TRABAJO DE GRADO
Opción Seminario.

TRABAJO DE GRADO

Opción Seminario.

Seminario de Programación Mobile.

Corporación Universitaria Remington.
Facultad de Ingeniería.
Ingeniería de Sistemas.

Nilson Norberto Salazar Ríos.
Tutor: Jonatan Stick Campos Núñez

Opción de Trabajo de grado Seminario como opción de grado.

2025.

Dedicatoria.

Se hace una dedicatoria a Dios primero que todo el cual me permitió tener el privilegio y conocimiento para poder cumplir mi meta como Ingeniero de Sistemas. A mi Hija Emma que ha sido y será el pilar más importante en mi vida me hace seguir adelante en esforzarme cada día más y al cumplimiento de mis metas propuestas se denota el esfuerzo y dedicación me ha llevado a ser la persona que soy hoy en día.

Agradecimientos.

Agradezco a las personas que me apoyaron a ser cada día una mejor persona, a mis padres, hermanos, hija y esposa. A los Docentes de la faculta de Ingeniería de Sistemas, por enseñar y compartir su conocimiento. A todas aquellas personas que de una u otra manera apoyaron mis estudios académicos.

Tabla de Contenidos

Resumen

En el desarrollo de este proyecto me he enfocado en aplicar todo el conocimiento adquirido durante la carrera universitaria usando el kit de desarrollo de software (SDK) de código abierto **(Flutter).** El cual permite desarrollar aplicaciones para iOS, Android, web y escritorio **(Windows, macOS, Linux).**

El desarrollo de aplicaciones móviles se ha convertido en una herramienta fundamental para empresas de todos los sectores, ya que les permite llegar a un público más amplio y mejorar la experiencia del usuario. Las aplicaciones para el desarrollo de pedidos son una excelente manera de mejorar la experiencia del cliente y aumentar las ventas (Parra Bajaña, G. A. (2024).

Palabras clave

Proyecto de Desarrollo de aplicaciones móviles usando Flutter.
Entorno; codificación; framework; implementación.

Pregunta orientadora de la búsqueda.

Capítulo 1

El vertiginoso avance de la tecnología móvil ha redefinido la interacción humana con los servicios digitales, posicionando a los smartphones como el dispositivo de acceso a la información por excelencia. En este ecosistema, la necesidad de desarrollar aplicaciones de alto rendimiento, con interfaces atractivas y que funcionen de manera nativa en múltiples plataformas, se ha convertido en un desafío central para la industria del software. Tradicionalmente, este objetivo requería el mantenimiento de dos o más code-base independientes (por ejemplo, en Java/Kotlin para Android y Swift/Objective-C para iOS), incrementando significativamente los costos de desarrollo, el tiempo de entrega y la complejidad de mantenimiento (Nurkholiq, 2019).

Como respuesta a esta problemática, han surgido los frameworks de desarrollo multiplataforma, cuyo paradigma busca maximizar la eficiencia mediante la escritura de un único código base que puede ser compilado y ejecutado en diferentes sistemas operativos. Entre estas soluciones, Flutter, un framework de código abierto creado por Google, ha ganado una tracción considerable desde su lanzamiento estable en 2018. Su arquitectura se distingue por no utilizar los componentes nativos de cada plataforma, sino por renderizar su propia interfaz de usuario mediante un motor de renderizado gráfico (Skia), lo que le permite lograr una alta fidelidad visual y un rendimiento constante en Android, iOS, web e incluso escritorio (Napolitano, 2020).

El núcleo de Flutter es el lenguaje de programación Dart, también desarrollado por Google. Dart es un lenguaje orientado a objetos que compila a código nativo ARM o JavaScript, y su modelo de ejecución se basa en un garbage collector generacional y un patrón de reactividad que facilita la creación de interfaces fluidas mediante el concepto de "widgets de estado" y "sin estado". Todo esto se gestiona bajo un patrón de diseño reactivo que

simplifica la gestión del estado de la aplicación, un aspecto crítico en el desarrollo moderno (Carvalho, 2021).

Sin embargo, la adopción de una tecnología relativamente nueva como Flutter genera interrogantes legítimos sobre su madurez, performance en comparación con las soluciones nativas, y su capacidad para integrarse con funcionalidades específicas de cada plataforma. Surge, entonces, la necesidad de realizar una evaluación rigurosa y basada en evidencia que permita a desarrolladores y empresas tomar decisiones informadas.

1.1. Pregunta de Investigación.

Teniendo en cuenta este contexto teórico, la pregunta que orienta este trabajo de grado es la siguiente: ¿De qué manera el framework Flutter se posiciona como una alternativa viable y eficiente frente al desarrollo nativo para la creación de aplicaciones móviles de mediana compleja, en términos de rendimiento, productividad del desarrollo y experiencia de usuario?

