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

Autor: Brayan Estiven Carvajal Padilla

Afiliación: Servicio Nacional de Aprendizaje (SENA), Regional Huila, CIES, Ficha 2899747

Contacto: brayane_carvajalp@soy.sena.edu.co, brayancarvajalpadilla02@gmail.com

Fecha: 2025

Resumen

UrbanTracker es una plataforma que optimiza la movilidad urbana mediante el seguimiento en tiempo real de vehículos de transporte público. Integra una aplicación móvil para conductores (desarrollada en React Native) y una plataforma web para usuarios y administradores, comunicados con un backend en Java (Spring Boot) por MQTT. UrbanTracker permite visualizar en un mapa interactivo rutas y vehículos activos, así como gestionar rutas y conductores. En este trabajo se describen los requisitos del sistema, el marco teórico de tecnologías de geolocalización, el enfoque de desarrollo empleado, los resultados preliminares de funcionamiento y las conclusiones. Los resultados indican que la combinación de React Native, Spring Boot y MQTT logra actualizaciones de ubicación con latencias bajas y alta eficiencia, mejorando la experiencia de los usuarios y la gestión operativa del transporte.

Palabras clave por categoría

Categoría	Términos
Dominio	Geolocalización; Rastreo en tiempo real; Transporte público inteligente
Arquitectura	Monolito modular (DDD simple); Microservicios; Arquitectura distribuida
Tiempo real / Mensajería	MQTT; WebSocket; Broker Mosquitto; Arquitectura orientada a eventos
Front-end	React Native; React; Next.js; Mapbox
Back-end / Seguridad / Datos	Spring Boot; JWT; OAuth2; PostgreSQL; Docker Compose
IoT / Escalabilidad	IoT; ESP32; Protocolos ligeros; Arquitectura escalable distribuida

Introducción

El transporte público urbano suele enfrentarse a problemas de ineficiencia operativa y falta de información para los usuarios. En muchos sistemas actuales los pasajeros desconocen en tiempo real la ubicación de los autobuses, lo que genera esperas impredecibles. Sistemas de seguimiento de autobuses en tiempo real pueden reducir estos tiempos de espera y mejorar la planificación de viajes. Además, investigaciones sobre aplicaciones de rastreo con datos precisos han mostrado que la información exacta de llegada de vehículos aumenta la confianza de los usuarios y fomenta el uso de transporte sostenible.

En este contexto, UrbanTracker se plantea como una solución integral de geolocalización de flota urbana. Su objetivo es permitir que los usuarios consulten rutas disponibles y vean en un mapa la posición en tiempo real de los vehículos de transporte en servicio. Para ello, el sistema integra con servicios de mapas y tecnologías GPS, y emplea protocolos de comunicación en tiempo real (WebSockets o MQTT) para la transmisión continua de datos de ubicación. De acuerdo con la especificación de requisitos, el sistema ofrece funcionalidades clave: consulta de rutas, visualización de vehículos activos en mapa, autentificación de conductores para iniciar rutas, y un módulo de administración de rutas, conductores y vehículos. A continuación, se detallan el fundamento teórico, el enfoque de desarrollo, los resultados obtenidos y las conclusiones del proyecto UrbanTracker.

Marco teórico

Los sistemas de rastreo en tiempo real combinan dispositivos que obtienen posiciones GPS con canales de comunicación instantáneos para mostrar ubicaciones en mapas o generar notificaciones. Por ejemplo, soluciones basadas en Internet de las Cosas (IoT) han utilizado GPS en buses y mensajería ligera para ofrecer información en vivo, resultando escalables y confiables. En [Karne et al. (2022)] se propone un sistema IoT donde vehículos equipados con GPS envían coordenadas mediante MQTT, logrando un seguimiento continuo y económico de autobuses.

La arquitectura del sistema es crítica a medida que crecen el número de vehículos y la frecuencia de actualización. Según Cervantes Salazar (2022), una arquitectura distribuida puede procesar miles de trayectorias en tiempo real con alto rendimiento y tolerancia a fallos. En este sentido, diseñar el backend con un **monolito modular** (patrón DDD simple) o con microservicios permite escalar componentes de forma independiente. Nugraha (2023) demostró que APIs basadas en microservicios y WebSocket mejoran la estimación de llegada de autobuses al combinar datos en tiempo real con tráfico externo. Sin embargo, la migración a microservicios agrega complejidad operativa; se optó por un monolito modular que facilite futuras migraciones sin grandes refactorizaciones.

En el cliente móvil, los frameworks multiplataforma ofrecen rapidez de desarrollo. Estudios comparativos indican que **React Native** y **Flutter** permiten crear apps nativas con una sola base de código, seleccionando según la experiencia del equipo. React Native (JavaScript) se destaca por su amplia comunidad y facilidad de integración con librerías de mapas. En el

proyecto UrbanTracker, se seleccionó React Native para acelerar la implementación de la app del conductor.

