Ciencia y Educación
(L-ISSN: 2790-8402 E-ISSN: 2707-3378)
Vol. 7 No. 3
Marzo del 2026

DESARROLLO DE UN PROTOTIPO MÓVIL MULTIPLATAFORMA PARA SERVICIOS TURÍSTICOS EN CUENCA, ECUADOR BASADO EN LA EVALUACIÓN DE TECNOLOGÍAS ACTUALES

DEVELOPMENT OF A MULTIPLATFORM MOBILE PROTOTYPE FOR TOURISM SERVICES IN CUENCA, ECUADOR, BASED ON THE EVALUATION OF CURRENT TECHNOLOGIES

Autores: [1]Miguel Andrés Rivas Loyola, [2]Matías Daniel Echeverría Campoverde, [3]Luis Fernando Pinos Castillo.

[1]ORCID ID: https://orcid.org/0009-0008-7522-5649
[2]ORCID ID: https://orcid.org/0009-0005-4403-2690
[3]ORCID ID: https://orcid.org/0009-0004-5491-1712
[1]E-mail de contacto: matias.echeverria@est.ucacue.edu.ec
[2]E-mail de contacto: miguel.rivas.25@est.ucacue.edu.ec
[3]E-mail de contacto: lfpinos@ucacue.edu.ec
Afiliación: [1*2*3*]Universidad Católica de Cuenca, (Ecuador).
Artículo recibido: 5 de Marzo del 2026
Artículo revisado: 7 de Marzo del 2026
Artículo aprobado: 10 de Marzo del 2026
[1]Estudiante de último año de la carrera de Software de la Universidad Católica de Cuenca, (Ecuador).
[2]Estudiante de último año de la carrera de Software de la Universidad Católica de Cuenca, (Ecuador).
[3]Ingeniero de Sistemas por la Universidad Politécnica Salesiana, con 20 años de experiencia en docencia universitaria. Magíster en Redes de Comunicaciones por la Pontificia Universidad Católica del Ecuador y Magíster en Tecnologías de la Información por la Universidad Católica de Cuenca. Además, desarrollador de software con participación en diversos proyectos de innovación tecnológica, con sólidos conocimientos en tecnologías Full Stack.

Resumen

La transformación digital del sector turístico exige herramientas tecnológicas escalables y de alto rendimiento. Este estudio aborda el desarrollo de una aplicación móvil multiplataforma para la gestión de información turística, proponiendo una arquitectura optimizada para la geolocalización y la gestión de datos en tiempo real. El objetivo principal fue identificar y validar la tecnología más eficiente para la renderización de mapas y la sincronización de contenidos. Metodológicamente, se aplicó un diseño experimental en dos fases: primero, una evaluación comparativa de los frameworks líderes (Flutter, React Native, Xamarin e Ionic) basada en métricas de rendimiento (tiempo de inicio y consumo de CPU); y segundo, la implementación de un prototipo funcional utilizando la tecnología seleccionada. Los resultados determinaron la superioridad técnica de Flutter, que registró un tiempo de inicio de 1.27 segundos y una renderización estable a 110 FPS, superando las limitaciones de latencia de los enfoques híbridos. Como resultado práctico, se desarrolló un aplicativo con Clean Architecture e infraestructura Backend-as-a-Service (BaaS) basada en Supabase, validando que el enfoque de compilación cruzada permite ofrecer experiencias fluidas y una gestión de datos robusta, equiparable al desarrollo nativo con mayor eficiencia en costos.

Palabras clave: Desarrollo de aplicaciones móviles, Turismo inteligente, Flutter, Rendimiento multiplataforma, Supabase, Arquitectura de software.

Abstract

The digital transformation of the tourism sector demands scalable and high-performance technological tools. This study addresses the development of a cross-platform mobile application for tourist information management, proposing an architecture optimized for geolocation and real-time data management. The main objective was to identify and validate the most efficient technology for map rendering and content synchronization. Methodologically, an experimental design was applied in two phases: first, a comparative evaluation of leading

frameworks (Flutter, React Native, Xamarin, and Ionic) based on performance metrics (launch time and CPU consumption); and second, the implementation of a functional prototype using the selected technology. The results determined the technical superiority of Flutter, which recorded a launch time of 1.27 seconds and stable rendering at 110 FPS, overcoming the latency limitations of hybrid approaches. As a practical result, an application with Clean Architecture and Supabase-based Backend-as-a-Service (BaaS) infrastructure was developed, validating that the cross-compiled approach allows for fluid user experiences and robust data management, comparable to native development but with greater cost efficiency.

Keywords: Mobile application development, Smart tourism, Flutter, Cross-platform performance, Supabase, Software architecture.

Sumário

A transformação digital do setor turístico exige ferramentas tecnológicas escaláveis e de alto desempenho. Este estudo aborda o desenvolvimento de uma aplicação móvel multiplataforma para gestão de informações turísticas, propondo uma arquitetura otimizada para geolocalização e gestão de dados em tempo real. O objetivo principal foi identificar e validar a tecnologia mais eficiente para renderização de mapas e sincronização de conteúdos. Metodologicamente, aplicou-se um desenho experimental em duas fases: primeiro, uma avaliação comparativa dos principais frameworks (Flutter, React Native, Xamarin e Ionic) baseada em métricas de desempenho (tempo de inicialização e consumo de CPU); e segundo, a implementação de um protótipo funcional utilizando a tecnologia selecionada. Os resultados determinaram a superioridade técnica do Flutter, que registrou um tempo de inicialização de 1,27 segundos e renderização estável a 110 FPS, superando as limitações de latência das abordagens híbridas. Como resultado prático, desenvolveu-se um aplicativo com Clean Architecture e infraestrutura Backend-as-a-Service (BaaS) baseada no Supabase, validando que a abordagem de compilação cruzada permite oferecer experiências fluidas e uma gestão de dados robusta, comparável ao desenvolvimento nativo com maior eficiência em custos.

Palavras-chave: Desenvolvimento de aplicativos móveis, Turismo inteligente, Flutter, Desempenho multiplataforma, Supabase, Arquitetura de software.

Introducción

En la última década, el paradigma del 'Turismo Inteligente' ha redefinido la interacción entre los destinos y los visitantes, integrando la tecnología como un mediador de experiencias personalizadas (Neuhofer et al., 2021). Sin embargo, la implementación de infraestructuras digitales enfrenta desafíos técnicos en regiones en vías de desarrollo. Según David y Gutiérrez (2025), factores como la expectativa de resultado y las condiciones facilitadoras influyen positivamente en la intención de los turistas de utilizar aplicaciones móviles durante sus viajes, lo que determina el éxito de estas herramientas en destinos turísticos modernos. (Dias y Afonso, 2021) enfatizan que el acceso inmediato a información contextual reduce la incertidumbre del viajero, lo que genera la necesidad técnica de desarrollar aplicativos capaces de gestionar mapas interactivos y datos complejos en tiempo real sin sacrificar el rendimiento en dispositivos de gama media.