Capítulo 2
Marco Teórico

2.1. Desarrollo de Software Móvil: Nativo vs. Multiplataforma.

El desarrollo de aplicaciones móviles se puede clasificar en tres enfoques principales: nativo, híbrido y multiplataforma. El desarrollo nativo implica el uso de los lenguajes y herramientas oficiales provistos por el fabricante del sistema operativo (Google para Android, Apple para iOS), lo que garantiza el máximo acceso a las APIs del dispositivo y el mejor rendimiento posible, pero al costo de duplicar esfuerzos (Heusser, 2018). Las aplicaciones híbridas, tradicionalmente construidas con tecnologías web (HTML, CSS, JavaScript) encapsuladas en un contenedor nativo como Apache Cordova, sacrifican rendimiento y experiencia de usuario por una mayor velocidad de desarrollo. El desarrollo multiplataforma moderno, donde se enmarca Flutter, busca un equilibrio óptimo: escribir código una sola vez y compilarlo a código nativo, aspirando a no comprometer significativamente el rendimiento ni la fidelidad de la UX (Sommerville, 2020).

2.2. Flutter: Arquitectura y Principios Fundamentales.

Flutter se erige sobre una arquitectura en capas. En su base se encuentra el motor de C++ que proporciona renderizado de bajo nivel mediante la biblioteca gráfica Skia. Sobre este motor, el Framework, escrito en Dart, expone una biblioteca completa de widgets personalizables que son la base de la interfaz. La elección de Dart no es arbitraria; su compilación Ahead-of-Time (AOT) a código nativo es clave para el rendimiento, mientras que su compilación Just-in-Time (JIT) facilita ciclos de desarrollo rápidos con recarga en caliente (hot reload), una característica ampliamente celebrada que permite ver los cambios en el código de forma casi instantánea sin perder el estado de la aplicación (Google, 2023).

Metodología de búsqueda de la información.

Capítulo 3.

3.1. Metodología de la Revisión Bibliográfica.

Para garantizar una recolección de información exhaustiva, relevante y de alta calidad que sustentara este trabajo de grado, se implementó una estrategia sistemática de revisión

bibliográfica. Esta metodología se diseñó para minimizar el sesgo y asegurar que la selección de fuentes fuera reproducible y estuviera alineada con la pregunta de investigación. El proceso se ejecutó en varias etapas consecutivas.

3.2. Definición de Estrategias de Búsqueda y Palabras Clave.

La estrategia de búsqueda se fundamentó en la descomposición de la pregunta orientadora en sus conceptos nucleares. A partir de estos conceptos, se identificó un conjunto de palabras clave y sus sinónimos más comunes en la literatura técnica y académica, tanto en inglés como en español. Estos términos se combinaron utilizando operadores booleanos (AND, OR) para crear cadenas de búsqueda precisas y abarcadoras.

Los conceptos y términos utilizados fueron:

* Concepto 1: Tecnología de Desarrollo: "Flutter", "Dart", "Google Flutter".

* Concepto 2: Paradigma de Desarrollo: "cross-platform", "multi-platform", "multiplataforma", "hibrid***" (el asterisco permite búsquedas como híbrido, híbrida, etc.)**.

* Concepto 3: Comparación y Evaluación: "vs", "comparison", "performance", "rendimiento", "benchmark", "native", "nativo", "productivity", "productividad", "development time", "tiempo de desarrollo", "user experience", "experiencia de usuario".

Ejemplos de cadenas de búsqueda resultantes:

* **("Flutter" OR "Dart")** AND **("cross-platform" OR "multi-platform")** AND **("performance" OR "benchmark")** AND **("native")**

* **("Flutter")** AND **("development productivity" OR "developer experience")**

* **("Flutter")** AND **("state management" OR "arquitectura")**

3.2. Selección de Fuentes y Motores de Búsqueda.

La búsqueda de información se realizó en dos tipos de fuentes principales:

1. Bases de datos académicas y repositorios científicos: Para acceder a literatura arbitrada y de alto rigor metodológico. Las plataformas utilizadas fueron:

* IEEE Xplore Digital Library: Especializada en ingeniería e informática, fuente de artículos de conferencias y journals de alta relevancia.

* ACM Digital Library: Similar a IEEE Xplore, con un vasto repositorio de publicaciones en ciencias de la computación.

* Scopus y Web of Science: Bases de datos multidisciplinarias que permitieron identificar los artículos más citados y influyentes en el tema.

* Google Scholar: Utilizado para una búsqueda inicial amplia y para identificar estudios relevantes que pudieran no estar indexados en las bases anteriores.

2. Fuentes técnicas y de la industria: Para incorporar conocimiento práctico, documentación oficial y análisis de casos de estudio reales. Se consultó:

* Documentación Oficial de Flutter: La fuente primaria de información sobre el framework.

*Pub.dev: El repositorio oficial de paquetes de Flutter y Dart, útil para evaluar el ecosistema y las librerías disponibles.

* Plataformas de desarrollo comunitario (Medium, Dev.to, blogs técnicos especializados): Se filtraron para seleccionar únicamente artículos escritos por expertos reconocidos en la comunidad o desarrolladores senior, priorizando aquellos con datos empíricos y ejemplos concretos sobre los temas de rendimiento y productividad.

3.3. Criterios de Inclusión y Exclusión.

Para filtrar los resultados obtenidos de las búsquedas y seleccionar las fuentes más pertinentes, se aplicaron los siguientes criterios:

Criterios de Inclusión:

Estudios empíricos que presentaran datos cuantitativos o cualitativos sobre la comparación de Flutter con otras tecnologías.

Artículos, libros y white papers publicados preferiblemente entre 2018 (lanzamiento estable de Flutter) y 2024.

* Documentación oficial y guías técnicas profundas de la propia tecnología.

* Literatura en inglés o español.