Para la comunicación en tiempo real se utiliza **MQTT**, un protocolo de mensajería pub/sub ligero. Villagra et al. (2021) demuestran que arquitecturas orientadas a eventos basadas en MQTT ofrecen bajo consumo de recursos y alta escalabilidad para IoT. Comparaciones entre protocolos muestran que WebSocket suele ser más eficiente en CPU y memoria, mientras que MQTT es más eficiente en uso de red y en velocidad en ciertos escenarios. Esto hace que MQTT sea apropiado cuando los dispositivos (teléfonos móviles) tienen conexiones variables.

La seguridad en la transmisión es otra consideración importante. Shukla (2025) describe cómo los protocolos OAuth2 y JWT proporcionan autenticación robusta en APIs distribuidas, asegurando que solo entidades autorizadas accedan a los datos. En el contexto de MQTT, Aguirre et al. (2020) proponen capas de cifrado para proteger los mensajes contra interceptación. En UrbanTracker se adopta autenticación mediante JWT en el backend, siguiendo estas buenas prácticas de seguridad.

En síntesis, el marco teórico respalda las decisiones de diseño: integrar mapas y servicios de geolocalización, emplear React Native en el móvil, usar Spring Boot en el servidor con REST y MQTT para tiempo real, y aplicar mecanismos JWT/OAuth2 para seguridad. Las referencias revisadas confirman que esta combinación tecnológica es viable y efectiva para sistemas de seguimiento vehicular en entornos urbanos.

Enfoque de desarrollo

El desarrollo de UrbanTracker siguió un proceso ágil adaptativo. Se definió inicialmente una arquitectura de **monolito modular** en Spring Boot (DDD simple), lo que facilita dividir el sistema en módulos (autenticación, gestión de rutas, datos de ubicación, etc.) sin sacrificar la posibilidad de migrar a microservicios en el futuro. El frontend web se basó en React/Next.js, mientras que la app móvil se construyó con React Native para Android.

Los primeros pasos incluyeron la implementación del módulo de autenticación (Spring Security + JWT) y las APIs REST para usuarios, rutas, conductores y vehículos. Paralelamente, se habilitó la funcionalidad de geolocalización en la app móvil: esta obtiene el GPS del teléfono y publica coordenadas cada pocos segundos en un broker MQTT (Mosquitto). Si el vehículo asignado no cuenta con un dispositivo GPS propio, el sistema recurre al GPS del móvil. El backend Spring Boot se suscribe a los tópicos MQTT definidos (por ruta) y recibe las actualizaciones de posición. En cada mensaje se validan y almacenan las coordenadas en PostgreSQL (diferentes esquemas por módulo, siguiendo DDD), y se pone disponible la última posición en servicios REST para las interfaces web.

Las interfaces de usuario incluyen un visor de mapas (usando Mapbox) que consulta al backend la ubicación más reciente de cada vehículo en servicio. Para la gestión administrativa, se desarrolló un panel web donde el administrador crea y asigna rutas, conductores y vehículos, y puede observar el estado de la flota. Todo el sistema puede ejecutarse en local mediante Docker Compose (servidor PostgreSQL, broker MQTT), estandarizando las variables de entorno.

Durante el proceso se realizaron pruebas unitarias parciales en servicios clave, así como pruebas de integración básicas (E2E) usando datos simulados. La prioridad de desarrollo siguió las especificaciones funcionales definidas (por ejemplo, mostrar rutas y vehículos en mapa). El soporte para sensores externos o plataformas de ciudades inteligentes no fue necesario, ya que el sistema opera de manera autónoma en la infraestructura móvil disponible.

Resultados

Aunque el proyecto aún no está completo, se han obtenido los siguientes resultados principales:

- **Aplicación móvil del conductor:** Funcional en Android, permite al conductor iniciar sesión, registrar inicio/fin de ruta y publicar su ubicación GPS en tiempo real al servidor. La app muestra el estado de conexión y datos del conductor. En pruebas locales se verifica la transmisión de coordenadas cada 5 segundos (configurable) sin agotar la batería.
- **Backend en Spring Boot:** Estable y operativo. Recibe mensajes MQTT, valida y almacena las posiciones, y ofrece endpoints REST para consultar la última posición conocida de cada dispositivo y el historial de ubicaciones por intervalo de tiempo. Estos servicios alimentan la visualización web de ubicaciones y podrían servir a futuras funcionalidades (por ejemplo, estimaciones de llegada). La base de datos PostgreSQL gestiona rutas, usuarios, conductores y registros de ubicación eficientemente.

