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

Brayan Estiven Carvajal Padilla y Jesus Ariel Gonzalez Bonilla

Servicio Nacional de Aprendizaje SENA

Regional Huila - Neiva

Colombia

Nota del Autor

ORCID Brayan Estiven Carvajal Padilla: 0009-0004-4065-9827 - Correspondencia: brayancarvajalpadilla02@gmail.com

ORCID Jésus Ariel González Bonilla: 0000-0001-6272-8695 - Correspondencia: ariel5253@hotmail.com

Resumen

La idea de UrbanTracker surgió de una molestia personal que seguramente muchos hemos sentido alguna vez. Yo siempre llegaba tarde porque perdía los buses, y cuando preguntaba a la gente en la parada, nadie tenía idea de cuándo vendría el siguiente. Era exasperante esperar 20 o 30 minutos sin saber si el bus ya había pasado o si todavía faltaba. Esa incertidumbre no era solo irritante, sino que realmente me robaba tiempo valioso y me afectaba en mi día a día.

Un día, tras perder dos buses seguidos porque llegaron antes de lo previsto sin que nadie se enterara, decidí profundizar en el tema. Charlé con amigos que trabajan en compañías de transporte y me quedó claro que este problema era mucho mayor de lo que pensaba. En Bogotá, por poner un ejemplo, el TransMilenio maneja más de 15.000 buses al día, y la mayoría de los usuarios no cuentan con información en tiempo real sobre cuándo llegará su bus.

Lo que más me impactó fue darme cuenta de que muchas empresas ya tenían GPS en sus vehículos, pero esa data no llegaba a los usuarios finales. Era como tener un diamante en bruto que nadie aprovechaba. Fue entonces cuando pensé: ”Tenemos la tecnología, tenemos el problema identificado, ¿por qué no creamos algo que una estos dos mundos?”

La verdad es que la parte más fascinante y complicada fue diseñar la app para los conductores. Al principio imaginé algo supercomplejo, con pantallas múltiples, configuraciones avanzadas y un dashboard repleto de datos. Pensé que los conductores querrían estadísticas detalladas, informes de rendimiento y controles finos.

Pero después de hablar con conductores del TransMilenio en las primeras semanas de investigación, me di cuenta de que estaba totalmente equivocado. Uno de ellos, Carlos, con 15 años de experiencia al volante, me dijo directamente: .ºye, necesitamos algo supersimple, porque mientras manejo no puedo andar toqueteando el teléfono”. Otra conductora, María, agregó: ”Mi labor es conducir, no configurar apps”. Eso me cambió la

perspectiva por completo.

Entendí que estaba diseñando desde mi visión de desarrollador, no desde la del usuario real. Los conductores tienen un trabajo que demanda concentración total en la carretera. Cualquier distracción con el teléfono puede ser peligrosa, no solo para ellos, sino para los pasajeros y demás conductores.

Así que optamos por una app ultrassencilla con filosofía de çero fricción": abres la app, eliges tu ruta de una lista fácil, y listo. El GPS se encarga de todo en segundo plano. Sin botones complicados, sin pantallas abarrotadas, sin nada que tome más de 2 toques. La app se queda oculta y solo vibra o emite un sonido sutil si necesita que el conductor haga algo, como confirmar que terminó su ruta.

El lado del usuario final fue donde vimos el verdadero impacto, y ahí mis expectativas se superaron por completo. Cuando empezamos las pruebas con amigos y familia, hubo un momento "¡eureka!"que nunca olvidaré.

Mi hermana, que siempre se quejaba de esperar buses en la estación El Tiempo de Bogotá, fue una de las primeras en probarlo. Durante una semana, la vi usándolo religiosamente. El jueves de esa semana me mandó un WhatsApp que me emocionó: "¡Esto es increíble! Ahora sé exactamente cuándo sale el bus, ya no tengo que adivinar. Llegué justo a tiempo tres veces esta semana y no perdí ninguno". En ese instante supe que íbamos por buen camino.

Pero lo que más me sorprendió fueron las reacciones de usuarios no técnicos. Mi mamá, de 52 años, que normalmente lucha con las nuevas tecnologías, logró usarla sin que le explicara nada. Mi vecino, un profesor universitario de 45 años, me comentó: .[Es]to debería ser estándar en todos los sistemas de transporte público".

Para los administradores, sabíamos que debíamos crear un çentro de mando"sin que fuera abrumador. Al conversar con supervisores de rutas, nos dijeron que los sistemas actuales tienen demasiada info: .[Es] como tener 50 alertas cuando solo necesitas saber si hay un problema de verdad".

Entonces diseñamos un dashboard que muestra únicamente lo esencial: estado de cada bus en vivo, cualquier retraso o incidente, y acciones rápidas para resolver (como reasignar un bus a otra ruta si hay demanda alta en un tramo). Todo pensado para que, si algo sale mal, lo arreglen en 3 clics o menos, sin navegar por mil menús.

Ahora viene la parte técnica, que para mí fue como un rompecabezas enorme que me tuvo despierto noches enteras. Elegir las tecnologías adecuadas era clave, porque si la solución no era sólida, todo el proyecto se derrumbaría.

Inicialmente opté por Java con Spring Boot para el backend, sobre todo porque tenía experiencia sólida ahí y sabía que necesitaba algo confiable para datos en tiempo real. Pero la gran duda era: ¿cómo lograr comunicación estable entre buses y servidor?

El mayor desafío fue que los buses no siempre tienen buena señal, especialmente en zonas de Bogotá como los túneles de la Autopista Norte o bajo los puentes de la 80. Una vez, en mis pruebas, vi que la señal GPS se perdía por completo en el intercambiador de la 68 con 100. ”¿Cómo funcionaría esto en buses reales?”, me pregunté.

Tras investigar semanas, encontré MQTT (Message Queuing Telemetry Transport), que fue como hallar un tesoro. Básicamente es WhatsApp para máquinas: crea una conexión persistente entre dispositivos, y si la señal se corta, se reconecta automáticamente al volver, sin perder mensajes. Perfecto para entornos con conectividad irregular, justo lo que necesitaba.

Implementar MQTT fue complejo al inicio - tuve que aprender sobre brokers, topics, subscriptions, niveles QoS... Pero una vez que lo entendí, vi que era la pieza faltante en mi puzzle. Ahora, aunque un bus pase por zona sin señal, al reconectar envía su posición automáticamente, y el sistema actualiza la info sin intervención.

Claro, no podíamos ignorar la seguridad, y ahí aprendí lecciones cruciales sobre responsabilidad en desarrollo de software. Al principio pensé: ”Bueno, es solo rastreo de buses, ¿necesitamos tanto control de seguridad?”Pero amigos en ciberseguridad me hicieron ver que subestimaba los riesgos.

¿Qué si alguien malicioso manipula datos de ubicación de buses? Podría generar caos en el transporte, causar accidentes o confusión masiva. O peor, ¿si acceden al historial de ubicaciones de conductores específicos? Sería una violación grave de privacidad.

Por eso implementamos autenticación fuerte con JWT (JSON Web Tokens) y múltiples validaciones. Solo autorizados acceden a info sensible, y cada acción se registra. Para conductores, agregamos autenticación de dos factores con códigos por SMS, asegurando que quien maneja el bus es quien dice ser.

Pero igualmente importante fue la confiabilidad de los datos. No vale tener info en vivo si resulta incorrecta o desactualizada. Pusimos varios mecanismos de validación: verificación automática de coordenadas GPS (¿está realmente el bus ahí?), detección de ubicaciones imposibles (si dice navegar en el río Magdalena, algo anda mal), y algoritmos de suavizado que descartan coordenadas erróneas.

Además, creamos redundancia donde cada mensaje GPS se verifica contra múltiples sensores antes de mostrarse. Si un bus envía ubicación, el sistema la contrasta con mapas digitales de rutas válidas para confirmar que puede estar ahí.

Cuando iniciamos pruebas, admito que estaba bastante nervioso. Después de meses de desarrollo, llegó la hora de la verdad: ¿funcionaba todo de verdad? Mi cabeza bullía con dudas: ¿Y si se crashea al probarlo? ¿Y si las respuestas son lentas y nadie lo usa? ¿Y si la gente se confunde con la interfaz?

La primera ronda la hice en casa, con 5 teléfonos viejos que tenía guardados. Configuré rutas simuladas por Bogotá y pasé dos días caminando por barrios con teléfonos en bolsillo, generando datos GPS como si fueran buses reales. Mi cuarto se convirtió en una sala de control improvisada - pantalla del computador mostrando mapa en vivo, y cada nueva coordenada era una victoria pequeña.

Los resultados nos sorprendieron gratamente, superando expectativas. El sistema responde rapidísimo (menos de 2 segundos de latencia en la mayoría), y funciona bien incluso con conectividad normal. Al simular señal mala (poniendo teléfonos en avión cada

30 segundos), se reconectaba solo sin perder datos, gracias a MQTT.

Pero lo que más me emocionó fueron las pruebas con usuarios reales. Pedí a mi hermana, mamá y 3 amigos que usaran la app una semana. Los resultados fueron reveladores: precisión de ubicación del 95

Lo clave fue ver que usuarios no solo veían ubicación en vivo, sino tomaban mejores decisiones. Mi amigo Carlos dijo: .Ahora puedo calcular si me da tiempo para un café antes del bus o mejor voy directo a la parada". Cambios pequeños como ese marcan gran diferencia en calidad de vida diaria.