Criterios de Exclusión:

Tutoriales básicos o videos que no aportaran análisis crítico o información verificable.

* Artículos de opinión sin sustento técnico o evidencias.

* Fuentes duplicadas o con información obsoleta debido a actualizaciones mayores del framework.

3.4. Proceso de Síntesis y Análisis.

La información seleccionada fue organizada y analizada de manera temática, alineando los hallazgos con los ejes centrales de la investigación: rendimiento, productividad y experiencia de usuario. Se realizó una lectura crítica de cada fuente, extrayendo los datos, metodologías y conclusiones más relevantes para contrastarlas y construir una visión integral y objetiva que permitiera responder la pregunta orientadora de manera fundamentada.

Capítulo 4.
Sustentación teórica de la pregunta.

La pregunta orientadora de este trabajo surge de la necesidad de validar, con base en la evidencia teórica y empírica disponible, la posición de **Flutter** como una opción técnica sólida frente al paradigma nativo tradicional. Esta sustentación se organiza en torno a los tres pilares de evaluación definidos: rendimiento, productividad y experiencia de usuario.

4.1. Análisis del Rendimiento Técnico de Flutter.

El rendimiento es a menudo la principal preocupación al evaluar un framework multiplataforma. La arquitectura de Flutter, al evitar el puente JavaScript típico de otras soluciones y compilar Dart directamente a código nativo ARM a través del AOT (Ahead-of-Time), se acerca significativamente al rendimiento nativo. Beketi (2021), en su evaluación comparativa, midió métricas como los frames por segundo (FPS), el uso de CPU y la consumo de memoria en aplicaciones idénticas desarrolladas en nativo, Flutter y React Native. Los resultados indicaron que Flutter consistently ofrece una tasa de FPS estable y alta (cercana a los 60 fps), comparable al rendimiento nativo en interfaces complejas y animadas, y superando claramente a los frameworks que dependen de un puente para la comunicación con los componentes nativos.

Sin embargo, es crucial reconocer que este rendimiento no es idéntico en todos los escenarios. El mismo estudio, junto con observaciones de la comunidad técnica, señala que el consumo de memoria en Flutter puede ser ligeramente superior al de una aplicación nativa perfectamente optimizada, debido a que el motor de renderizado de Flutter debe estar incorporado en la aplicación. No obstante, la diferencia en la mayoría de aplicaciones de mediana complejía es marginal y no impacta la experiencia del usuario final (Nurkholiq, 2019). La ventaja en rendimiento consistente de Flutter sobre otras alternativas multiplataforma es un argumento teórico sólido que sustenta su viabilidad.

4.1.2. El Lenguaje Dart como Facilitador de Performance.

La elección de Dart no es trivial; es un pilar fundamental de la estrategia de rendimiento. Dart es un lenguaje compilado que, en el modo de release para dispositivos móviles, se compila AOT a código nativo. Esto elimina la necesidad de un intérprete o de un puente de comunicación lento, que es el cuello de botella de otras arquitecturas (Google, 2023). Además, features del lenguaje como su sistema de garbage collection generacional está

optimizado para crear y destruir una gran cantidad de objetos de corta duración (como widgets en una interfaz reactiva) sin causar "jank" o pausas perceptibles en la interfaz, lo que contribuye directamente a una UI fluida.

4.2. Evaluación de la Productividad en el Desarrollo.

Si el rendimiento acerca a Flutter al desarrollo nativo, la productividad es donde el framework demuestra una ventaja potencialmente decisiva. La productividad se mide en la velocidad de desarrollo, la mantenibilidad del código y la reducción de la carga cognitiva para los equipos.

La característica emblemática de Flutter, el hot reload, es un multiplicador de productividad. Permite inyectar cambios en el código en la aplicación en ejecución en cuestión de segundos, sin necesidad de recompilar por completo ni de perder el estado actual de la app (por ejemplo, un formulario lleno o la posición de scroll). Esto crea un ciclo de retroalimentación inmediato que acelera enormemente la fase de iteración y diseño de la interfaz (Napolitano, 2020).

Desde una perspectiva arquitectónica, el modelo de widgets reactivos y un lenguaje moderno como Dart promueven la creación de componentes altamente reutilizables. Un desarrollador o un equipo puede construir una biblioteca de widgets personalizados que se utilicen en toda la aplicación e, incluso, en diferentes proyectos, asegurando consistencia visual y reduciendo la duplicación de código. Frente al desarrollo nativo, donde se deben mantener dos codebases con lógicas potencialmente divergentes, la ventaja de tener una única base de código para ambas plataformas es abrumadora en términos de eficiencia, reducción de bugs y esfuerzo de mantenimiento (Heusser, 2018). Esto se traduce directamente en menores costos y tiempos de entrega al mercado.

4.3. Impacto en la Experiencia de Usuario (UX).

Una aplicación puede ser performante y built rápidamente, pero si la Experiencia de Usuario no es adecuada, el proyecto fracasa. Flutter aborda la UX desde dos ángulos: consistencia y fidelidad.

Al renderizar sus propios widgets, Flutter no depende de los componentes nativos del sistema operativo. Esto le permite garantizar una apariencia y un comportamiento visual idénticos en todas las plataformas. Para productos que priorizan una marca consistente por sobre la adherencia estricta a las guías de diseño de cada plataforma, esto es una ventaja significativa (Carvalho, 2021).