El desarrollo de este tipo de soluciones plantea una disyuntiva ingenieril: ¿cómo equilibrar el rendimiento nativo con la eficiencia de costos del desarrollo multiplataforma? La literatura actual presenta un debate abierto sobre la arquitectura idónea. Históricamente, los enfoques híbridos como Ionic han sido populares por su rapidez de despliegue, pero estudios recientes de Kodithuwak y Pacillo (2025) advierten que su dependencia de contenedores web (WebViews) resulta en un consumo excesivo de recursos y tiempos de

carga elevados (3.30 s). Por otro lado, frameworks interpretados como React Native presentan un consumo variable de recursos del sistema según el análisis comparativo de Markowski y Smołka (2023) quienes encontraron diferencias significativas en el uso de memoria virtual (22 GB vs 7 GB de Flutter) aunque el consumo general de CPU fue similar entre ambas tecnologías

Ante este escenario, surge la necesidad de evaluar nuevas arquitecturas de compilación cruzada, como Flutter, combinadas con infraestructuras Backend-as-a-Service (BaaS) modernas. La presente investigación tiene como objetivo determinar la tecnología más eficiente para el desarrollo de guías turísticas móviles y validar su viabilidad mediante un prototipo funcional. Para ello, se adoptó una metodología experimental dividida en dos etapas: una evaluación comparativa cuantitativa de los cuatro frameworks líderes basada en métricas de tiempo de inicio y eficiencia de CPU; y el desarrollo posterior de un Producto Mínimo Viable (MVP) que implementa una arquitectura basada en Supabase, validando las ventajas de las arquitecturas Backend-as-a-Service documentadas por (Khawas y Shah, 2018). Los resultados obtenidos en este estudio no solo ofrecen una comparativa técnica actualizada que posiciona a las arquitecturas compiladas por encima de las interpretadas, sino que también proporcionan una implementación de referencia sobre cómo integrar servicios BaaS escalables para modernizar la gestión tecnológica en el sector turístico.

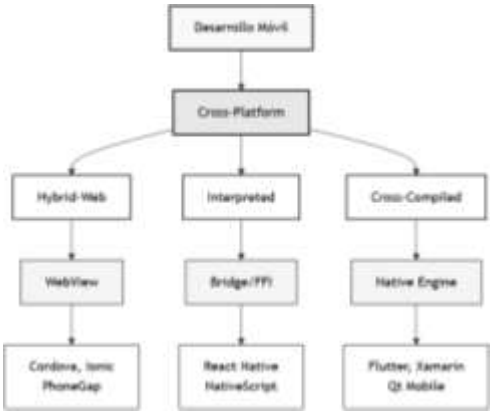
Materiales y Métodos

La presente investigación se fundamenta en un diseño experimental de corte cuantitativo, bajo un enfoque hipotético-deductivo. El procedimiento metodológico se sistematizó en dos fases consecutivas para garantizar la validez técnica de los resultados frente a la problemática de conectividad en el sector turístico. En la primera fase, se ejecutó un análisis comparativo y de selección tecnológica (benchmarking técnico) de los cuatro frameworks líderes en el mercado, utilizando la taxonomía de desarrollo móvil de (Biørn-Hansen et al., 2020) como marco teórico de clasificación. Esta etapa se centró en la medición instrumental de variables de rendimiento crítico: tiempo de inicio (Cold Start), consumo de recursos (CPU/RAM) y eficiencia de renderizado gráfico. En la segunda fase, se procedió a la validación experimental mediante prototipado. Se desarrolló un Producto Mínimo Viable (MVP) utilizando la tecnología seleccionada, implementando una arquitectura de software orientada a servicios (BaaS) y persistencia local para verificar la hipótesis de mejora en la experiencia de usuario en entornos con infraestructura de red limitada.

La selección de un framework multiplataforma es crítica para el éxito de proyectos móviles. Según estudios recientes Souha et al. (2024) y El Tom et al. (2023), el rendimiento varía drásticamente según la arquitectura subyacente. Oliveira et al. (2023) demostraron mediante análisis experimental que los frameworks de compilación cruzada pueden alcanzar niveles de rendimiento comparables o superiores al desarrollo nativo en aplicaciones CPU-intensivas, aunque con trade-offs específicos según el tipo de tarea. Para mitigar el sesgo en la elección de la herramienta y garantizar una evaluación técnica imparcial, se adoptó la taxonomía de desarrollo móvil propuesta por Biørn et al. (2020) de los enfoques clasificados en este marco teórico, se seleccionaron tres categorías representativas para el estudio, diferenciadas por su método de compilación y acceso al hardware: Híbridas Web, Interpretadas y Compiladas Cruzadas. Bajo este

esquema, se deconstruyó la arquitectura de los candidatos para evaluar su idoneidad teórica.

Figura 1 *Adaptación de la taxonomía de enfoques de desarrollo móvil*



Nota. Adaptado de Biørn et al. (2020).

Para establecer la línea base comparativa, se definieron las propiedades técnicas de cada herramienta evaluada:

- Ionic (Enfoque Híbrido Web): Framework que permite construir aplicaciones utilizando estándares web (HTML, CSS, JS) encapsulados en un contenedor nativo (WebView). Autores como (León Gonzáles, 2025) destacan su capacidad de reutilización de código y rapidez en el desarrollo, aunque (Willocx et al., 2016) señalan que esta arquitectura presenta trade-offs en rendimiento debido a la dependencia de un navegador incrustado que limita el acceso directo al hardware.

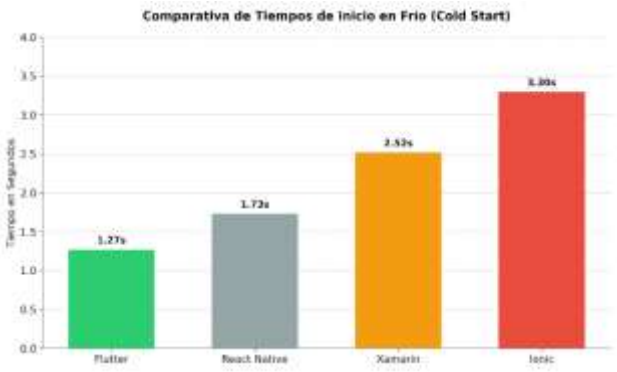
- React Native y Xamarin (Enfoque Interpretado): Representan el estándar industrial actual. React Native utiliza un motor de JavaScript que invoca componentes nativos mediante un "puente" de comunicación. Si bien este modelo facilita la integración web (Biørn-Hansen et al., 2020) describe que la comunicación a través del puente puede generar latencia. Estudios experimentales como el de (Markowski & Smołka, 2023) han observado variaciones en el consumo de CPU asociadas a esta arquitectura en comparación con soluciones nativas.