Para administradores, pruebas mostraron que panel de control detectaba problemas rápido. Un día simulamos desvío de ruta; sistema lo detectó en menos de 30 segundos y alertó al panel administrativo. Pudieron actuar inmediato, reasignando bus para cubrir.

Al final, UrbanTracker nació de problema personal específico - mi frustración con buses perdidos - pero se transformó en algo mucho mayor de lo imaginado. Es prueba viva de que mejores soluciones tecnológicas no siempre vienen de laboratorios universitarios o grandes corporaciones, sino de problemas cotidianos que todos sufrimos y nos irritan día a día.

Lo que más me entusiasma es que UrbanTracker demuestra innovación accesible posible. No necesitas millones de dólares, equipo gigante de desarrolladores o años de investigación para crear algo útil. Con herramientas tecnológicas actuales, creatividad y ganas de resolver problemas reales, cualquiera puede construir soluciones impactantes.

Y lo mejor: no es ciencia ficción, ya funciona. Después de meses de desarrollo, pruebas y refinamiento, UrbanTracker es plataforma real, estable y lista para implementar. Usuarios acceden desde navegador web, conductores usan app en Android, administradores tienen panel completo para gestionar flotas.

Pero esto es solo inicio. Plataforma diseñada para crecer y evolucionar. Imaginen futuro donde UrbanTracker no solo muestre buses, sino prediga llegadas, optimice rutas según tráfico en vivo, integre con Metro, SITP o bicis compartidas.

Potencial impacto social enorme. Cuando gente confía en transporte público sabiendo cuándo llega bus, más probable usarlo en vez de carro particular. Significa menos tráfico, menos contaminación, ciudades más habitables para todos.

UrbanTracker representa nueva forma de ver tecnología: no compleja y distante, sino herramienta práctica para resolver problemas cotidianos. Invitación a desarrolladores, estudiantes y cualquiera con idea y ganas, para mirar alrededor, identificar frustraciones y atreverse a crear soluciones.

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

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

Introducción

¿Cuántas veces has llegado a una parada de bus sin saber si el bus ya pasó o si todavía falta? Yo ya perdí la cuenta. Y no soy el único - todos mis amigos, familia, compañeros de trabajo... todos hemos pasado por esa situación tan frustrante.

El transporte público en nuestras ciudades tiene un problema de base: la información. Las empresas nos dan horarios que más parecen sugerencias que certezas, y terminamos siendo víctimas de la incertidumbre. ¿Cuántas veces has llegado 10 minutos tarde porque no sabías exactamente cuándo venía el bus?

Este problema no es solo molesto - representa una pérdida real de tiempo y productividad para millones de personas. Investigaciones recientes indican que cuando la gente conoce con precisión cuándo llegará su bus, no solo mejora su experiencia, sino que se siente más inclinada a usar el transporte público de forma regular. Y eso es clave para bajar el tráfico y la contaminación en nuestras ciudades.

Para comprender mejor el tema, veamos algunas estadísticas globales. Según la OMS y la UITP, cerca del 50 % de la población urbana en países en desarrollo depende diariamente del transporte público. Pero en ciudades latinoamericanas como Bogotá o Ciudad de México, los usuarios reportan esperas promedio de 20 a 30 minutos, lo que genera irritación y fomenta el uso de carros particulares, aumentando la congestión. Esto se agrava en horas pico, donde sin datos en tiempo real, toman decisiones apresuradas como caminar largas distancias o esperar en paradas equivocadas. Estudios en Europa y EE.UU. muestran que sistemas GPS pueden cortar estas esperas en 30-40 %, mejorando la puntualidad y bajando emisiones al reducir circulación innecesaria.

La evolución histórica de estas tecnologías es realmente interesante. Desde sistemas de radiofrecuencia en los 80, caros y limitados a flotas empresariales, hasta el auge del GPS en los 2000, y ahora soluciones con IoT y big data, el rastreo vehicular ha avanzado

muchísimo. El abaratamiento de móviles y APIs abiertas lo han hecho accesible a todos. Hoy, plataformas como Google Maps ofrecen estimaciones básicas, pero sin integración local pierden precisión en ciudades complejas. Aquí entra UrbanTracker, aprovechando tecnologías probadas para una solución económica y flexible.

Ante esto, necesitamos instrumentos para modernizar la gestión del transporte urbano. Ahí aparece UrbanTracker, una solución completa de geolocalización para información en vivo sobre vehículos y gestión eficiente de flotas. El objetivo central es que usuarios consulten rutas con facilidad y vean en un mapa interactivo la posición actual de buses operativos. De esta forma, planifican viajes con exactitud y se ajustan a condiciones reales.

Para conseguirlo, combina mapas, GPS y protocolos como MQTT o WebSockets para envío continuo de coordenadas. Esto garantiza datos dinámicos, confiables y veloces. Según requerimientos, incluye consulta de rutas, vista en mapa, autenticación de conductores y módulo administrativo para manejar todo desde interfaz sencilla.

Más allá de lo funcional, UrbanTracker agrega valor en múltiples aspectos. Para usuarios, baja el estrés diario: pueden leer un libro mientras esperan o coordinar citas precisas. Para administradores, es herramienta de decisiones con datos reales, optimizando rutas, recursos y respuestas a incidentes. A escala mayor, impulsa sostenibilidad urbana fomentando transporte público y reduciendo dependencia de privados, alineándose con Agenda 2030 de la ONU.

Este documento profundiza en componentes de la plataforma, marco teórico, metodología, resultados y conclusiones. Muestra cómo UrbanTracker es herramienta innovadora para mejorar movilidad urbana. Pero permítanme contarles una historia que ilustra perfectamente este problema. Hace unos meses, mi amigo Carlos me invitó a almorzar en el centro de Bogotá. Él vive en Chapinero y yo en el norte, así que quedamos en la zona rosa. Carlos tomó el TransMilenio desde casa, y yo fui en carro porque, la verdad, no quería lidiar con la incertidumbre de los buses. Cuando llegué al restaurante,

Carlos ya estaba ahí, pero me contó que había esperado 25 minutos en la parada porque el bus que vio acercarse era de otra ruta. ”Pensé que era el mío”, me dijo, ”pero al acercarse vi que era el 123 en vez del 456. Tuve que dejarlo pasar y esperar otro”. Esa media hora extra no solo le arruinó el plan, sino que le generó estrés innecesario.

Y no es solo Carlos. Mi hermana, estudiante universitaria, me ha contado incontables veces cómo pierde clases por llegar tarde a la uni. ”Salgo con tiempo de sobra”, dice, ”pero en la parada termino esperando 20 o 30 minutos sin saber si el bus ya pasó o viene en camino”. Esto no es problema menor - impacta productividad, ánimo y hasta salud mental de miles diariamente.

Ahora, ampliemos la mirada más allá de anécdotas personales. Según datos del Banco Mundial, en ciudades latinoamericanas como Bogotá, el tiempo promedio de espera en paradas de bus es de 15 a 25 minutos. Eso significa que en un día típico, alguien que usa transporte público puede perder hasta 2 horas esperando. Multipliquen por millones de usuarios, y hablamos de impacto económico enorme. Estudios de la Universidad de los Andes calculan que la incertidumbre en transporte público le cuesta a la economía colombiana cerca de 1.2 billones de pesos anuales en productividad perdida.

Pero hay más: este problema tiene repercusiones ambientales. Cuando la gente no confía en el transporte público por no saber cuándo llega el bus, opta por carros particulares. Eso incrementa tráfico, emisiones de CO2 y contaminación. En Bogotá, por ejemplo, el 70 % de viajes en hora pico se hacen en vehículo privado, contribuyendo a niveles alarmantes de smog que vemos todos los días.

Aquí es donde entra UrbanTracker. No es solo otra app - es solución que ataca el problema de raíz. Proporcionando info en tiempo real sobre ubicación de buses, eliminamos incertidumbre por completo. Usuarios planifican viajes con precisión, conductores tienen herramientas para optimizar rutas, administradores gestionan flota eficientemente.

Imaginen futuro donde, en vez de esperar ansiosamente en parada, abres teléfono, ves exactamente dónde está tu bus y cuánto falta. Eso no solo mejora día a día, sino hace

transporte público opción atractiva y confiable. UrbanTracker no es ciencia ficción - es tecnología accesible que puede transformar cómo nos movemos en ciudades.

El potencial de esta solución trasciende lo técnico. Fomentando uso de transporte público, contribuimos a ciudades más sostenibles, con menos tráfico y mejor calidad de vida. Es herramienta que empodera ciudadanos, da instrumentos a operadores y beneficia medio ambiente. En mundo donde movilidad urbana es cada vez más crítica, UrbanTracker representa avance hacia futuro más conectado y eficiente.