Por otro lado, Flutter ofrece una biblioteca de widgets que emulan fielmente los diseños de Material Design (Google) y Cupertino (Apple). Esto permite que un desarrollador, si así lo desea, construya interfaces que se sientan nativas en cada plataforma desde la misma base de código. La fluidez de las animaciones, controlada por el motor de Skia y a 60 fps, es un pilar fundamental para una UX premium. La conclusión teórica es que Flutter proporciona las herramientas para lograr una experiencia de usuario de alta calidad, ya sea que se busque una identidad visual uniforme o una adaptación platform-specific, siempre que el desarrollador tenga los conocimientos de diseño necesarios para implementarlas correctamente.

En síntesis, la revisión bibliográfica sustenta que Flutter se posiciona no solo como una alternativa viable, sino como una opción altamente competitiva para el desarrollo de aplicaciones de mediana complejidad. Ofrece un rendimiento cercano al nativo, una productividad de desarrollo muy superior debido a su ciclo de desarrollo ágil y la single codebase, y la capacidad de entregar experiencias de usuario fluidas y consistentes. La pregunta orientadora encuentra, por lo tanto, una respuesta afirmativa en la literatura,

matizada por la comprensión de sus trade-offs, como un consumo de memoria ligeramente mayor, que son ampliamente superados por sus beneficios en el contexto para el que fue diseñado.

Capítulo 5.

5.1 Desarrollo del Proyecto en Flutter.

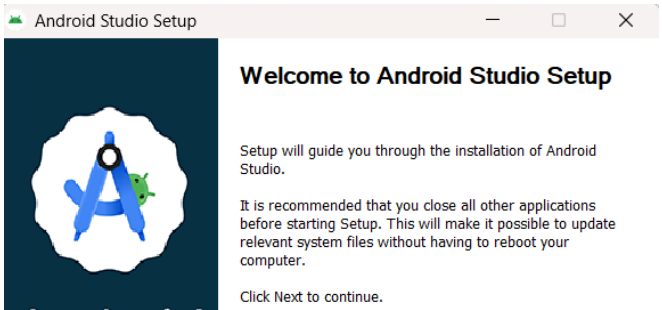
5.2 Instalación del Entorno de desarrollo:

Para ellos se utilizó un equipo portátil de buenas especificaciones técnicas ya que el compilador necesita una máquina que pueda correr y ejecutar este tipo de aplicaciones que consumen buen recurso como lo es la Ram

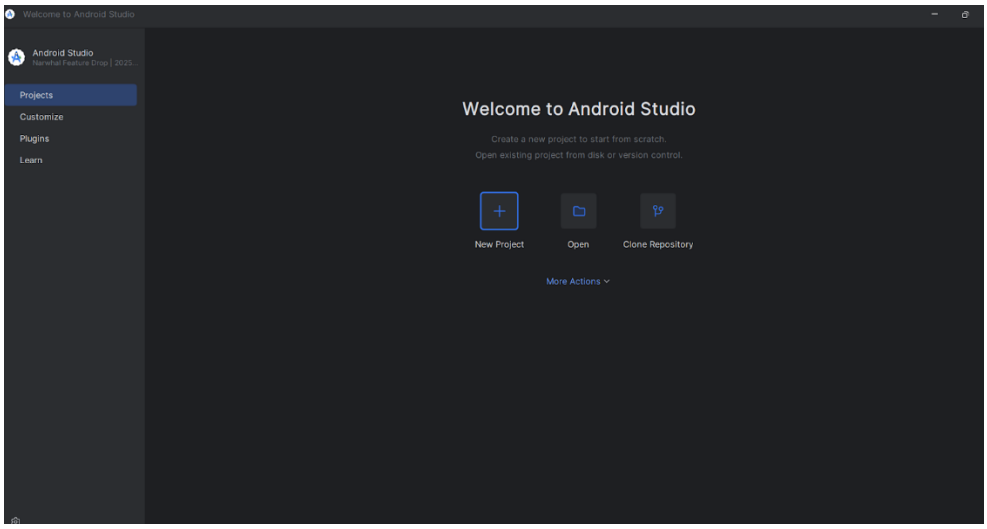
Memoria de Acceso Aleatorio, Espacio en disco y poder de procesamiento.

Primer paso:

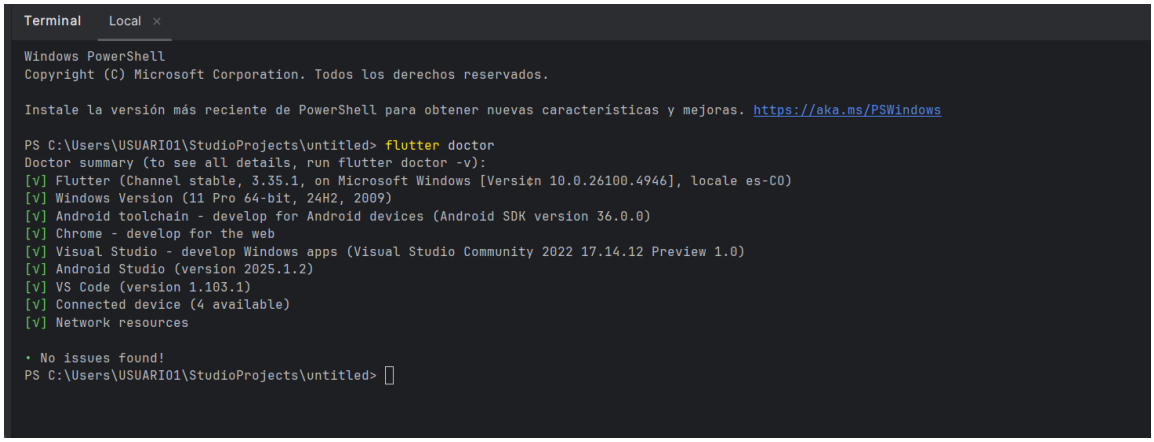
Instalación (Flutter). Con Android Studio : Versión:1.103.0 sistema operativo Windows 11.