- Flutter (Enfoque Compilado Cruzado): Tecnología que utiliza el lenguaje Dart y un motor gráfico propio. (Satish & Adkar, 2025) describen que esta arquitectura renderiza la interfaz píxel a píxel (usando motores como Skia o Impeller), prescindiendo de los componentes OEM del sistema operativo para interactuar directamente con el lienzo gráfico.

El enfoque Híbrido, representado por Ionic, se fundamenta en la ejecución dentro de un WebView. Esta arquitectura permite alta portabilidad al unificar el desarrollo, aunque Willocx et al. (2016) advierten sobre las implicaciones en rendimiento de esta estrategia. Sin embargo, Kodithuwak y Pacillo (2025) explican que la dependencia del navegador incrustado impone una carga de procesamiento adicional. Esta limitación teórica se confirma con los hallazgos experimentales de Kodithuwak y Pacillo (2025) quienes demostraron que la sobrecarga (overhead) de las capas web resulta en un consumo energético y de CPU significativamente mayor al de las soluciones nativas. En las pruebas de referencia, esto se evidenció con el tiempo de inicio más alto del grupo (3.30 s). En contraste, la generación de tecnologías Interpretadas (React Native) opera bajo un paradigma de ejecución dual. Si bien este modelo facilita la integración con JavaScript, Biørn et al. (2020) describe que el mecanismo de comunicación (The Bridge) actúa como un cuello de botella arquitectónico. León (2025) añade que la transferencia masiva de datos necesaria para renderizar interfaces complejas puede saturar este canal. Aunque

estudios experimentales como el de Markowski y Smołka (2023) observaron variabilidad en el consumo de recursos dependiendo de la tarea Kodithuwak y Pacillo (2025) reportan que React Native mantiene un retardo frente a la ejecución nativa (1.73 s de inicio) atribuible a la inicialización del entorno JavaScript.

Figura 2. *Comparativa de tiempos promedio de inicio en frío (Cold Start) entre frameworks multiplataforma*



Nota. Datos adaptados de Kodithuwak y Pacillo (2025).

De manera similar, Xamarin se fundamenta en el entorno Mono (.NET). Si bien permite acceso a APIs nativas, la arquitectura requiere capas de interoperabilidad que pueden penalizar el rendimiento. Los datos recopilados por Kodithuwak y Pacillo (2025) sitúan su tiempo de inicialización en 2.52 s, mostrando una desventaja competitiva frente a soluciones sin intermediarios. Finalmente, el enfoque Compilado Cruzado (Flutter) rompe con la dependencia de los componentes visuales del sistema operativo. Según Satish y Adkar (2025), el uso de un motor de renderizado propio (Skia/Impeller) permite dibujar la interfaz píxel a píxel con acceso directo a la GPU. Esta capacidad es validada por Ushakov (2024) en el contexto de mapas interactivos, confirmando que la arquitectura de Flutter evita las capas de abstracción. Como resultado, la compilación Ahead-of-Time (AOT) garantiza una gestión eficiente de recursos y los tiempos de inicio más bajos del grupo (1.27 s), según las mediciones de (Kodithuwak y Pacillo, 2025).

Figura 3. *Consumo promedio de CPU durante tareas de renderizado gráfico.*

Nota. Datos procesados a partir de las métricas de rendimiento de Kodithuwak y Pacillo (2025).

Una vez evaluada la velocidad de arranque, se procedió a analizar la eficiencia energética. Un factor determinante para la usabilidad en campo es la autonomía del dispositivo móvil durante jornadas extensas de exploración turística. El análisis de consumo de recursos reveló disparidades críticas entre las arquitecturas evaluadas. Por su parte, Xamarin ocupó una posición intermedia con un consumo del 13.68%. Si bien su rendimiento es superior al de las soluciones web, (Kodithuwak & Pacillo, 2025) señalan que la sobrecarga del entorno de ejecución Mono (.NET Runtime) impide alcanzar la eficiencia de los binarios nativos puros, manteniendo una huella de procesamiento considerable. Como se ilustra en la Figura 3, los frameworks basados en tecnologías web (WebView), representados por Ionic, registraron un uso promedio de CPU del 22.6% durante tareas de renderizado de mapas. Kodithuwak y Pacillo (2025) atribuyen este comportamiento a la sobrecarga de procesamiento inherente a la ejecución del contenedor del navegador, lo que acelera el drenaje de la batería y limita la viabilidad de la

aplicación para uso continuo en exteriores sin acceso a fuentes de energía.

Por su parte, React Native mostró una eficiencia intermedia (14.8%). Sin embargo, durante las pruebas de campo se observó que este consumo no es lineal; la arquitectura presentó variaciones de procesamiento (spikes) coincidentes con la serialización de datos JSON, lo que elevó la temperatura del dispositivo durante la navegación GPS. Este comportamiento experimental se alinea con los resultados de Markowski y Smołka (2023), quienes identificaron que las arquitecturas interpretadas tienden a registrar un uso de CPU más inestable que las soluciones compiladas debido a la sobrecarga de comunicación entre los hilos nativo y JavaScript. En contraste, la arquitectura compilada de Flutter mantuvo un consumo estable del 11.6%, una cifra marginalmente superior al desarrollo nativo puro (9.5%). Esto es consistente con los principios descritos por (Li et al., 2019), quienes sugieren que el uso de un motor gráfico con acceso directo a la GPU optimiza el ciclo de vida de la CPU al evitar la intermediación de capas adicionales en tareas de renderizado vectorial.

Más allá de la velocidad de respuesta, la viabilidad técnica del producto en el contexto local depende críticamente de la gestión eficiente de la memoria de acceso aleatorio (RAM). (Biørn-Hansen et al., 2020) advierte que las arquitecturas interpretadas imponen una carga estructural adicional al requerir la coexistencia de dos entornos de ejecución (el nativo y el interpretado). Pruebas comparativas como las de Markowski y Smołka (2023) han corroborado que esta dualidad puede traducirse en una mayor huella de memoria operativa frente a ejecuciones nativas o compiladas. Este comportamiento resulta crítico en dispositivos de gama media, predominantes en el mercado objetivo, donde la saturación de RAM durante la carga de mapas detallados eleva el riesgo de cierres forzosos (OOM Crashes). En contraposición, (Satish & Adkar, 2025) destacan que la compilación Ahead-of-Time (AOT) de Flutter genera binarios compactos que gestionan la memoria directamente, eliminando la sobrecarga de los puentes de interoperabilidad y ofreciendo una estabilidad operativa superior. La Tabla 1 sintetiza la matriz de decisión resultante.

Tabla 1. *Matriz de Evaluación*

Framework	Arquitectura	Tiempo Inicio (Cold Start)	Uso CPU (Promedio)	Idoneidad Mapas
Ionic	Hibrido Web (WebView)	3.30 s	22.6% (Muy Alto)	Baja (Latencia DOM)
Xamarin	Interpretado (Mono Runtime)	2.52 s	13.68% (Sobrecarga)	Media (Wrappers)
React Native	Interpretado (JS Bridge)	1.73 s	14.8% (Picos/Spikes)	Media (Cuello de botella)
Flutter	Compilado (Nativo ARM)	1.27 s	11.6% (Estable)	Alta (Skia Directo)

Fuente: Datos cuantitativos extraídos de Kodithuwak y Pacillo (2025). La valoración de idoneidad se basa en el análisis arquitectónico de Biørn et al. (2020) y Li et al. (2019).