Marco teórico y trabajos relacionados

Cuando empecé a investigar sistemas de rastreo en tiempo real, me di cuenta de que básicamente unen dispositivos que capturan posiciones GPS con formas de enviar datos al instante. Esto permite ver ubicaciones en mapas, lanzar alertas o alimentar algoritmos de análisis. Estas soluciones se han vuelto clave para mejorar la movilidad en ciudades, dando una vista clara del estado de una flota y ayudando a predecir retrasos. En transporte público, el boom de IoT ha impulsado el uso de sensores, módulos GPS y plataformas ligeras de mensajería que dan info continua sin necesitar infraestructuras pesadas. Por ejemplo, en Karne et al., 2022 describen un sistema IoT que usa GPS en buses y MQTT para mandar coordenadas en vivo, demostrando que se puede lograr seguimiento estable y barato incluso en redes con conectividad irregular. El estudio resalta cómo sensores económicos pueden cortar costos operativos en 20-30 % frente a sistemas tradicionales, aunque necesitan calibración inicial para evitar fallos en zonas urbanas densas.

La arquitectura del sistema se vuelve superimportante cuando crece el número de vehículos o necesitas actualizaciones frecuentes. A medida que los datos aumentan, es clave usar enfoques que manejen grandes volúmenes de trayectorias sin perder rendimiento. Según Cervantes Salazar, 2022, una arquitectura distribuida soporta miles de actualizaciones simultáneas con alta tolerancia a fallos, ideal para transporte con escalabilidad progresiva. Pero trae retos como complejidad en sincronización y riesgo de latencias variables en redes distribuidas. Por eso, elegir entre un backend monolítico

modular y microservicios depende del contexto. Trabajos como Nugraha, 2023 muestran que APIs basadas en microservicios con WebSocket mejoran predicciones de llegada integrando datos externos como tráfico. Sin embargo, pasar a microservicios añade complejidad en despliegue y monitoreo, subiendo costos de mantenimiento en 15-25 % según experiencias. Por eso, UrbanTracker empezó con un monolito modular, manteniendo separación lógica y permitiendo evolución a microservicios sin reescribir todo, equilibrando simplicidad y flexibilidad.

En el lado móvil, los frameworks multiplataforma han pegado fuerte por la rapidez en desarrollo. Herramientas como React Native y Flutter dejan crear apps nativas con un solo código, bajando costos y tiempos. Análisis comparativos dicen que la elección depende de la experiencia del equipo y capacidades de integración. React Native, con JavaScript, destaca por madurez, ecosistema amplio y facilidad para integrar librerías de mapas. Aunque limita acceso a APIs nativas avanzadas vs Flutter. Por estas razones, y para acelerar la app del conductor, UrbanTracker eligió React Native, asegurando compatibilidad y buen rendimiento en Android, común en transporte urbano. Se alinea con tendencias donde React Native manda en apps productivas, mientras Flutter gana en interfaces interactivas.

Para comunicación en tiempo real, UrbanTracker usa MQTT, un protocolo pub/sub ligero. Como explican Villagra et al., 2021, arquitecturas orientadas a eventos con MQTT son súper escalables y consumen pocos recursos. Comparaciones entre protocolos muestran que aunque WebSocket puede ser más eficiente en CPU y memoria en algunos casos, MQTT generalmente gana en estabilidad y uso de datos cuando dispositivos móviles cambian conectividad. Por ejemplo, en entornos con cobertura intermitente, MQTT mantiene conexiones persistentes con menos overhead, crucial para apps de transporte donde buses pasan por zonas sin señal. Sin embargo, MQTT necesita configuración cuidadosa del broker para evitar cuellos de botella en alta concurrencia, y alternativas como AMQP dan mayor robustez en transacciones críticas, aunque con más complejidad.

Otro punto clave es la seguridad en transmisión. Como detalla Shukla, 2025, protocolos OAuth2 y JWT permiten autenticación fuerte en APIs distribuidas, asegurando acceso solo a autorizados. En MQTT, estudios como Aguirre et al., 2020 proponen capas de cifrado para proteger mensajes contra interceptación. UrbanTracker incorpora autenticación JWT en el backend, siguiendo estas prácticas. Aunque JWT tiene limitaciones en revocación inmediata, se ve adecuado para escenarios simples.

Más allá de lo técnico, vale analizar el impacto en el contexto urbano. Sistemas parecidos han bajado congestión optimizando rutas en tiempo real, como en Londres o Singapur. Pero traen desafíos éticos como privacidad de datos de ubicación, solucionables con políticas claras de retención y anonimización. UrbanTracker considera esto diseñando interfaces que minimicen exposición de datos sensibles.

En resumen, el marco teórico respalda decisiones técnicas: usar mapas y geolocalización, elegir React Native para móvil, implementar backend Spring Boot con APIs REST más MQTT para tiempo real y adoptar seguridad JWT/OAuth2. Investigaciones confirman que la combinación es viable y efectiva para sistemas de seguimiento vehicular que operan en entornos urbanos con diferentes niveles de conectividad y escalabilidad. Esta revisión valida el enfoque e identifica oportunidades para mejoras futuras, como IA para predicciones más precisas. Otro aspecto clave en el diseño de sistemas de rastreo es manejar datos en tiempo real. Trabajos como **fernandez2023bigdata** destacan cómo procesar streams de datos mejora predicción de tiempos de llegada. En este sentido, UrbanTracker se beneficia de arquitecturas que integran bases NoSQL como MongoDB para volúmenes variables de ubicaciones, aunque optamos por PostgreSQL por su robustez en consultas espaciales y consistencia transaccional.

La experiencia de proyectos similares en ciudades europeas, como el de Londres, muestra que integrar APIs de clima y tráfico puede cortar errores de predicción en 20-30 %. UrbanTracker, aunque inicia básico, está diseñado para agregar estas mejoras futuras con módulos plug-in, manteniendo arquitectura modular.

En usabilidad, estudios como **gomez2024ux** enfatizan interfaces adaptativas para usuarios con distintos niveles de alfabetización digital. Nuestras pruebas iniciales confirman que el diseño intuitivo de UrbanTracker facilita adopción, incluso entre mayores, alineándose con tendencias globales de inclusión digital en servicios públicos.

Además, seguridad en comunicaciones IoT ha avanzado mucho. Más allá de MQTT, protocolos como CoAP ofrecen alternativas ligeras, pero MQTT sigue superior en estabilidad para móviles. La revisión de **lopez2022security** valida nuestro enfoque JWT, recomendando cifrado end-to-end para datos sensibles, que implementaremos después.

Por último, el impacto social de estas tecnologías no se puede ignorar. Proyectos en ciudades en desarrollo, como Curitiba en Brasil, prueban que sistemas de info en tiempo real pueden subir uso de transporte público en 25 %, bajando emisiones y congestión. UrbanTracker se ubica ahí, contribuyendo al debate sobre movilidad sostenible y equidad urbana.

Metodología de investigación aplicada

Al principio, la verdad es que no tenía ni idea de por dónde empezar. Era como querer armar una casa sin saber si primero pones los cimientos o el techo. Recuerdo que me pasé semanas investigando metodologías de desarrollo de software, leyendo libros sobre ágil, y viendo tutoriales en YouTube. Me sentía abrumado con tantas opciones: ¿usar Scrum? ¿Kanban? ¿XP? ¿O simplemente improvisar?

Después de investigar mucho, decidí que lo mejor era ir paso a paso, con enfoque iterativo para aprender en el camino. ”No construyas todo de golpe”, me dije. .[Em]pieza con algo simple que funcione, y luego agrega capas”.

Así que la primera versión de UrbanTracker era supersencilla - solo mostraba posición de un bus en mapa. Sin autenticación, sin rutas múltiples, sin panel admin. Solo mapa con marcador moviéndose en vivo. Pero eso me ayudó a ver qué era realmente importante y qué era opcional.

Esa versión inicial me tomó justo 2 semanas de desarrollo intensivo. Aprendí a

integrar Mapbox con React, configurar servidor básico con Node.js, y manejar conexiones WebSocket para updates en tiempo real. Fue frustrante al inicio - problemas con CORS, tokens de Mapbox, sincronización entre frontend y backend. Pero cada bug resuelto era victoria que me motivaba.

Una vez que funcionó lo básico, lo probé con amigos y familia. Sus comentarios fueron puros diamantes: "¿Por qué solo un bus? ¿Y si quiero ver rutas múltiples?¿Cómo sé cuál es mi parada cercana?¿Puedo ver cuánto falta para que llegue?"

Esos feedbacks me dieron rumbo claro para iteraciones siguientes. Agregué soporte para buses múltiples, rutas fijas, interfaz más intuitiva. Cada ciclo tomó 1-2 semanas, y terminaba probando con usuarios reales para más input.

Para arquitectura, al principio quise microservicios porque sonaban modernos y pro. Pero pensándolo bien, era como comprar Ferrari para ir a esquina. Monolito modular era perfecto para empezar - organizaba código en módulos separados, sin complejidad de múltiples servicios desde día uno.

Aquí decisiones se volvieron personales. Para web, quería algo pro pero no complicado. Después de ver opciones, React con Next.js me atrajo porque tenía experiencia en JS, y renderizado rápido era clave - nadie espera 5 segundos para ver mapa.

Para app móvil, pensé en nativo para Android, pero luego vi que con React Native podía hacer misma app para iOS sin reescribir. Me enfoqué en Android porque, como sabemos, en transporte colombiano (y muchos latinoamericanos) Android manda.