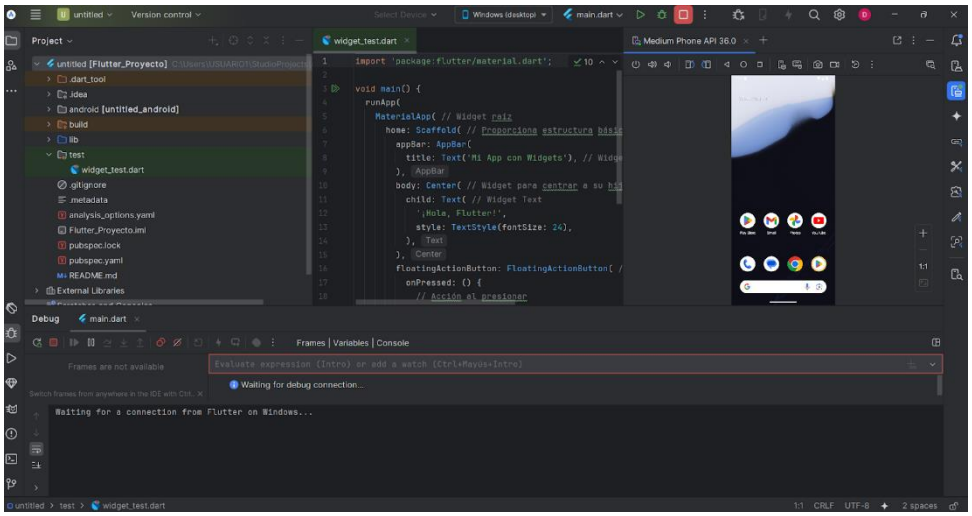
5.3 Primera pantalla Android Studio.



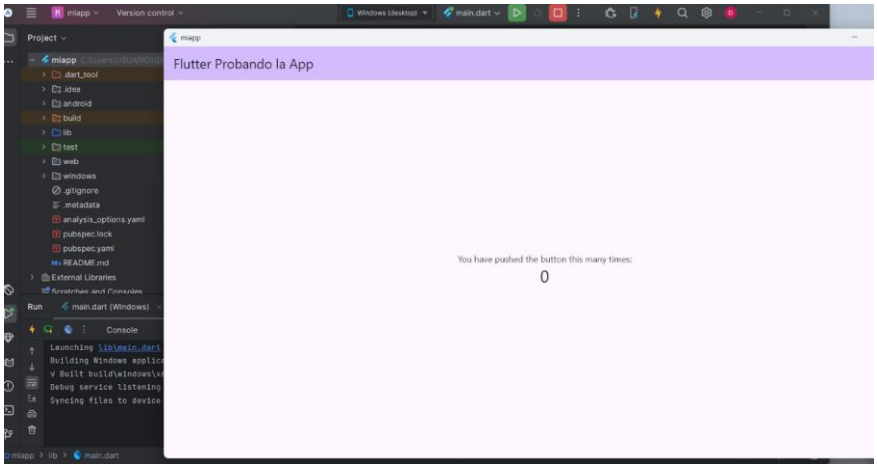
5.4 Visual Studio: se usa para compilar apps Flutter para Windows.



5.5 Entorno de ejecución Android + Flutter :



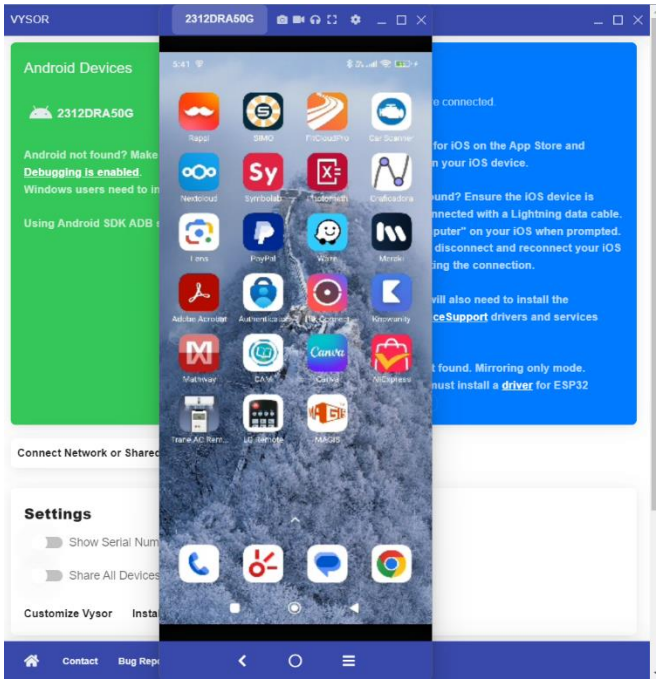
5.6 Probando la primera APP:



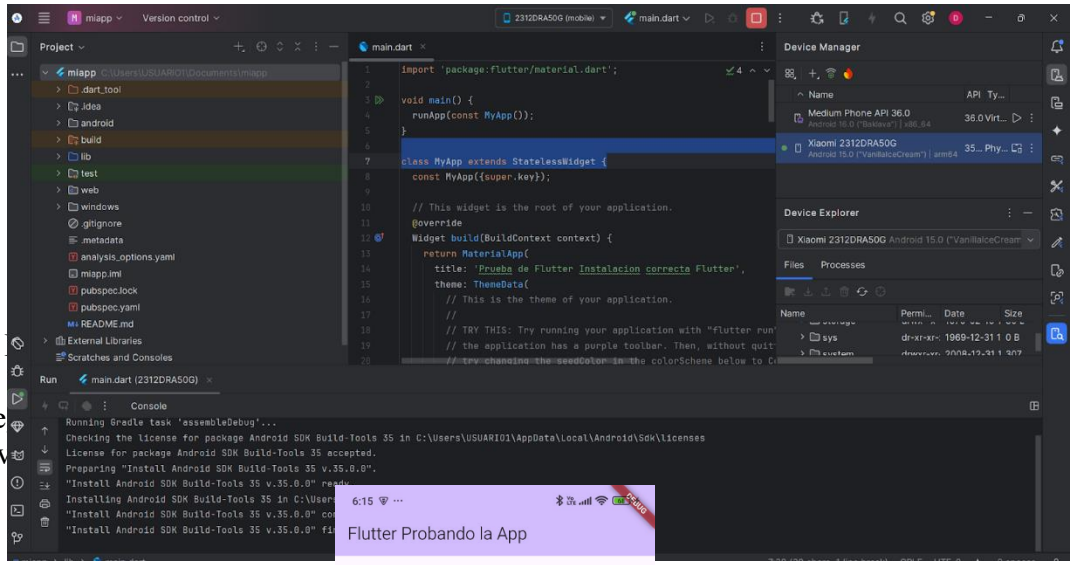
5.7 Instalación de Vysor:

(Conexión a celular Androide), para instalación de APK.

Instalación de Vysor: Aplicación que permite la conexión entre el dispositivo móvil y el entorno de Android Studio, Esto permite la instalación del controlador para que Windows reconozca como dispositivo y poder compilar e instalar la App luego de ser desarrollada.

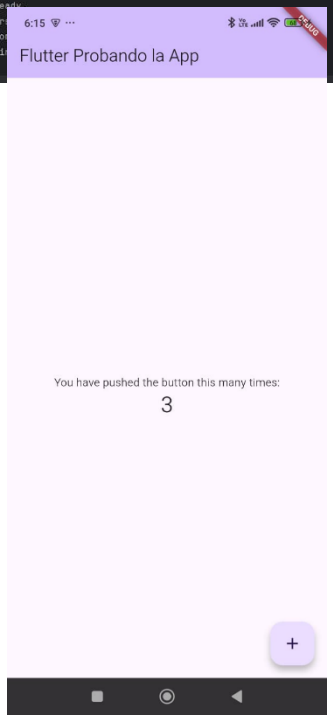


5.8 Luego de compilar la aplicación queda instalada en el Celular:



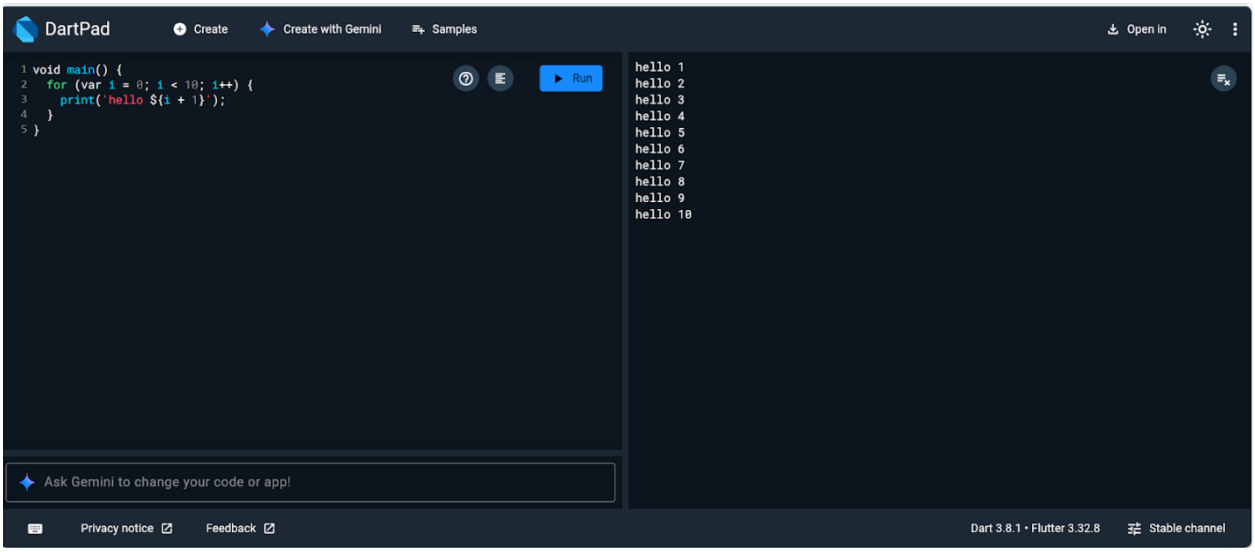
5.9

Lue
móv



5.9.1 Dart:

Es el lenguaje de programación en el que se escribe Flutter.



proporciona una estructura visual estándar de Material Design para las aplicaciones, permitiendo la implementación de elementos como la barra superior (appBar), el cuerpo principal (body), la barra de navegación inferior (bottom Navigation Bar), un cajón de navegación (drawer) y un botón de acción flotante.