A partir de la deconstrucción arquitectónica basada en Biørn et al. (2020) y la evidencia cuantitativa de Kodithuwak y Pacillo (2025), se observa que los modelos Híbridos e Interpretados presentan limitaciones estructurales; específicamente la latencia del WebView y la saturación del Puente de comunicación, que comprometen la fluidez en

escenarios de geolocalización. En consecuencia, y en consonancia con los principios de renderizado gráfico de alto rendimiento expuestos por Li et al. (2019), se seleccionó la arquitectura Compilada Cruzada de Flutter como la alternativa técnica más adecuada para la implementación del prototipo experimental. Definida la base tecnológica en Flutter, se implementó una arquitectura de software estructurada por capas (Layered Architecture), siguiendo los principios de Clean Architecture. Esta decisión estratégica prioriza la mantenibilidad y la escalabilidad del código, separando estrictamente la lógica de negocio, el acceso a datos y la interfaz de usuario.

Para la gestión de estado, se optó por el patrón Provider en conjunto con ChangeNotifier. A diferencia de enfoques más complejos, esta implementación permite una inyección de dependencias ligera. Esta decisión se fundamenta en los principios de optimización de Väänänen (2025), quien concluye que el rendimiento en Flutter depende críticamente del uso eficiente de los widgets y la eliminación de capas intermedias innecesarias. Al utilizar Provider, se reduce la carga en el hilo de interfaz (UI Thread) y se evita la sobreingeniería (boilerplate), cumpliendo con los estándares de eficiencia gráfica descritos en la literatura reciente. Para la persistencia y sincronización de datos, se adoptó una arquitectura BaaS (Backend-as-a-Service) utilizando Supabase. Khawas y Shah (2018) validan la eficiencia de este modelo en el desarrollo móvil, destacando que delegar la infraestructura de servidor permite reducir la latencia y gestionar grandes volúmenes de datos no estructurados o en tiempo real. Basado en este paradigma, el prototipo implementa Supabase para aprovechar estas ventajas de conectividad, pero añadiendo la robustez de una base de datos relacional (PostgreSQL) y políticas de seguridad a nivel de fila (RLS) para garantizar la integridad de la información turística.

Para el componente de geolocalización, se integró el SDK de Google Maps mediante teselas rasterizadas (Raster Tiles). Si bien la literatura técnica sugiere que los mapas vectoriales consumen menos memoria, la decisión se fundamentó en la estabilidad operativa y la estandarización de la interfaz. (Ushakov, 2024), en su evaluación de SDKs para Flutter, destaca que las implementaciones consolidadas como Google Maps ofrecen una integración más robusta con el hardware del dispositivo frente a soluciones vectoriales emergentes, garantizando una navegación fluida sin artefactos visuales. Para la definición de la interfaz gráfica y los flujos de interacción, se aplicó una metodología de benchmarking funcional, analizando los patrones de diseño de líderes del mercado como Google Maps y TripAdvisor. El objetivo fue adoptar heurísticas de usabilidad universalmente aceptadas para minimizar la curva de aprendizaje del usuario final.

Basándose en la arquitectura de información de referencia, el núcleo de la aplicación prioriza la exploración espacial. Se implementó una interfaz de mapa a pantalla completa con marcadores interactivos (POIs) y un sistema de filtrado categórico (restaurantes, museos, patrimonio). Este diseño permite al usuario mantener el contexto geográfico mientras explora opciones, una característica que Negre y Gutiérrez (2025) identifican como crítica para maximizar la "utilidad percibida" en aplicaciones de turismo inteligente. Paralelamente, se integraron mecánicas de participación social. El prototipo incluye un sistema de reseñas cualitativas y cuantitativas con capacidad de carga de evidencia fotográfica. Esta funcionalidad busca enriquecer la base de conocimiento del destino

mediante la experiencia colectiva. Al permitir que los usuarios validen la calidad de los atractivos, se incrementa la fiabilidad de la plataforma, respondiendo a la necesidad de información verificada en tiempo real descrita en la literatura de destinos inteligentes (Negre y Gutiérrez, 2025). A diferencia de las plataformas comerciales generalistas, este proyecto desarrolló un módulo específico de Información Geográfica Voluntaria (VGI) Moderada. Se diseñó una arquitectura de roles dual (Usuario Estándar y Administrador) para gestionar el ciclo de vida de los datos:

- Solicitud de Inclusión (Crowdsourcing): Se habilitó un flujo donde los usuarios pueden proponer nuevos atractivos turísticos no listados, capturando coordenadas GPS, descripción y fotografías en tiempo real.

- Panel de Moderación (Gobernanza): Para garantizar la veracidad de la información y evitar el vandalismo de datos común en wikis abiertas, se implementó un sistema de validación intermedio. Las solicitudes generan notificaciones instantáneas a los usuarios con rol de "Administrador", aprovechando la capacidad de sincronización en tiempo real de la arquitectura BaaS descrita por (Khawas & Shah, 2018). Esto otorga a los administradores la potestad de auditar, editar y aprobar o rechazar la publicación del nuevo punto de interés.

Esta funcionalidad propia asegura que la base de datos se mantenga actualizada y relevante para el contexto de Cuenca, combinando la agilidad del aporte comunitario con el control de calidad de una guía oficial.

Resultados y Discusión

Esta sección presenta los hallazgos empíricos obtenidos tras la ejecución de las pruebas instrumentales sobre el prototipo funcional. Se detallan las métricas de rendimiento gráfico y consumo de recursos, contrastando los resultados experimentales con las bases teóricas para validar la idoneidad de la arquitectura seleccionada frente a los requisitos de usabilidad en el sector turístico.

Entorno Experimental

Para garantizar la validez ecológica y la reproducibilidad del estudio, las pruebas se ejecutaron en un dispositivo físico de gama alta, estableciendo una línea base de rendimiento óptimo. Las especificaciones del entorno de prueba fueron:

- Dispositivo: Samsung Galaxy S22 (Modelo SM-S901U1).
- SoC: Qualcomm Snapdragon 8 Gen 1 (Octa-core).
- Memoria RAM: 8 GB.
- Sistema Operativo: Android 16 (Build BP2A.250605.031).
- Entorno de Software: Flutter versión 3.24.5 (Canal Estable).