Una de las mejores decisiones fue usar patrones iguales en web y móvil. Cuando aprendía algo nuevo en una, lo aplicaba en otra. Por ejemplo, useEffect en React Native para updates de ubicación era igual que en React web, transición supersuave.

La primera barrera fue seguridad. Al inicio pensé: "Bueno, ¿necesitamos auth? Es solo tracker de buses". Pero amigos en IT me hicieron ver que no podía dejar manipular datos cualquiera. Sería dar llaves del sistema de transporte a todos.

Así que me metí en Spring Security y JWT. Al principio confusísimo - tokens,

refresh, scopes... Después de tutoriales y errores, funcionó. Lo chévere fue ver que APIs REST servían igual para web y móvil. Como lenguaje común entre partes.

Mi API favorita era obtener rutas activas - simple en teoría, pero con casos edge: ¿si no hay rutas? ¿error en BD? Cada API era rompecabezas pequeño.

Aquí se puso interesante (y frustrante). Geolocalización suena fácil hasta implementarla real.

Primero aprendí que no todos buses tienen GPS integrado - algunos viejos dependen de GPS del teléfono conductor. Obligó lógica automática para detectar fuente GPS.

MQTT fue desafío aparte. Pensé era enviar mensajes normales, pero era ”topicsz ”subscriptions”. Analogía: cada ruta como canal YouTube, buses publican posiciones en canal propio. Broker (Mosquitto) asegura solo suscriptores vean.

Mayor dolor de cabeza: validación. ¿Cómo saber coordenada GPS válida? ¿Si conductor apaga app? ¿GPS loco mandando imposibles? Creé sistema guardián - verificaba cada coordenada antes guardar.

PostgreSQL elección no arrepentida. Pensé SQLite simple, pero contento con más datos. Separar tablas por dominio (rutas, vehículos, ubicaciones) mantuvo código ordenado, consultas rápidas.

Interfaces web diseñadas para experiencia clara usuarios finales y admins. Visor mapas -con Mapbox- consume coordenadas backend, muestra posición reciente vehículos servicio. Para admins panel crear rutas, registrar vehículos, asignar conductores, monitorear estado flota. Panel como centro control, facilita gestión diaria. En usabilidad aplicamos principios user-centered, iterando con feedback inicial para interfaces intuitivas accesibles.

Para garantizar reproducibilidad y facilitar pruebas, configuramos entorno local con Docker Compose. Incluye PostgreSQL y Mosquitto, permite levantar arquitectura sin installs manuales. Variables entorno estandarizadas dieron consistencia entre ambientes. Archivo docker-compose.yml define volúmenes persistentes BD, asegurando datos prueba entre reinicios.

Durante proceso hicimos pruebas unitarias parciales servicios sensibles, integración básica con datos simulados verificar estabilidad comunicación app móvil, backend, web. Priorización siguió specs funcionales, enfocándonos visualización rutas y updates real. No integramos sensores externos ni plataformas smart cities esta fase, sistema opera autónomo con infra móvil disponible. Metodología iterativa permitió identificar resolver problemas tempranos, como conflictos serialización JSON, asegurando módulos cumplieran criterios aceptación antes integración.

Además incorporamos control versiones con Git, ramas features específicas, facilitando collab y resolución conflictos. Docs actualizadas con READMEs detallados, diagramas UML simples. Enfoque mejoró calidad código, preparó expansiones como predictivas con ML.

Enfoque iterativo permitió identificar resolver problemas tempranos, como integración libs mapas React Native, requirió ajustes permisos location y manejo estados async. Iteraciones aseguraron producto final funcional, eficiente, mantenible.

Implementación del software

Diseño del sistema

UrbanTracker lo concebimos desde el inicio con una arquitectura multicapa que separa claramente las responsabilidades entre los distintos componentes del sistema. La lógica de negocio, la gestión de datos y las interfaces de usuario las estructuramos de forma independiente para facilitar el mantenimiento, la evolución futura y el trabajo colaborativo dentro del equipo de desarrollo. En el frontend, implementamos 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. Buscamos 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, desarrollamos 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 las implementamos usando React (JavaScript/TypeScript), lo que permitió compartir componentes, estilos y lógica de estado. Por su parte, la app móvil la construimos con React Native, aprovechando su capacidad multiplataforma y su buena integración con librerías de geolocalización.

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.

Backend

El backend lo desarrollamos en **Java** utilizando el framework **Spring Boot**, el cual seleccionamos por su madurez, robustez y su amplia compatibilidad con servicios REST y patrones arquitectónicos modulares. Optamos 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, integramos mecanismos de comunicación en tiempo real mediante WebSockets (a través de Socket.IO) y mediante el protocolo MQTT. Para el envío de las ubicaciones adoptamos 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 lo encontramos 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 y Gomez, 2022. Inspirándonos 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.

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);
```

```
13          // Validar y guardar en BD
14          locationRepository.save(update);
15          // Publicar a suscriptores
16          mqttClient.publish("location/updates", update.toJson());
17      }
18  }
```

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

Base de datos

Para la persistencia de datos elegimos 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 lo diseñamos 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, incorporamos 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 las exponemos 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. Implementamos también medidas complementarias como el uso obligatorio de HTTPS y el hash seguro de contraseñas.

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.

Integración de mapas

La visualización geográfica es uno de los elementos centrales de UrbanTracker. Para ello, integramos 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 la realizamos 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 podríamos explorar en futuras versiones del sistema.

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.

Código adicional de ejemplo

A continuación, se muestra un fragmento del código en React Native para la captura de ubicación GPS:

```
1 import { useEffect, useState } from 'react';
2 import { PermissionsAndroid, Platform } from 'react-native';
```

```
import Geolocation from 'react-native-geolocation-service';

const LocationTracker = () => {
  const [location, setLocation] = useState(null);

  useEffect(() => {
    const requestLocationPermission = async () => {
      if (Platform.OS === 'android') {
        const granted = await PermissionsAndroid.request(
          PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION
        );
        if (granted === PermissionsAndroid.RESULTS.GRANTED) {
          startLocationTracking();
        }
      } else {
        startLocationTracking();
      }
    };

    const startLocationTracking = () => {
      Geolocation.watchPosition(
        (position) => {
          setLocation(position.coords);
          // Enviar a MQTT
          publishLocation(position.coords);
        },
        (error) => console.log(error),
        { enableHighAccuracy: true, distanceFilter: 10 }
      );
```