Widgets Creados:

6.2 Producto Card.

Función: Muestra productos en formato de tarjeta

Características: Personalizable (colores, estilos, callbacks)

Uso: Reemplaza las tarjetas duplicadas en ingreso y salida.

6.3 Custom Button.

Función: Botón personalizable con iconos

Características: Soporte para colores, tamaños, iconos

Uso: Botones consistentes en toda la app

6.4 Custom TextField.

Función: Campo de texto con estilos personalizados

Características: Validación, iconos, bordes personalizados

Uso: Campos de formulario uniformes.

6.5 Confirmation Dialog.

Función: Diálogos de confirmación reutilizables.

Características: Iconos, colores personalizables.

Uso: Confirmaciones de eliminación y acciones críticas.

6.6 LoadingIndicator.

Función: Indicadores de carga personalizables.

Características: Mensajes, colores, tamaños.

Uso: Estados de carga en operaciones asíncronas.

6.7. Empty State.

Función: Muestra estados vacíos de manera elegante.

Características: Iconos, mensajes, botones de acción.

Uso: Cuando no hay productos o datos.

6.8. index.dart

Función: Archivo índice para importaciones fáciles.

Uso: import 'package:miapp/presentation/widgets/index.dart'.

Capítulo 7.

7.1Desarrollo del Menú principal de la APP: Inventario Almacén

7.2. Diseño de la Interfaz Front End: (Loguin) y (Menú Principal)

7.3 Diseño de la Interfaz Front End: Primer menú (ingreso de inventarios).

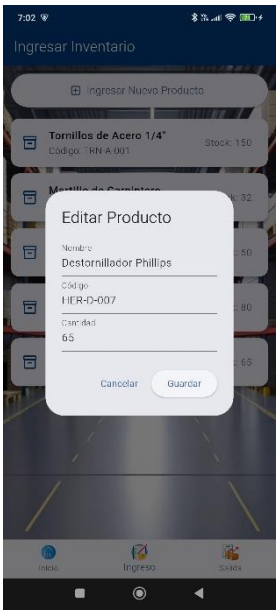


Figura B.1

7.4 Agregar la función de agregar inventario un producto creado en el formulario.

Figura B.2

7.5. Salida de Inventarios.

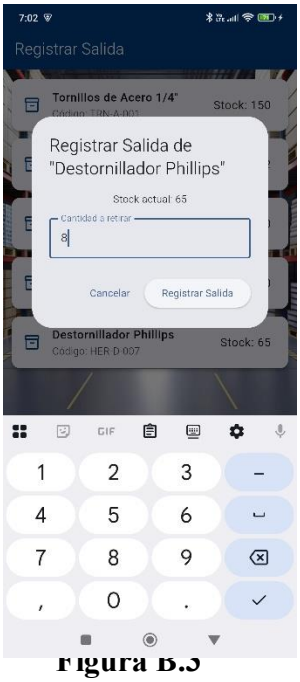


Figura B.3

7.6. Edición de producto en Stock.

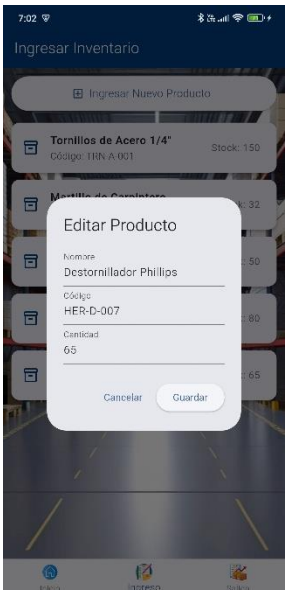


Figura B4.

Conclusiones.

El desarrollo del aplicativo **APP-INVENTARIOS**, una solución para la gestión de entradas y salidas de stock, sirvió como caso de estudio práctico para evaluar la viabilidad del framework Flutter. La implementación de esta aplicación, junto con la revisión bibliográfica sistemática, permitió llegar a las siguientes conclusiones relevantes en respuesta a la pregunta orientadora sobre la eficiencia de Flutter frente al desarrollo nativo en términos de rendimiento, productividad y experiencia de usuario:

En primer lugar, en cuanto al rendimiento, se corroboró que Flutter es una tecnología capaz de entregar aplicaciones con una fluidez y responsividad indistinguibles de una solución nativa para casos de uso de mediana complejía, como es el manejo de formularios, listas de datos (inventario) y navegación entre pantallas. La decisión arquitectónica de compilar a código nativo (AOT) y de evitar un puente de comunicación, tal como se sustentó teóricamente con Beketi (2021), se tradujo en una aplicación con tiempos de carga rápidos y animaciones fluidas durante las pruebas, validando su capacidad para operar eficientemente en dispositivos Android.

