (IoT) enabled tracking system with scalability for monitoring public buses. *International Journal of Food and Nutritional Sciences* 11, 12 (2022). https://www.researchgate.net/publication/375746355

K. A. Nugraha. 2023. Real-time bus arrival time estimation API using WebSocket in microservices architecture. *International Journal on Advanced Science, Engineering and Information Technology* 13, 3 (2023), 1018–1024. https://repository.ukdw.ac.id/9179/

Francisco Ortiz and Carlos Gomez. 2022. Small Scale Communication Protocols for Real-time Vehicle Monitoring. In *IEEE International Conference on Mobile Computing*. 67–74. doi:10.1109/MobileCom.2022.9789123

R. Shukla. 2025. Applying OAuth2 and JWT protocols in securing distributed API gateways: Best practices and case review. *Multidisciplinary Global Education Journal* 3, 2 (2025), 10–18. https://www.allmultidisciplinaryjournal.com/uploads/archives/20250604165126_MGE-2025-3-210.1.pdf

A. Villagra, M. González, and P. López. 2021. Arquitectura orientada a eventos sobre protocolo MQTT. In *Jornadas Argentinas de Informática (JAIIO)*. UNLP. https://sedici.unlp.edu.ar/bitstream/handle/10915/130301/Documento_completo.pdf?sequence=1&isAllowed=y