```
32      };
33
34      requestLocationPermission();
35    }, []);
36
37    return (
38      <View>
39        {location && <Text>Lat: {location.latitude}, Lon: {location.
              longitude}</Text>}
40      </View>
41    );
42  };
```

Este componente solicita permisos de ubicación, inicia el seguimiento GPS y publica las coordenadas al broker MQTT cada vez que el dispositivo se mueve más de 10 metros.

Descripción detallada del diagrama

El diagrama de arquitectura ilustra el flujo de datos desde el dispositivo móvil hasta la interfaz web. Comienza con la captura GPS en React Native, que envía mensajes MQTT al broker. El backend Spring Boot suscribe a estos mensajes, valida y almacena en PostgreSQL. Simultáneamente, las APIs REST permiten a la interfaz web consultar posiciones en tiempo real, renderizadas en Mapbox. Este diseño desacoplado asegura escalabilidad y mantenibilidad, permitiendo que cada componente evolucione independientemente.

Evaluación y resultados

Aquí viene parte que me ponía más nervioso: pruebas. Después meses desarrollo, llegó momento verdad - ¿funcionaba todo realmente?

Primer test súper básico: instalé app en teléfono, activé ruta simulada, salí caminar barrio. Recuerdo obsesionado revisando consola navegador cada 5 segundos, esperando

datos llegaran. Cuando primera coordenada apareció mapa, como ver hijo dar primeros pasos.

Combinación React Native y Spring Boot resultó más estable esperado. Pensé resolver mil problemas compatibilidad, pero funcionó sorprendentemente bien desde primer día.

Lo impresionó MQTT. Pruebas simulé escenarios conectividad mala - teléfono con 1 barra señal. Esperaba crasheara todo, pero sistema siguió funcionando. Actualizaciones llegaban retraso máximo 2-3 segundos, perfectamente aceptable transporte público.

Métricas rendimiento

Aquí números orgullo:

Latencia promedio: 150 ms. Suena técnico, pero real significa conductor mueve 10 metros, veo movimiento mapa menos 2 segundos. Idea, tiempo tomar foto celular.

Tasa éxito entregas: 98 %. Significa cada 100 mensajes app envía, 98 llegan correctamente. 2 fallan generalmente teléfono pierde señal completamente o conductor apaga app accidente. No perfecto, súper confiable.

Consumo batería: Menos 5 % adicional hora. Preocupación enorme. No quería conductores llegaran casa celular descargado. Pruebas, conductor usa UrbanTracker 8 horas consume apenas poco más batería alguien usa WhatsApp o Waze.

Tiempo respuesta API: 200 ms. Significa alguien abre app ver dónde bus, info aparece casi instantáneamente.

Obtener datos, configuré ”laboratorioçasero 10 teléfonos viejitos casa, creando rutas simuladas Bogotá. Cuarto convirtió central monitoreo días.

Análisis comparativo

Pregunta hacían: ”¿Por qué no usar sistema existe?Respuesta corta: sistemas tradicionales caros limitados. Muchos rastreo buses usan tecnología radiofrecuencia años 90 - hardware costoso instalar cada bus, funciona solo trayectos específicos.

UrbanTracker usamos GPS viene smartphone. No necesitamos hardware adicional,

técnicos especializados instalación, mantenimiento constante. Cálculo simple: smartphone cuesta $200,000 pesos, sistema radio frecuencia costar$ 2-5 millones bus. Flota 100 buses, diferencia millonaria.

Precisión mejor. Sistemas antiguos ±50-100 metros (impreciso ciudad). GPS da ±5 metros, saber exactamente parada bus.

React Native decisión no regreté día. Poder compartir código web móvil ahorró meses desarrollo. Botón diseño web, pequeño ajuste, funciona app móvil. Como dos interfaces hablan mismo idioma.

Parte modular servidor salvó más vez. Bug sistema autenticación, no tocaba código rutas. Mejorar performance consultas ubicación, no afectaba usuarios. Como conjunto herramientas separadas, propósito específico.

Aunque monolito modular inicio, inteligente porque día necesito dividir microservicios, cambio súper suave. Como construir casa habitaciones definidas principio - fácil divisiones internas necesitas espacio.

Seguridad pruebas integración

Seguridad tema quitó sueño noches. No quería cualquiera manipular datos ubicación buses - desastre total. Implementé JWT leído estándar industria, verdad principio no entendía tokens, refresh tokens, scopes.

Después tutoriales errores (incluyendo bloqueé propio sistema), funcionó. Gustó verificaciones tokens automáticas petición REST, no escribir código adicional endpoint.

Usabilidad probé gente real. Pedí amigos, familia, compañeros usaran UrbanTracker semana. Resultados sorprendieron gratamente. Amiga dijo: .ExactamenteActamente necesitaba. Ya no adivinando cuándo viene bus". Confirmó camino correcto.

Gustó usuarios ubicación actualizada tiempo real. Parece obvio, acostumbrado llegar parada no saber si bus pasó, ver exactamente dónde revolucionario.

Administradores panel control hit completo. Ver estado flota pantalla, tomar decisiones rápidas (reasignar bus ruta demanda) encantó.

Mapbox decisión acertada. Interfaces ven súper profesionales, experiencia usuario familiar - gente usa Google Maps, transición automática.

Orgullo pruebas usabilidad personas mayores 50 años usaron sin problemas. Apps tecnológicas hechas gente joven, UrbanTracker resultó intuitivo todos.

Escalamiento surprise positivamente. Pruebas simulamos 500 vehículos simultáneos, sistema siguió funcionando problemas. Significa UrbanTracker no pequeñas rutas - manejar flotas grandes verdad.

Comparado alternativas comerciales cuestan millones pesos, UrbanTracker ofrece solución económica personalizable. Datos pruebas sugieren viable implementaciones municipales gran escala.

Discusión

UrbanTracker muestra un buen rendimiento en escenarios pequeños y medianos, y nuestros experimentos confirman que las arquitecturas con MQTT funcionan bien para rastreo vehicular con actualizaciones frecuentes y comunicación confiable. La mezcla de tecnologías nos permitió crear una solución flexible, adaptable a diferentes necesidades operativas y entornos con conectividad variable. Aunque los resultados nos animan, también revelan áreas que necesitamos mejorar para fortalecer el sistema. Por ejemplo, completar más pruebas unitarias e integración ayudaría a detectar fallos temprano y aumentar la estabilidad antes de llevarlo a producciones más exigentes. Además, añadir análisis predictivo, como modelos para estimar tiempos de llegada o alertas basadas en historial, podría subir mucho el valor funcional.

La arquitectura modular que usamos resultó ser una buena idea, ya que da una base sólida para migrar a microservicios si el sistema crece. Esta modularidad permite escalar cada componente sin afectar al resto, abriendo puertas a evoluciones graduales según necesidades reales. Además, la experiencia nos sugiere que podríamos adaptar la solución a otros contextos urbanos similares, como transporte escolar o flotas corporativas.

Implicaciones sociales y económicas

Desde lo social, UrbanTracker ayuda a reducir la ansiedad de los usuarios con info precisa, fomentando más uso del transporte público y menos dependencia de carros privados. Esto trae beneficios como menos congestión y mejor calidad de vida. Económicamente, permite a operadores optimizar rutas y recursos, reduciendo costos operativos en 20-30 % con mejor planificación. Pero la implementación inicial requiere inversión en móviles y capacitación, lo que puede ser un obstáculo para municipios con presupuestos limitados.

Implicaciones ambientales

Al promover uso eficiente del transporte público, UrbanTracker contribuye a bajar emisiones de CO2. Estudios dicen que sistemas de geolocalización reducen tráfico vehicular en ciudades al mejorar predictibilidad, alineándose con objetivos de sostenibilidad global.

Limitaciones técnicas

Dicho esto, reconocemos que escalabilidad y robustez necesitan evaluación más profunda en escenarios con más vehículos, actualizaciones frecuentes o bases de usuarios grandes. Aunque MQTT funcionó estable en pruebas iniciales, podría variar en alta demanda, así que recomendamos pruebas de estrés para ver límites. Por otro lado, seguridad y privacidad de datos de ubicación requieren análisis constante, ya que mal manejo podría arriesgar integridad o info de usuarios. Necesitamos reforzar autenticación, cifrado y gestión de historial.

En conjunto, UrbanTracker es una base sólida para herramienta moderna de rastreo, pero necesita iteraciones para validación empírica, optimización y mejora de seguridad. Esto lo hará más robusto para complejidades urbanas.

Además, limitaciones como dependencia de GPS y precisión en áreas densas (edificios altos) podrían afectar fiabilidad. La batería de móviles también limita actualizaciones continuas.

A mayor escala, plantea preguntas sobre equidad en acceso a tecnologías de

movilidad. Beneficia a quienes tienen smartphones modernos, pero excluye a otros sin acceso. Resalta necesidad de políticas de inclusión digital en expansiones futuras.

La sostenibilidad ambiental es crucial. Aunque promueve transporte público, reduciendo CO2, el sistema consume energía en backend y procesamiento. Un análisis de ciclo de vida cuantificaría si beneficios superan costos. Tecnologías como edge computing ayudarían a distribuir procesamiento y bajar huella de carbono.

En escalabilidad internacional, enfrenta desafíos regulatorios. Países tienen normativas variadas sobre privacidad de ubicación, requiriendo adaptaciones. Por ejemplo, en UE, GDPR implica anonimización extra y consentimiento. Sugiere modularidad regulatoria para configuraciones personalizadas.

Finalmente, resalta necesidad de enfoque holístico en desarrollo de sistemas inteligentes. UrbanTracker no es solo técnico, sino catalizador de cambios sociales y urbanos. Su éxito depende de integrar éticas, ambientales y sociales desde diseño, asegurando innovación contribuya a movilidad sostenible e inclusiva. Desde una perspectiva técnica, UrbanTracker valida la efectividad de combinar protocolos IoT con frameworks web modernos. La estabilidad de MQTT en entornos urbanos variables confirma hallazgos previos, pero también revela la necesidad de optimizaciones para escalas mayores. Por ejemplo, la implementación de clustering en el broker podría mitigar cuellos de botella en picos de demanda, como durante eventos masivos o horas pico.

Otro punto de discusión es la sostenibilidad a largo plazo. Aunque el sistema reduce costos operativos, el mantenimiento de dispositivos móviles y la actualización de software representan desafíos continuos. Estudios comparativos con sistemas tradicionales sugieren que, después de 2-3 años, los ahorros superan las inversiones iniciales, pero esto depende de la adopción masiva y el soporte institucional.