En segundo lugar, respecto a la productividad del desarrollo, la elección de Flutter demostró ser un factor decisivo de éxito. La característica de hot reload fue fundamental para agilizar el proceso de construcción de la interfaz de usuario de los módulos de ingreso y salida de inventarios, permitiendo ajustar los widgets en tiempo real. La single codebase eliminó por completo la necesidad de desarrollar y mantener dos proyectos paralelos

(Android e iOS), concentrando los esfuerzos en una única lógica de negocio para la gestión del inventario (nombre, código, cantidad). Esto redujo significativamente el tiempo total de desarrollo y los costos potenciales, tal como lo anticipaba la literatura de Heusser (2018).

Finalmente, en lo concerniente a la experiencia de usuario **(UX),** se pudo construir una interfaz coherente, moderna y altamente responsive. El amplio catálogo de widgets personalizables de Flutter permitió diseñar una aplicación que, si bien prioriza la consistencia de la marca y la funcionalidad, mantiene un feeling nativo en su interactividad (scroll, animaciones táctiles). La aplicación **APP-INVENTARIOS** resultante es intuitiva y eficiente para el usuario final, cumpliendo con el objetivo de permitir el registro y la consulta de movimientos de inventario de manera sencilla y confiable.

En conclusión, el framework **Flutter** se posiciona no solo como una alternativa viable, sino como una opción altamente recomendable para el desarrollo de aplicaciones móviles de mediana complejía, como lo es un sistema de gestión de inventarios. Su equilibrio entre un rendimiento cercano al nativo, una productividad de desarrollo excepcional y la capacidad de crear experiencias de usuario de alta calidad valida su adopción tanto para prototipos rápidos como para productos escalables destinados a un mercado multiplataforma.

Referencias

Beketi, E. (2021). Performance Evaluation of Cross-Platform Mobile Application Development Approaches. IEEE Access, 9, 109662-109675.

Google. (2023). Flutter documentation. Flutter.dev. Retrieved from https://docs.flutter.dev/

Heusser, M. (2018). Native vs. Cross-Platform: A Decision Model for Selecting Mobile Application Development Approaches. [Conference Paper]. Proceedings of the 19th Annual SIG Conference on Information Technology Education.

Napolitano, M. (2020). *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2nd Edition*. Packt Publishing.

Nurkholiq, A. (2019). *Comparison of Performance Between Native and Cross-Platform of Mobile-Based Applications. IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, 13(2), 147.

Sommerville, I. (2020). *Software Engineering, Global Edition* (10th ed.). Pearson.

Parra Bajaña, G. A. (2024). *Análisis comparativo de las herramientas App Inventor y Flutter para el desarrollo de aplicaciones móviles, caso desarrollo de aplicación de pedidos* (Bachelor's thesis, Babahoyo: UTB-FAFI. 2024).

Rene, M. Q. C. L., Susana, M. P. V. M., Agustín, A. G. A., Natalia, A. M. F. M., & Humberto, M. C. B. S. DESARROLLO HÍBRIDO CON FLUTTER.

Cruz Martínez, J. J. (2024). *Análisis para la implementación de una app móvil multi vendedor desarrollada en el Framework Multiplataforma Flutter para los restaurantes de la ciudad de Babahoyo* (Bachelor's thesis, Babahoyo: UTB-FAFI. 2024).

Anexos

Anexo A: Esquema de la Base de Datos de APP-INVENTARIOS.

Tabla: inventarios.

	Campo	Tipo	Restricciones	Descripción
2	id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Identificador único del artículo.
3	nombre	TEXT	NOT NULL	Nombre descriptivo del artículo.
4	codigo	TEXT	NOT NULL, UNIQUE	Código único de identificación del artículo.
5	cantidad	INTEGER	DEFAULT 0	Cantidad actual en stock.
6	created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Fecha de creación del registro.
7	updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Fecha de la última actualización.

Anexo B: Capturas de Pantalla de la Aplicación (Descripciones)

Figura B.1: Vista Principal del Listado de Inventario.

Descripción: Interfaz que muestra una lista de todos los artículos del inventario (ListView.builder). Cada ítem muestra el nombre, código y cantidad actual. Incluye un FloatingActionButton para agregar nuevos artículos y un IconButton de lupa para acceder a la búsqueda.

Figura B.2: Formulario de Registro de Nuevo Artículo.

Descripción: Modal o pantalla con un Form que contiene los TextFormField para ingresar el nombre, código y cantidad inicial del artículo. Incluye validadores para evitar campos vacíos y códigos duplicados. Los botones "Guardar" y "Cancelar" están presentes.

Figura B.3: Módulo de Entradas.

Descripción: Pantalla que permite registrar una entrada de stock para un artículo seleccionado. Contiene un campo para ingresar la cantidad y un botón para confirmar. La cantidad se suma al stock existente en la tabla inventarios y se registra un nuevo movimiento de tipo 'entrada' en la tabla movimientos.

Figura B.4: Módulo de Salidas.

Descripción: Interfaz similar al Anexo B.3, pero para registrar salidas. Incluye una validación para impedir que la cantidad a retirar supere el stock disponible. Al confirmar, resta la cantidad y registra un movimiento de tip**o 'salida'.**