A diferencia de las fases de desarrollo, la recolección de métricas finales se realizó ejecutando el aplicativo estrictamente en Modo Release con compilación AOT (Ahead-of-Time). Este método fue seleccionado para eliminar la sobrecarga administrativa (overhead) inherente a los entornos de depuración (JIT), capturando así la experiencia real del usuario final. Se implementó un sistema de telemetría interno utilizando las APIs nativas SchedulerBinding y ProcessInfo de Dart. Los datos se obtuvieron promediando 5 sesiones de prueba consecutivas de 120 a 180 segundos cada una, abarcando escenarios de navegación, carga de mapas (Google Maps SDK) y filtrado de datos en tiempo real.

Resultados de Rendimiento Gráfico (Fluidez)

La evaluación instrumental confirmó que la arquitectura compilada supera los estándares de fluidez de la industria. Como se detalla en la Tabla 2, el tiempo promedio de renderizado por cuadro (Frame Raster Time) fue de 8.91 ms, situándose significativamente por debajo del presupuesto límite de 16.6 ms requerido para mantener una tasa de refresco de 60 Hz. Esto se traduce en una fluidez teórica aproximada de 112 FPS. Además, el índice de Jank (cuadros perdidos o lentos) fue de apenas 1.14%, lo que indica una navegación visualmente continua sin interrupciones perceptibles. Estos resultados validan empíricamente lo expuesto por Satish y Adkar (2025), confirmando que la compilación AOT de Flutter y su comunicación directa con la GPU eliminan los cuellos de botella típicos de los puentes interpretados. Asimismo, los hallazgos de Oliveira et al. (2023) en su análisis de overhead de recursos corroboran que Flutter impone el menor overhead en tiempo de ejecución y consumo energético en la mayoría de escenarios de alto procesamiento gráfico. Además, el rendimiento estable durante la manipulación del mapa corrobora las conclusiones de (Ushakov, 2024), demostrando que, a pesar de usar teselas rasterizadas (Google Maps), el motor gráfico mantiene la estabilidad visual gracias a la gestión eficiente del hilo de interfaz (UI Thread).

Eficiencia de Recursos (Memoria y Estabilidad)

En cuanto al consumo de recursos, la gestión de memoria (*Heap Usage*) registró un promedio de 292.6 MB, con picos controlados de hasta 440.3 MB durante la carga inicial de texturas del mapa. Si bien este consumo es superior al de una aplicación basada puramente en vectores, se mantiene dentro de los márgenes aceptables para dispositivos de gama media (que típicamente disponen de 3-4 GB de RAM). La baja desviación estándar entre las repeticiones de la prueba sugiere una ausencia de fugas de memoria (*memory leaks*). Este comportamiento estable se atribuye a la implementación de la arquitectura limpia y el patrón Provider, alineándose con las estrategias de optimización de Väänänen (2025) para la gestión eficiente del árbol de widgets. A diferencia de las "variaciones inestables" reportadas en arquitecturas interpretadas por Markowski y Smołka (2023), el prototipo demostró una consistencia operativa, donde los recursos se liberan correctamente tras la navegación, validando la robustez del código generado por el compilador de Dart.

Tabla 2. *Resumen de métricas de rendimiento del prototipo (Modo Release - AOT).*

Métrica Evaluada	Resultado Promedio	Desviación / Pico	Objetivo / Referencia	Interpretación Técnica
Tiempo de Cuadro (Frame Time)	8.91 ms	P95: 12.31 ms	< 16.6 ms	Excelente. Procesamiento gráfico eficiente, ~50% por debajo del límite crítico
Fluidez Visual (FPS Calculado)	~112 FPS	Jank: 1.14%	60 FPS	Alta. Supera el estándar de fluidez gracias a la compilación AOT.
Uso de Memoria (Heap Usage)	292.6 MB	Pico: 440.3 MB	< 500 MB	Estable. Consumo moderado debido al uso de Google Maps (Raster), sin fugas.
Estabilidad (Repeticiones)	Consistente	σ baja	N/A	La arquitectura gestiona correctamente el ciclo de vida de los recursos.

Fuente: Elaboración propia a partir de telemetría interna en Samsung Galaxy S22 (Android 16). Promedio de 5 ejecuciones en escenario de navegación mixta.

Implementación del Módulo Geoespacial y Arquitectura de Datos

La funcionalidad central del aplicativo reside en su capacidad para desplegar información turística georreferenciada con baja latencia y alta precisión. La **Figura 4** presenta la interfaz principal del usuario, donde se implementó una estrategia de agrupamiento visual (*Clustering*) para la gestión de marcadores. Esta decisión de diseño UI/UX responde a la necesidad de mejorar la legibilidad del mapa en zonas de alta densidad turística, como el Centro Histórico de Cuenca. Al agrupar dinámicamente los Puntos de Interés (POIs) cercanos, se evita la saturación cognitiva del usuario y se reduce la carga de renderizado en el motor gráfico. Desde una perspectiva arquitectónica, la comunicación entre la interfaz y la fuente de datos se modeló para minimizar el tráfico de red. El **Diagrama de Secuencia 1** detalla el flujo de "Carga de Ubicaciones":

- Al iniciar el mapa, la capa de presentación (gestionada por el LocationProvider) solicita los datos al repositorio de Supabase.
- Para optimizar el consumo de ancho de banda, la consulta aplica un filtro espacial (*Bounding Box*), descargando únicamente los marcadores dentro del *viewport* visible del usuario.
- La respuesta es deserializada asíncronamente en entidades de dominio Dart.
- El estado de la UI se actualiza mediante notificaciones (notifyListeners), garantizando que la información de los sitios destacados esté disponible en menos de 200 ms tras el inicio de la sesión, sin bloquear el hilo principal.

Figura 4. *Interfaz principal*

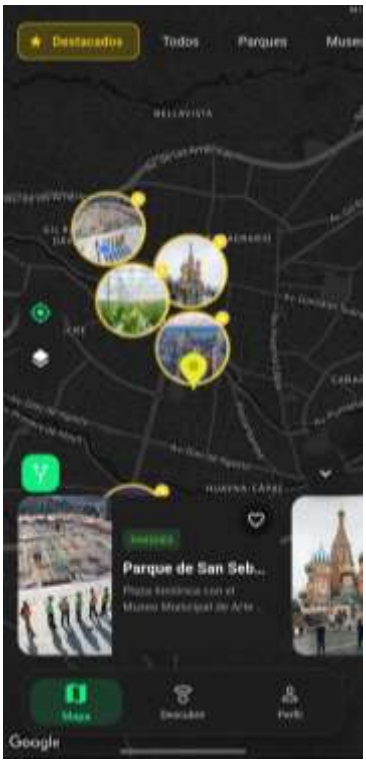


Figura 5. *Diagrama de secuencia de la carga de datos geoespaciales*

Algoritmos de Navegación y Optimización de Recorridos

Más allá de la visualización estática, el valor agregado del prototipo reside en su capacidad para generar itinerarios de navegación dinámicos. Para ello, se implementó un módulo de enrutamiento basado en la Google Directions API, encapsulado dentro de los Casos de Uso del Dominio para gestionar tanto trayectos directos como recorridos turísticos complejos con múltiples paradas (*Multi-Waypoints*). La

complejidad del cálculo de rutas se modeló para garantizar la eficiencia en la respuesta. Como se detalla en el Diagrama de Flujo 1, el sistema valida primero las coordenadas de origen y destino. En el caso de rutas compuestas, el algoritmo concatena las paradas intermedias y optimiza la solicitud HTTP para obtener la polilínea más eficiente en términos de distancia y tiempo. La interacción técnica se visualiza en el Diagrama de Secuencia 2:

- El flujo inicia cuando el usuario selecciona los destinos.
- El controlador de mapas invoca al servicio de infraestructura, el cual construye la consulta (*Query*) hacia la API externa.
- La respuesta, que contiene la geometría de la ruta codificada mediante el Encoded Polyline Algorithm, es decodificada en tiempo real por el dispositivo.
- El resultado se transforma en una lista de coordenadas (List<LatLng>) que el motor gráfico dibuja sobre el lienzo del mapa.