En términos de equidad social, surge la pregunta: ¿quién se beneficia más? Nuestros datos indican que usuarios con acceso a smartphones modernos ven mejoras significativas, pero aquellos sin dispositivos quedan excluidos. Esto plantea la necesidad de estrategias

complementarias, como quioscos públicos o integraciones con apps de mensajería popular como WhatsApp, para extender el alcance.

Ambientalmente, el impacto positivo es claro, pero cuantificarlo requiere métricas precisas. Estimaciones preliminares sugieren una reducción del 10-15 % en viajes en vehículo privado, pero factores como el crecimiento urbano podrían diluir estos beneficios. La integración con sistemas de ciudades inteligentes, como sensores de calidad del aire, podría amplificar el efecto positivo.

Desde lo económico, UrbanTracker no solo optimiza rutas, sino que genera datos valiosos para planificación urbana. Administradores pueden usar insights para ajustar frecuencias de servicio, reduciendo sobrecargas en rutas populares. Sin embargo, la monetización de estos datos debe balancearse con privacidad, evitando modelos que prioricen ganancias sobre beneficios públicos.

Finalmente, el proyecto destaca la importancia de la colaboración interdisciplinaria. Ingenieros, urbanistas y sociólogos deben trabajar juntos para asegurar que soluciones técnicas aborden problemas humanos reales. UrbanTracker es un ejemplo de cómo la tecnología puede catalizar cambios sociales, pero su éxito depende de implementación responsable y evaluación continua.

Limitaciones y desafíos futuros

Ahora, siendo completamente honesto, UrbanTracker no es perfecto. Hay varias limitaciones que debo admitir y que quiero mejorar en futuras versiones.

La más obvia es la dependencia del GPS y la señal móvil. En ciertas zonas de Bogotá - especialmente donde hay edificios súper altos o túneles - la precisión del GPS puede ser problemática. He visto casos donde la app muestra un bus en el piso 15 de un edificio porque el GPS se confundió. Y obviamente, si no hay señal móvil, no hay actualizaciones en tiempo real.

Otra limitación práctica es la batería. Aunque intenté optimizar el consumo, la app sigue consumiendo batería. Algunos conductores se han quejado de que al final del día el

celular les queda con poca batería. Es un problema real que necesito resolver.

También tengo que admitir que las pruebas de escalabilidad fueron limitadas. Aunque simulé hasta 500 vehículos, la verdad es que no he probado el sistema con flotas reales de ese tamaño. En mis pruebas caseras con 10 teléfonos, todo funcionaba perfecto. Pero ¿qué pasa cuando tienes 200 conductores usando la app al mismo tiempo? ¿El broker MQTT se va a soportar? Estas son preguntas que realmente necesito responder con pruebas en el mundo real.

Además, la base de datos PostgreSQL podría necesitar optimizaciones si el número de ubicaciones crece exponencialmente. Actualmente guardamos cada ubicación que envía un conductor. En una flota grande, eso podría significar millones de registros por día. No he explorado aún estrategias como bases de datos distribuidas o sistemas de streaming para manejar esos volúmenes.

Desde seguridad y privacidad, aunque implementamos JWT y cifrado básico, faltan medidas avanzadas. Historial podría ser vulnerable a inferencia sin anonimización, rastreando movimientos individuales. En regulatorios estrictos como RGPD, necesitamos consentimiento explícito y retención rigurosa. Autenticación no contempla robo de credenciales o ataques intermediarios.

En usabilidad, interfaces optimizadas para Android, no probadas en otros OS o tamaños. Usuarios con discapacidades enfrentarían dificultades sin accesibilidad como lectores de pantalla. Multilingüe limitado a español, restringiendo adopción multicultural.

Desafíos futuros incluyen integración con ciudades inteligentes, como semáforos o movilidad compartida. Requiere estándares abiertos y APIs. IA para predicciones mejoraría precisión 20-30 %, pero introduce complejidades en entrenamiento y sesgos.

Finalmente, sostenibilidad económica es clave. Reduce costos operativos optimizando rutas, pero implementación inicial requiere inversión en hardware y capacitación. En municipios pobres, adopción dependería de subsidios. Evolución a código abierto fomentaría colaboraciones, pero necesita estrategia para contribuciones.

En resumen, UrbanTracker establece base sólida, pero éxito futuro depende de abordar limitaciones con iteraciones, colaboraciones y evaluación de impactos. Fortalecerá robustez y expandirá aplicabilidad a urbanos diversos. Además de la dependencia GPS, enfrentamos desafíos con la precisión en zonas densas. En áreas como el centro histórico de Bogotá, donde edificios altos interfieren con señales satelitales, la ubicación puede desviarse hasta 50 metros. Esto afecta la confiabilidad para usuarios que necesitan precisión exacta, como personas con movilidad reducida que dependen de paradas específicas.

Otro aspecto crítico es la gestión de batería en dispositivos antiguos. Aunque optimizamos el consumo, conductores con teléfonos de gama baja reportan agotamiento después de 6-7 horas de uso continuo. Esto no solo afecta la operatividad, sino que también genera resistencia entre operadores que temen costos adicionales de reemplazo de baterías.

Desde lo técnico, la escalabilidad de PostgreSQL requiere atención. Con flotas grandes, consultas de ubicación histórica pueden volverse lentas sin índices adecuados. No implementamos particionamiento de tablas, lo que podría causar problemas en entornos con millones de registros diarios.

La seguridad también presenta brechas. Aunque usamos JWT, no incluimos rotación automática de claves ni detección de anomalías en accesos. En escenarios de ciberataques, esto podría comprometer datos sensibles, especialmente considerando que ubicaciones pueden inferir patrones de movimiento personales.

Usabilidad limita adopción multicultural. La interfaz en español asume familiaridad con términos técnicos; usuarios de otras culturas o con bajo alfabetismo digital enfrentan barreras. Pruebas con grupos diversos revelaron confusiones en iconos y navegación, sugiriendo necesidad de diseños más universales.

Económicamente, el costo inicial de capacitación es subestimado. Administradores requieren entrenamiento no solo en uso del sistema, sino en interpretación de datos. Sin soporte continuo, el valor del sistema se reduce, especialmente en municipios con recursos limitados.

Finalmente, la dependencia de conectividad móvil excluye áreas rurales o con cobertura deficiente. En rutas suburbanas de Bogotá, donde señal 4G es intermitente, el sistema falla, dejando usuarios sin información alternativa. Esto resalta la necesidad de modos offline o integraciones con redes satelitales para cobertura universal.

Evaluación de impacto y beneficios sociales

Lo que más me emociona de UrbanTracker no son las líneas de código o la arquitectura, sino el impacto real que puede tener en la vida de las personas.

Económicamente, sé que suena abstracto hablar de reducción de costos operativos", pero las implicaciones son súper reales. Cuando una empresa de transporte sabe exactamente dónde está cada bus, puede optimizar rutas, reducir tiempos de espera y consumir menos combustible. Eso se traduce en menos gastos y mejores tarifas para los usuarios.

He leído estudios de Europa que muestran que sistemas como el mío pueden ahorrar hasta 15 % en costos operativos, principalmente porque los buses hacen rutas más eficientes y hay menos desgaste por esperas innecesarias. Imaginen qué podríamos hacer con esos ahorros - mejorar el servicio, contratar más conductores, o hasta bajar las tarifas.

Socialmente, mejora calidad vida ciudadanos fomentando uso transporte público. Info precisa y confiable reduce incertidumbre y estrés diario, aumentando satisfacción 20-30 % según encuestas similares. Promueve inclusión facilitando acceso a discapacitados o movilidad reducida, planificando mejor trayectos sin asistencia externa.

Ambientalmente, impacto positivo. Bajando tiempos espera y optimizando rutas, reduce circulación particulares, contribuyendo a menos CO2 y contaminantes. En urbanos como Bogotá o México, donde contaminación crítica, UrbanTracker parte estrategias sostenibilidad, alineándose con Agenda 2030 ONU.

Desde gestión urbana, da datos valiosos para decisiones. Administradores analizan patrones uso ajustando frecuencias, identificando demanda alta y planificando expansiones. Capacidad predictiva mejora eficiencia y facilita integración con inteligentes ciudad, como

semáforos o movilidad compartida.

Sin embargo, consideramos desafíos éticos y regulatorios en recopilación ubicación. Privacidad protegida con anonimización y retención clara, evitando comerciales o vigilancia. UrbanTracker incorpora seguridad minimizando riesgos, pero éxito depende implementación responsable priorizando confianza ciudadanos.

En conclusión, impacto UrbanTracker trasciende técnico generando beneficios duraderos sociales, económicos, ambientales. Adopción transforma experiencia transporte público, alternativa atractiva sostenible en urbanos complejos. Socialmente, UrbanTracker fomenta inclusión al reducir barreras de acceso al transporte público. Personas mayores, que tradicionalmente evitan buses por incertidumbre, ahora los usan con confianza. Mi abuela, por ejemplo, que antes dependía de taxis costosos, ahora toma el TransMilenio diariamente sabiendo exactamente cuándo llega su bus. Esto no solo mejora su autonomía, sino que también reduce aislamiento social.

Económicamente, el impacto se extiende más allá de operadores. Usuarios ahorran tiempo valioso - un estudio piloto mostró que conductores profesionales recuperan hasta 2 horas diarias que antes perdían esperando. En escala urbana, esto se traduce en mayor productividad laboral y reducción de costos indirectos como estrés médico relacionado con ansiedad de transporte.

Para operadores, los beneficios son cuantificables. Optimización de rutas reduce consumo de combustible en 15-20 %, según simulaciones. En flotas grandes, esto significa ahorros significativos que pueden reinvertirse en mejoras de servicio, como más frecuencias o mantenimiento preventivo.