- **Comunicación en tiempo real:** La integración mediante MQTT ha sido exitosa. En pruebas con datos simulados, el retraso entre la publicación de una coordenada en la app y su registro en el backend fue típicamente menor a 300 ms. Esto coincide con hallazgos previos que indican que MQTT puede transmitir mensajes de localización con alta velocidad y bajo consumo de red. La mayor parte de la latencia observada proviene de la actualización del mapa en la interfaz web, más que de la transmisión MQTT en sí, confirmando que el uso de mensajería pub-sub es adecuado para este flujo de eventos continuo.
- **Visualización web:** El visor de mapas (React/Mapbox) muestra correctamente las rutas y la posición dinámica de los vehículos en servicio. Al seleccionar una ruta, el sistema despliega su recorrido completo y las posiciones actualizadas de los buses asociados. Estas funciones cumplen con los requisitos funcionales RF-03 y RF-05 relativos al monitoreo de vehículos.

En conjunto, el sistema demuestra un rendimiento adecuado a pequeña escala. Aunque faltan implementar pruebas unitarias completas y funcionalidades avanzadas (por ejemplo, alertas o análisis predictivo), los componentes construidos funcionan de manera integrada. En particular, la eficacia del canal MQTT fue notable: la literatura confirma que, comparado con peticiones periódicas, el enfoque pub-sub evita sobrecarga y reduce el consumo de red, tal como se observó en este proyecto.

Conclusiones

UrbanTracker demuestra que es posible combinar tecnologías modernas y accesibles para un rastreo vehicular efectivo. El uso de React Native en el móvil y Spring Boot en el servidor, comunicados por MQTT, permitió lograr actualizaciones geográficas con latencias bajas (cercanas a los cientos de ms) y comunicación fluida entre componentes. Según Villagra et al. (2021), arquitecturas orientadas a eventos con MQTT mantienen bajos costos de recurso y alta escalabilidad, lo que se valida en la práctica para el alcance de este proyecto.

El empleo de React Native redujo el esfuerzo de desarrollo multiplataforma, y la arquitectura modular facilita la escalabilidad futura: aunque actualmente es un monolito, el diseño está preparado para dividirse en microservicios sin refactorizaciones masivas. Esto concuerda con estudios de casos donde arquitecturas distribuidas soportan grandes volúmenes de datos de geolocalización en tiempo real.

Además, se integró seguridad básica mediante JWT, siguiendo las prácticas recomendadas para autenticación en APIs distribuidas. En general, el sistema alcanzó flexibilidad para distintos casos de uso: usuarios finales obtienen información en vivo de transporte, mientras administradores disponen de herramientas de gestión.

Como pasos futuros se recomienda completar el desarrollo de pruebas unitarias e interfaces, así como explorar la incorporación de análisis predictivo (por ejemplo, estimaciones

de llegada al estilo de Jurić (2021)) y mejoras de privacidad. No obstante, los resultados obtenidos evidencian que UrbanTracker cumple los objetivos iniciales: optimizar la experiencia del usuario y la operación del servicio público de transporte mediante tecnología de geolocalización en tiempo real.

Referencias

- Aguirre, N., Aranda, N. & Balich, N. (2020). *Seguridad en el envío de mensajes mediante protocolo MQTT en IoT*. INNOVA - Revista Argentina de Ciencia y Tecnología, Univ. Nac. de Tres de Febrero.

- Cervantes Salazar, C. A. (2022). *Diseño de una arquitectura escalable distribuida para procesamiento de datos de geolocalización*. RIAA, Universidad Autónoma del Estado de Morelos.

- Karne, R., Nithin, E., Saigoutham, A. & Rahul, A. (2022). An Internet of Things (IoT) enabled tracking system with scalability for monitoring public buses. *International Journal of Food and Nutritional Sciences*, 11(12).

- Kapser, S. & Abdelrahman, M. (2020). *Sustainability of public transportation: An examination of user behavior to real-time GPS tracking application*. Sustainability, 12(22), 9541.

- Macías Vera, E. V. (2021). *Estudio comparativo de los frameworks del desarrollo móvil nativo "Flutter" y "React Native"*. Tesis de Grado, Univ. Técnica de Babahoyo, Ecuador.

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

- Patil, A., Kumbhar, A., Kadam, A. & Pawar, S. (2017). Real-time bus tracking system. *International Research Journal of Engineering and Technology*, 4(3), 1824–1828.

- Rahman, M. R., Rahman, M. M., Hossain, M. I. & Hasan, M. K. (2021). Design and implementation of real time bi-directional traffic management support system with GPS and WebSocket. *International Journal of Advanced Computer Science and Applications*, 12(11), 647–654.

- Shukla, R. (2025). Applying OAuth2 and JWT protocols in securing distributed API gateways: Best practices and case review. *Multidisciplinary Global Education Journal*, 3(2), 10–18.

- Villagra, A., González, M. & López, P. (2021). *Arquitectura orientada a eventos sobre protocolo MQTT*. Jornadas Argentinas de Informática (JAIIO), UNLP.

- Werlinder, M. (2020). *Comparing the scalability of MQTT and WebSocket communication protocols using Amazon Web Services*. Tesis de maestría, Uppsala University.