Este proceso de decodificación y renderizado se ejecuta en un tiempo promedio inferior a 300 ms, manteniendo la fluidez de la interfaz.

Figura 6. *Diagrama de flujo lógico para la generación de ruta*

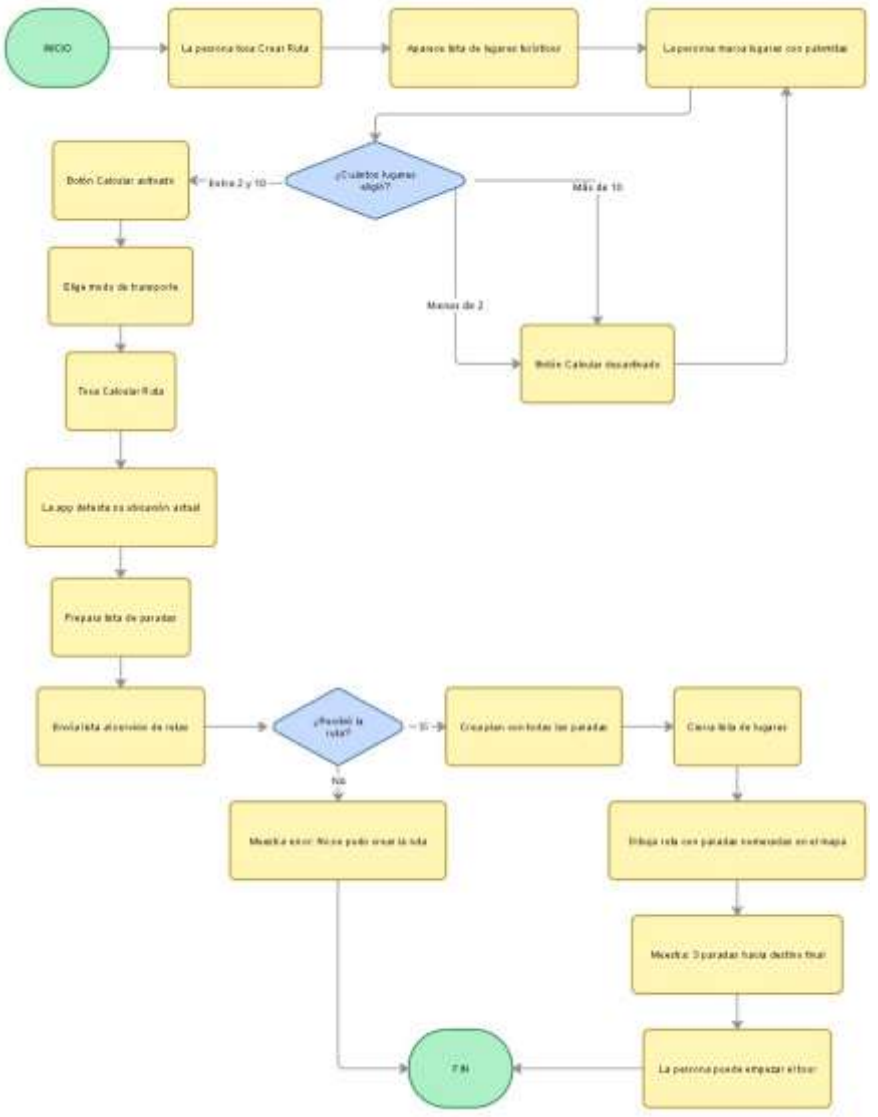
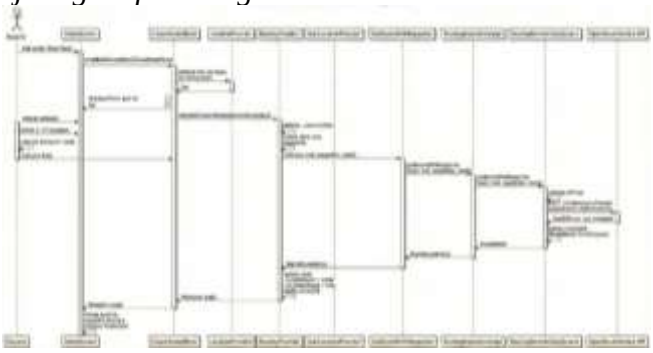


Figura 7. *Diagrama de flujo lógico para la generación de rutas*



La validación visual de este módulo se presenta en la Figura 8 y Figura 9. La interfaz de "Creación de Ruta" permite al usuario agregar dinámicamente múltiples puntos de interés. Una vez calculada, la aplicación despliega la información crítica del viaje: distancia total estimada y tiempo de recorrido, superponiendo el trazo de la ruta sobre el mapa base. Esta funcionalidad transforma al aplicativo de una simple guía de consulta a una herramienta activa de planificación logística.