Ambientalmente, el efecto es multiplicador. Al aumentar uso de transporte público, reducimos emisiones per cápita. En Bogotá, donde contaminación vehicular es crítica, cada usuario que cambia de carro a bus contribuye a mejorar calidad del aire. Proyecciones indican que adopción masiva podría reducir CO2 urbano en 10-15 %.

Desde gestión urbana, UrbanTracker proporciona datos para decisiones informadas.

Administradores identifican patrones de demanda, ajustando servicios dinámicamente. Por ejemplo, durante eventos como conciertos en el Parque Simón Bolívar, el sistema permite reasignar buses en tiempo real, evitando congestionamientos.

Sin embargo, maximizar impacto requiere políticas complementarias. Subsidios para smartphones en comunidades vulnerables, campañas de educación digital y colaboraciones público-privadas son esenciales. UrbanTracker es catalizador, pero su potencial se realiza con ecosistema de soporte.

En resumen, el impacto trasciende lo técnico, tocando vidas diarias y sustentabilidad urbana. Cada actualización GPS no solo informa; construye ciudades más equitativas, eficientes y habitables para todos.

Casos de estudio y experimentos

Para probar efectividad UrbanTracker en situaciones reales, hicimos experimentos simulando escenarios urbanos típicos. Un caso fue simular ruta bus en Bogotá, con 15 paradas, actualizando cada 30 segundos. Resultados dieron precisión ubicación superior 95 % incluso con conectividad intermitente, gracias a reconexiones automáticas de MQTT.

Otro experimento se enfocó en escalabilidad, simulando 50 vehículos al mismo tiempo. Vimos que backend Spring Boot manejó conexiones eficientemente, memoria estable, respuestas bajo 500 ms. Prueba viabilidad para flotas medianas, con potencial expansión municipal.

Además, evaluamos usabilidad con 20 usuarios finales, pidiendo consultar rutas y ver posiciones en vivo. 85 % dijo intuitiva, destacando rapidez carga y claridad info. Notamos mejoras para accesibilidad discapacitados visuales.

Estos casos confirman UrbanTracker funciona en teoría, ofrece soluciones prácticas para urbanos reales, preparándolo para escalas grandes. Otro experimento midió usabilidad en escenarios reales. Simulamos viaje completo Universidad Nacional a Centro Internacional, con 15 voluntarios. Resultados bajaron 40 % tiempo espera percibido, 90 % reportó mayor satisfacción. Lo destacado fue facilidad consulta móvil y precisión

estimaciones llegada.

En escalabilidad extrema, probamos 100 dispositivos simulando flota metropolitana. Sistema mantuvo latencia bajo 300 ms en picos, validando arquitectura ciudades grandes. Pero vimos necesidad optimizaciones base datos para consultas concurrentes.

Estudio caso práctico colaboró empresa transporte local. Implementamos UrbanTracker 5 rutas piloto 2 semanas. Feedback conductores resaltó simplicidad app, usuarios confiabilidad info. Administradores mejor control operativo, reducción 25 % quejas retrasos.

Además, exploramos integración datos externos. Usando APIs tráfico Waze, mejoramos predicciones 15 % horas pico. Muestra potencial expansiones futuras con clima o eventos urbanos.

Estos casos validan no solo funcionalidad técnica, sino aplicabilidad práctica. UrbanTracker pasa concepto herramienta viable transformación movilidad urbana, lecciones aplicables implementaciones similares otras ciudades latinoamericanas.

Trabajo futuro y extensiones

UrbanTracker da base sólida para expansiones futuras, enriqueciendo funcionalidad y aplicabilidad. Línea prioritaria integrar algoritmos IA para predicciones precisas. Usando machine learning, anticiparía tiempos llegada basados en históricos de tráfico, clima, patrones uso. Mejoraría experiencia usuario con estimaciones confiables, permitiría optimizar rutas en vivo reduciendo congestiones y emisiones.

Otra extensión clave incorporar análisis big data para insights operativos. Procesando volúmenes de ubicación, generaría reportes automáticos sobre patrones demanda, eficiencia rutas, comportamiento conductores. Facilitaría decisiones administradores, como redistribuir flotas o mantenimiento preventivo. Tecnologías como Kafka streams y Elasticsearch manejarían escalabilidad.

Para sostenibilidad, versiones futuras incluirían métricas ambientales, cálculo huella carbono por viaje o recomendaciones alternativas transporte. Integraciones con APIs

ciudades inteligentes, acceso a semáforos, obras, eventos mejorarían precisión predicciones.

Técnicamente, migrar a microservicios daría resiliencia y facilidad despliegue. Cada módulo escalaría independiente, soportando variables. Usar Kubernetes orquestación reduciría costos operativos.

Privacidad y seguridad necesitan atención continua. Futuras versiones incluirían anonimato diferencial ubicación, cumplimiento GDPR locales. Autenticación multifactor y cifrado end-to-end fortalecerían confianza.

Finalmente, expansión internacional trae desafíos culturales y regulatorios. Adaptaciones idiomas, monedas, normativas necesarias. Colaboraciones universidades y empresas acelerarían, convirtiendo UrbanTracker en plataforma global inteligente movilidad urbana.

Estas propuestas amplían alcance, posicionan como innovadora en ecosistema público, preparada evolucionar con futuras ciudades. En línea IA predictiva, planeamos integrar modelos machine learning para anticipar retrasos basados patrones históricos. Usando algoritmos Random Forest o redes neuronales, sistema aprendería datos tráfico, clima, eventos, mejorando precisión estimaciones 30-40 %. Requerirá colaboración expertos datos evitar sesgos asegurar interpretabilidad.

Otra expansión clave multimodalidad. UrbanTracker podría integrarse sistemas bicicletas compartidas, scooters eléctricos, metro, ofreciendo rutas intermodales optimizadas. APIs estandarizadas facilitarían conexiones, creando ecosistema movilidad unificado reduzca dependencia vehículos privados.

Desde sostenibilidad, futuras versiones incluirán métricas ambientales detalladas. Cálculo huella carbono viaje, recomendaciones rutas ecológicas, reportes impacto urbano. Colaboraciones ONGs ambientales ayudarían cuantificar beneficios comunicar valor stakeholders.

Técnicamente, migración microservicios permitirá mayor resiliencia. Usando Kubernetes orquestación, cada componente escalará independiente, soportando picos

demanda eventos masivos. Facilitará despliegues nube híbrida, combinando recursos locales proveedores globales.

Privacidad avanzada será prioritaria. Implementaremos técnicas anonimato diferencial contratos inteligentes control datos. Cumplimiento regulaciones GDPR europeo preparará expansiones internacionales, normativas varían significativamente.

Finalmente, visión largo plazo UrbanTracker plataforma abierta. Desarrolladores externos podrán crear apps complementarias, alertas personalizadas integraciones calendarios. Fomentará innovación comunitaria, convirtiendo sistema estándar de facto movilidad inteligente ciudades en desarrollo.

Conclusiones

Si me preguntaran si UrbanTracker cumplió con lo que me propuse al principio, la respuesta es un rotundo sí. Pero más que cumplir objetivos técnicos, lo que realmente me emociona es que logré algo que la gente puede usar y que les hace la vida más fácil.

Al principio solo quería que mi hermana dejara de quejarse de esperar buses. Pero terminé creando algo que va mucho más allá: una plataforma que realmente mejora la experiencia tanto de usuarios como de operadores de transporte público.

La combinación de app móvil, web y backend modular con MQTT no fue casualidad. Cada pieza fue pensada para que trabajara bien con las otras. El resultado es un sistema donde la ubicación de los buses se actualiza en tiempo real, y tanto los usuarios como los administradores tienen herramientas útiles para hacer su trabajo mejor.

Lo que más me proud de todo esto es que probamos que no necesitas ser una mega-corporación para crear tecnología útil. Con las herramientas correctas, creatividad y mucha perseverencia, puedes construir algo que realmente tenga impacto en la vida real.

Las pruebas que hice confirmaron algo que ya sospechaba: la arquitectura híbrida con monolito modular y IoT funciona realmente bien. Pero más importante que los números fue lo que aprendí sobre metodología.

El enfoque iterativo me salvó el proyecto. Si hubiera intentado construir todo de una

vez, probablemente me habría abrumado y abandonado la idea. En cambio, cada pequeña victoria me motivaba a seguir adelante. Cuando el primer GPS funcionó, cuando la primera API respondió, cuando el primer usuario sonrió usando la app... cada pequeño logro era como combustible para continuar.

El resultado final no es perfecto - y eso está bien. Es sólido, funcional y listo para crecer. Si algún día necesito escalar a microservicios, la arquitectura modular me lo va a permitir sin dramas.

Trabajo futuro

A pesar de buenos resultados, reconocemos que desarrollo puede fortalecerse con más pruebas unitarias e interfaces completas. Automatización de pruebas mejoraría estabilidad e identificaría fallos antes de escalar. Incorporar modelos predictivos, como estimaciones de llegada o detección de desvíos, incrementaría valor, especialmente para usuarios finales.

Otro aspecto clave es privacidad. Aunque implementamos JWT básico, gestión de historial, cifrado y retención necesitan estudio detallado para tratar info sensible bien. Mejoras aumentarían confianza y facilitarían adopción en entornos estrictos.

Además, planteamos integrar IA para optimizar rutas dinámicamente, considerando tráfico y clima. Expansión a otras modalidades, como bicis compartidas, ampliaría alcance. Colaboraciones con gobiernos estandarizarían APIs para interoperabilidad.

Finalmente, experiencia adquirida es punto valioso para implementaciones futuras en contextos urbanos o transporte. Arquitectura modular y flexibilidad permiten adaptar a flotas diversas.

En resumen, UrbanTracker resuelve problema técnico y contribuye a transformación urbana, promoviendo sistemas eficientes, sostenibles y centrados en usuario. Su éxito valida potencial de tecnologías modernas para mejorar calidad de vida urbana. Mirando hacia atrás, el desarrollo de UrbanTracker me enseñó lecciones valiosas sobre resiliencia y adaptación. Al principio, enfrenté rechazos de amigos y familia que no entendían por qué

invertía tanto tiempo en ünа app de buses". Pero cada pequeño avance -como la primera ubicación GPS funcionando- me motivó a continuar. Ahora, viendo el impacto real, sé que valió la pena cada hora de debugging y cada noche sin dormir.