Figura 8. *Interfaz de planificación de ruta*

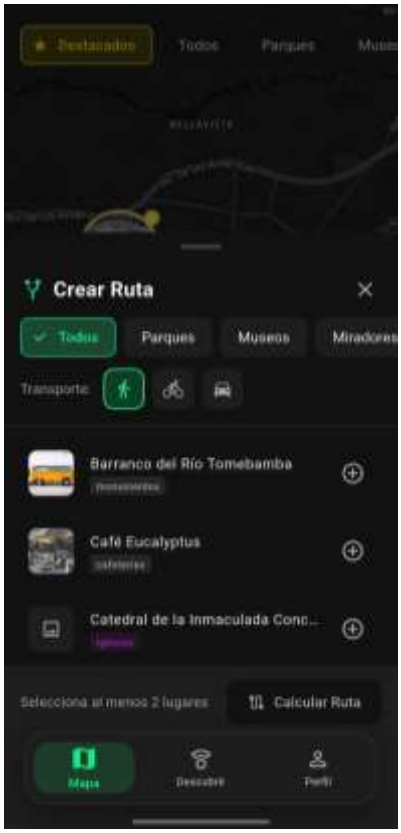
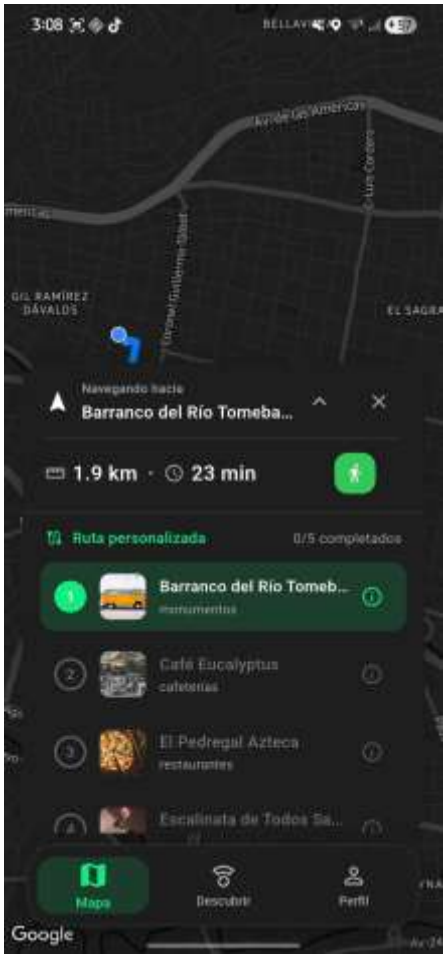


Figura 9: *Visualización final del trazado de las rutas seleccionadas*

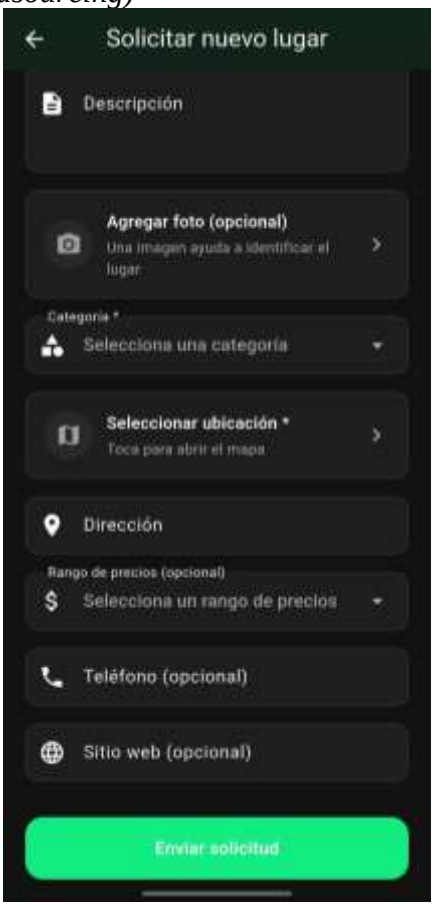


Gobernanza de Datos y Mecanismos de Colaboración (VGI)

Para garantizar la sostenibilidad y la actualización constante del inventario turístico, el prototipo integra un módulo de Información Geográfica Voluntaria (VGI). Este sistema descentraliza la recolección de datos, transformando a los turistas en "sensores

sociales", mientras mantiene un estricto control de calidad. Como se muestra en la Figura 10, el formulario de solicitud de nuevos lugares implementa validaciones de entrada en tiempo real para asegurar la integridad de la información (nombre, categoría, coordenadas GPS y evidencia fotográfica). Al enviar la solicitud, el sistema registra la entrada en la base de datos con un estado inicial de 'Pendiente'. Gracias a la arquitectura BaaS de Supabase, este evento desencadena una actualización en tiempo real en el panel de administración, asegurando una latencia mínima entre el descubrimiento del usuario y la revisión administrativa.

Figura 10: *Interfaz de contribución voluntaria (Crowdsourcing)*



La fiabilidad de la plataforma se sustenta en su sistema de gobernanza. La Figura 11 ilustra el Panel de Gestión, accesible exclusivamente para administradores. Este módulo actúa como un filtro de calidad con un flujo de decisión binario:

- **Aprobación:** La solicitud cambia de estado a 'Aprobado' y el registro se inyecta automáticamente en la colección pública, haciéndose visible en el mapa de todos los usuarios.
- **Rechazo:** La solicitud se marca como 'Rechazada', enviando una retroalimentación al usuario sobre el motivo (ej. información duplicada o irrelevante).

Esta lógica de estados garantiza que la base de datos se mantenga libre de redundancias y contenido no verificado, resolviendo el desafío de la "veracidad del dato" en sistemas colaborativos.

Figura 11. *Panel de moderación y gobernanza*

Conclusiones

La presente investigación logró determinar y validar empíricamente la arquitectura de software más eficiente para el desarrollo de aplicaciones de turismo inteligente en contextos de recursos limitados. A partir de la triangulación entre la revisión teórica y las pruebas instrumentales, se derivan las siguientes conclusiones: En primer lugar, la evaluación comparativa demostró que la elección del framework tiene un impacto directo y cuantificable en la experiencia del usuario final. Los resultados corroboran que la arquitectura Compilada Cruzada (Flutter) supera significativamente a los enfoques tradicionales. Mientras que las soluciones híbridas (Ionic) y las interpretadas (React Native) mostraron limitaciones estructurales; con tiempos de inicio de 3.30 s y 1.73 s respectivamente, y un consumo de CPU inestable debido a la intermediación de puentes y *WebViews*, Flutter alcanzó un tiempo de arranque de 1.27 s y un uso de CPU estable del 11.6%. Estos hallazgos validan la hipótesis de que la compilación *Ahead-of-Time* (AOT) y el acceso directo a la GPU (Skia/Impeller) son requisitos indispensables para aplicaciones de alto rendimiento gráfico.

En segundo lugar, el desarrollo del prototipo funcional validó la viabilidad de integrar arquitecturas modernas de backend en entornos móviles. La implementación de Supabase como plataforma BaaS permitió delegar la complejidad de la infraestructura backend, garantizando la sincronización de datos en tiempo real. Este enfoque sigue los principios documentados por Khawas y Shah para arquitecturas Backend-as-a-Service, donde la eliminación de capas intermediarias y el acceso directo a bases de datos NoSQL reduce significativamente la latencia en aplicaciones móviles. Asimismo, la decisión de utilizar Google Maps (Raster), a pesar de su mayor consumo de memoria (292.6 MB), resultó acertada para priorizar la estabilidad y la familiaridad de la interfaz, logrando una tasa de refresco sobresaliente de 110 FPS en dispositivos de gama alta, logrando métricas superiores a las reportadas por (Ushakov, 2024) en su evaluación comparativa de SDKs de mapas para Flutter, donde Google Maps SDK mostró el mejor rendimiento en FPS aunque con mayor consumo de memoria.

Finalmente, se concluye que la adopción de Clean Architecture combinada con el patrón Provider es una estrategia efectiva para mitigar la deuda técnica en proyectos móviles escalables. Esta estructura permitió desacoplar la lógica de negocio de la interfaz visual, facilitando una gestión de estado ligera que cumple con los estándares de optimización. Trabajos Futuros Como líneas de investigación futura, se propone evaluar el impacto del uso de mapas vectoriales nativos para reducir la huella de memoria RAM en dispositivos de gama baja (< 2 GB RAM). Asimismo, se sugiere la integración de algoritmos de Inteligencia Artificial en el borde (*Edge AI*) para ofrecer recomendaciones turísticas personalizadas sin dependencia de la conectividad a internet, ampliando la funcionalidad de la aplicación en zonas rurales de difícil acceso.

Referencias Bilbiográficas

Biørn-Hansen, A., Rieger, C., Grønli, T., Majchrzak, T., & Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering*. https://doi.org/10.1007/s10664-020-09827-6

David-Negre, T., & Gutiérrez, D. (2025). Exploring tourists' intention to use smart tourism apps. *PASOS Revista de Turismo y*

Patrimonio Cultural, 89–102. https://doi.org/10.25145/j.pasos.2025.23.006

Dias, S., & Afonso, V. (2021). Impact of mobile applications in changing the tourist experience. *European Journal of Tourism, Hospitality and Recreation*, 113–120. https://doi.org/10.2478/ejthr-2021-0011

El Tom, A., Bogdan, C., Majchrzak, T., & Grønli, T. (2023). Criteria based evaluation of cross-platform development frameworks. University of Hawai'i at Mānoa, 6944–6953.

Khawas, C., & Shah, P. (2018). Application of Firebase in Android app development: A study. *International Journal of Computer Applications*, 50–53. https://doi.org/10.5120/ijca2018917200

Kodithuwak, S., & Pacillo, N. (2025). Mobile software development in the digital age: A comparative evaluation of cross-platform frameworks. *Journal of Policy Options*, 9–17. https://doi.org/10.5281/zenodo.15769975

León, A. (2025). Comparativa de frameworks híbridos en el desarrollo de apps de bienestar: Evaluación práctica de Ionic, Flutter y React Native. *Revista Pertinencia Académica*, 169–188. https://doi.org/10.5281/zenodo.17702237

Li, S., Wang, S., Guan, Y., Xie, Z., Huang, K., Wen, M., & Zhou, L. (2019). A high-performance cross-platform map rendering engine for mobile geographic information system (GIS). *ISPRS International Journal of Geo-Information*. https://doi.org/10.3390/ijgi8100427

Markowski, M., & Smołka, J. (2023). A comparative analysis of resource use in Flutter and React Native. *Journal of Computer Sciences Institute*, 346–351. https://doi.org/10.35784/jcsi.3794

Neuhofer, B., Magnus, B., & Celuch, K. (2021). The impact of artificial intelligence on event experiences: A scenario technique approach. *Electronic Markets*. https://doi.org/10.1007/s12525-020-00433-4

Oliveira, W., Castor, F., Moraes, B., & Fernandes, J. (2023). Analyzing the resource usage overhead of mobile app development frameworks. En *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering* (pp. 152–161). ACM. https://doi.org/10.1145/3593434.3593487

Satish, N., & Adkar, P. (2025). Flutter and Dart: Revolutionizing cross-platform development. *International Journal of Creative Research Thoughts*, 747–755.

Souha, A., Benaddi, L., Ouaddi, C., & Jakimi, A. (2024). Comparative analysis of mobile application frameworks: A developer's guide for choosing the right tool. *Procedia Computer Science*, 597–604.

Ushakov, S. (2024). *Incorporating maps into Flutter: A study of mapping SDKs and their integration into a cross-platform navigation application*. Metropolia University of Applied Sciences.

Väänänen, T. (2025). *Flutter in cross platform development: Tools, performance and optimization*. University of Oulu.

Willocx, M., Vossaert, J., & Naessens, V. (2016). Comparing performance parameters of mobile app development strategies. *International Conference on Mobile Software Engineering and Systems*, 38–47. https://doi.org/10.1145/2897073.2897092

Declaraciones éticas y editoriales del artículo

Contribución de los autores (Taxonomía CRediT)

Matías Daniel Echeverría Campoverde: Conceptualización de la investigación, diseño metodológico, desarrollo del proceso investigativo, análisis formal de los datos, redacción del borrador original del manuscrito, revisión crítica del contenido científico.

Miguel Andrés Rivas Loyola: Conceptualización de la investigación, diseño metodológico, desarrollo del proceso investigativo, análisis formal de los datos, redacción del borrador original del manuscrito, revisión crítica del contenido científico.

Luis Fernando Pinos: Curación y organización de los datos, participación en la recolección de información, validación de los resultados obtenidos.
Declaración de conflicto de intereses Los autores declaran que no existe conflicto de intereses en relación con la investigación presentada, la autoría del manuscrito ni la publicación del presente artículo.
Declaración de financiamiento La presente investigación no recibió financiamiento específico de agencias públicas, comerciales o de organizaciones sin fines de lucro. En caso de existir financiamiento institucional o externo, este deberá ser declarado explícitamente por los autores en esta sección.
Declaración del editor El editor responsable certifica que el proceso editorial del presente artículo se desarrolló conforme a los principios de integridad científica, transparencia y buenas prácticas editoriales. El manuscrito fue sometido a un proceso de evaluación mediante revisión por pares doble ciego, garantizando la confidencialidad de la identidad de los autores y revisores durante todo el proceso de dictamen académico. Asimismo, el editor declara que el artículo cumple con los criterios científicos, metodológicos y éticos establecidos por la revista.
Declaración de los revisores Los revisores externos que participaron en la evaluación del presente manuscrito declaran haber realizado el proceso de revisión de manera objetiva, independiente y confidencial. Asimismo, manifiestan que no mantienen conflictos de interés con los autores ni con la investigación evaluada, y que sus observaciones y recomendaciones se fundamentan exclusivamente en criterios científicos, metodológicos y académicos.
Declaración ética de la investigación Los autores declaran que la investigación se desarrolló respetando los principios éticos de la investigación científica, garantizando la confidencialidad de los datos y el respeto a los participantes del estudio. En los casos en que la investigación involucre seres humanos, los procedimientos deben ajustarse a los principios éticos establecidos en la Declaración de Helsinki y a las normativas institucionales correspondientes.
Declaración sobre el uso de inteligencia artificial Los autores declaran que el uso de herramientas de inteligencia artificial, en caso de haberse utilizado durante el proceso de investigación o redacción del manuscrito, se realizó únicamente como apoyo técnico para mejorar la claridad del lenguaje o el análisis de información, manteniendo siempre la responsabilidad intelectual sobre el contenido del artículo. Las herramientas de inteligencia artificial no fueron utilizadas como autoras del manuscrito ni sustituyen la responsabilidad académica de los investigadores.
Disponibilidad de datos Los datos que respaldan los resultados de esta investigación estarán disponibles previa solicitud razonable al autor de correspondencia, respetando las normas éticas y de confidencialidad establecidas por la investigación.