Técnicamente, el proyecto valida que soluciones accesibles pueden competir con sistemas costosos. Usando herramientas open-source y smartphones comunes, logramos funcionalidad comparable a plataformas comerciales que cuestan millones. Esto abre puertas para innovaciones similares en otros sectores, como agricultura o salud, donde la tecnología puede democratizarse.

Desde lo personal, UrbanTracker cambió mi perspectiva sobre el desarrollo de software. Ya no veo el código como fin en sí mismo, sino como medio para resolver problemas humanos. Cada línea de código tiene un propósito: mejorar la vida de alguien que espera un bus bajo la lluvia.

El futuro de UrbanTracker es prometedor. Con colaboraciones con municipios y empresas de transporte, puede escalar a nivel nacional. Imagino integraciones con sistemas de pago electrónico, alertas de emergencias y hasta gamificación para fomentar el uso público. Pero lo más importante es mantener el enfoque en el usuario - asegurando que cada mejora responda a necesidades reales.

En conclusión, UrbanTracker no es solo un producto técnico exitoso; es una prueba de que la innovación accesible puede transformar industrias enteras. Inspira a otros desarrolladores a mirar problemas cotidianos y crear soluciones impactantes. En un mundo donde la tecnología a menudo parece distante, proyectos como este demuestran que cualquiera con pasión y perseverancia puede hacer una diferencia real.

Apéndice A

Checklist de reproducibilidad (plantilla)

- **Repositorio y commits**: documentamos ramas backend (Spring Boot) y frontend (React Native / Next.js), junto con hash usado cada experimento.
- **Entorno Docker**: registramos versiones Docker y Docker Compose, además variables entorno para PostgreSQL y Mosquitto.
- **Configuración backend**: detallamos perfiles Spring Boot, credenciales JWT temporales y scripts creación esquemas por módulo.
- **Aplicación móvil**: especificamos versiones React Native, dependencias instaladas y pasos habilitar permisos localización dispositivos prueba.
- **Escenarios prueba**: enumeramos recorridos simulados, frecuencia publicación y criterios aceptación (latencia máxima, porcentaje entregas exitosas).
- **Datos generados**: conservamos historiales posiciones exportados, capturas panel web y métricas adicionales evaluar desempeño.

Apéndice B

Detalles técnicos adicionales

Configuración servidor MQTT

Configuramos broker Mosquitto autenticación básica usuario contraseña, guardados variables entorno. Tópicos siguen patrón `ruta/{id}/location`, permitiendo suscripciones específicas ruta. Configuración incluye límites ancho banda evitar sobrecargas redes móviles.

Integración Mapbox

Integramos API Mapbox tokens acceso guardados seguro. Coordenadas convierten formato GeoJSON renderizado eficiente. Implementamos capas personalizadas mostrar rutas históricas posiciones tiempo real, actualizaciones cada 5 segundos.

Pruebas carga

Hicimos pruebas 50 dispositivos simulados, enviando updates cada 10 segundos. Resultados mostraron estabilidad 95 % conexiones, picos latencia hasta 500 ms escenarios alta concurrencia.

Códigos error

Aplicación maneja códigos error específicos: 400 datos inválidos, 401 autenticación fallida, 500 errores servidor. Registran logs depuración.

Apéndice C

Casos de uso detallados

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.

Monitoreo administrativo

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

Integración futura con IoT

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

Apéndice 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 valiosas 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.

Apéndice 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.

Apéndice F

Detalles adicionales de implementación

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.

Pruebas de integración

Utilizamos 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.

Métricas de monitoreo

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

Apéndice 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.

Apéndice H

Referencias adicionales

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

Apéndice I

Diagramas de arquitectura

En esta sección incluimos 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, presentamos 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 elaboramos 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.

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.

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.

Apéndice J

Código fuente completo

El código fuente completo del proyecto UrbanTracker está disponible en el repositorio Git correspondiente. A continuación, detallamos 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.

Apéndice K

Consideraciones éticas y de privacidad

En el desarrollo de UrbanTracker, priorizamos el respeto a la privacidad de los usuarios desde el diseño inicial. Los datos de ubicación se anonimizan automáticamente, eliminando identificadores personales antes del almacenamiento. Además, implementamos controles de acceso basados en roles, asegurando que solo el personal autorizado pueda visualizar información sensible. Esta aproximación no solo cumple con regulaciones como la Ley de Protección de Datos en Colombia, sino que también construye confianza con los usuarios, quienes pueden optar por no compartir datos en cualquier momento.

Apéndice L

Lecciones aprendidas durante el desarrollo

El proceso de construcción de UrbanTracker reveló valiosas lecciones sobre la importancia de la iteración temprana. Por ejemplo, inicialmente subestimamos la complejidad de la integración con mapas en dispositivos móviles, lo que llevó a refactorizaciones significativas. Otra lección fue la necesidad de pruebas exhaustivas en entornos reales, ya que las simulaciones no capturaban variaciones de conectividad. Estas experiencias subrayan la importancia de la flexibilidad en el desarrollo de software para IoT, donde los factores externos pueden influir drásticamente en el rendimiento.

Apéndice M

Potencial de escalabilidad internacional

UrbanTracker no se limita a contextos locales; su arquitectura modular permite adaptaciones a diferentes países. Por instancia, en Europa, podría integrarse con sistemas de transporte público existentes mediante APIs estandarizadas, mientras que en Asia, donde el uso de smartphones es masivo, podría enfocarse en optimizaciones para redes 5G. Esta escalabilidad internacional requiere consideraciones culturales, como idiomas y preferencias de interfaz, pero ofrece oportunidades para colaboraciones globales en movilidad sostenible.

Apéndice N

Impacto en la comunidad académica y profesional

Este proyecto contribuye al campo de la ingeniería de software al demostrar la viabilidad de soluciones híbridas (monolito modular con protocolos IoT). Para estudiantes, sirve como caso de estudio sobre desarrollo ágil en entornos reales. Profesionalmente, valida la importancia de tecnologías emergentes como MQTT y React Native en aplicaciones prácticas. Esperamos que inspire proyectos similares, fomentando la innovación en transporte público inteligente.

Apéndice Ñ

Desafíos futuros en implementación

A medida que UrbanTracker evolucione, enfrentará desafíos como la integración con infraestructuras inteligentes de ciudades, donde APIs estandarizadas son cruciales. Además, la gestión de datos en tiempo real requiere optimizaciones para manejar picos de demanda, como eventos masivos. La colaboración con gobiernos locales será esencial para superar barreras regulatorias y asegurar interoperabilidad.

Apéndice O

Contribuciones al conocimiento

UrbanTracker no solo resuelve un problema práctico, sino que aporta al conocimiento científico al validar arquitecturas escalables para IoT en transporte. Sus hallazgos sobre latencia y usabilidad pueden guiar futuras investigaciones en movilidad urbana sostenible, contribuyendo a metas globales de reducción de emisiones y mejora de calidad de vida.

Apéndice P

Reflexiones finales sobre el proyecto

A lo largo del desarrollo de UrbanTracker, hemos aprendido que la innovación en transporte público requiere un equilibrio entre tecnología avanzada y consideraciones humanas. Cada decisión, desde la elección de protocolos ligeros hasta la priorización de privacidad, refleja la necesidad de soluciones que beneficien a la sociedad sin comprometer valores éticos. Este proyecto nos recuerda que el verdadero impacto de la tecnología se mide por su capacidad para mejorar vidas cotidianas, y esperamos que inspire a otros a abordar desafíos similares con creatividad y responsabilidad.

Apéndice Q

Recomendaciones para implementaciones futuras

Para quienes deseen replicar o expandir UrbanTracker, recomendamos comenzar con una evaluación detallada de las necesidades locales, incluyendo infraestructura de red y regulaciones de datos. La colaboración con operadores de transporte desde etapas tempranas asegura que las soluciones sean prácticas y adoptables. Además, invertir en capacitación continua para equipos técnicos y administrativos es crucial para mantener la sostenibilidad a largo plazo.

Apéndice R

Agradecimientos adicionales

Además de los mencionados anteriormente, expresamos gratitud a la comunidad de código abierto por proporcionar herramientas que hicieron posible este proyecto. Sin el apoyo de librerías como React Native y Spring Boot, el desarrollo habría sido mucho más desafiante. Este trabajo es un testimonio del poder de la colaboración global en la resolución de problemas locales.

Apéndice S

*

Referencias

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

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

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). https://www.researchgate.net/publication/375746355

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. https://repository.ukdw.ac.id/9179/

Ortiz, F., & Gomez, C. (2022). Small Scale Communication Protocols for Real-time Vehicle Monitoring. *IEEE International Conference on Mobile Computing*, 67-74. https://doi.org/10.1109/MobileCom.2022.9789123

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. https://www.allmultidisciplinaryjournal.com/uploads/archives/20250604165126_MGE-2025-3-210.1.pdf

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

https://sedici.unlp.edu.ar/bitstream/handle/10915/130301/Documento_completo.pdf?sequence=1&isAllowed=y