

**Universidad de Los Andes**

**Facultad de Ingeniería**

**Minería de repositorios: Benchmark Kotlin Multiplatform, React Native, Flutter**

**Asesor:**

**Camilo Andrés Escobar Velásquez**

**Trabajo de Pregrado 2024-1**

**Estudiante:**

**Santiago Hernández Facio Lince**

**201922432**

**Repositorio del proyecto: https://github.com/shernandezf/Tesis2024**

**Resumen**

En el intrincado y vasto universo del desarrollo de aplicaciones móviles, este documento pretende servir como una guía, más conocida como benchmark, esencial para los desarrolladores de aplicaciones móviles, enfocándose en tres de los frameworks de programación más usados para el desarrollo de aplicaciones híbridas. El objetivo es realizar una comparativa exhaustiva y evaluativa de Kotlin Multiplatform, React Native y Flutter, basada en una serie de métricas relevantes que permitan identificar las fortalezas y debilidades inherentes a cada lenguaje. A través de este análisis, buscamos facilitar la selección del framework de programación más adecuado para futuros proyectos, asegurando una decisión informada y estratégica que responda a las necesidades específicas del desarrollador.

**Palabras clave**

*Benchmark, Aplicaciones Móviles, Framework, React native, Kotlin, Flutter.*

**Introducción**

Ante la creciente diversificación de fabricantes de dispositivos y sistemas operativos para dispositivos móviles, los desarrolladores se encuentran ante un dilema tecnológico significativo al elegir entre desarrollo nativo, web o híbrido. Esta decisión inicial es crucial, ya que sienta las bases para seleccionar posteriormente un lenguaje de programación que no solo cumpla con los requisitos funcionales del proyecto, sino que también optimice los atributos de calidad deseados. En un mercado saturado de opciones, la elección de un lenguaje de programación puede ser abrumadora, especialmente para quienes están a punto de embarcarse en nuevos proyectos o incluso para aquellos que buscan adentrarse en el mundo de la programación. Afortunadamente, es posible clasificar los lenguajes de programación según sus puntos fuertes; mientras algunos se destacan en el desarrollo de videojuegos, otros, como los que nos ocupan, están especialmente diseñados para el desarrollo de aplicaciones móviles híbridas.

A pesar de reducir la elección en tecnologías específicas para desarrollo móvil híbrido, la abundancia de opciones sigue presentando un desafío considerable. En respuesta a esto, desarrolladores de todo el mundo han comenzado a realizar evaluaciones comparativas utilizando diversas métricas, tales como velocidad, usabilidad, mantenimiento y capacidad de respuesta. Los resultados de estas comparativas se analizan para identificar ventajas y limitaciones específicas, lo que culmina en la publicación de informes de referencia o benchmarks que buscan orientar a la comunidad de desarrolladores.

Sin embargo, a pesar de la creciente demanda en el sector de aplicaciones móviles híbridas, aún se observa una escasez de referencias y estudios comparativos profundos que evalúen los lenguajes disponibles en el mercado actual. Esta carencia de información constituye una barrera significativa para la toma de decisiones informadas en un campo tan dinámico y en constante evolución como lo es el desarrollo de aplicaciones móviles.

**Solución**

Ante el desafío de seleccionar eficazmente entre los diversos frameworks disponibles para el desarrollo de aplicaciones móviles híbridas, esta tesis se propone desarrollar un Benchmark confiable y exhaustivo. Este Benchmark comparará tres de los frameworks híbridos más populares en la actualidad: Kotlin Multiplatform, React Native y Flutter. La comparación se realizará desde múltiples perspectivas, utilizando métricas diseñadas específicamente para evaluar los lenguajes de programación subyacentes de cada framework:

* Kotlin: a través de Kotlin Multiplatform.
* JavaScript: utilizando React Native.
* Dart: con Flutter.

Además de las métricas de lenguaje, se establecerán criterios adicionales para comparar directamente los frameworks en sí, así como las aplicaciones finales desarrolladas con cada uno. Estos criterios incluirán la evaluación de la funcionalidad, la eficiencia y la escalabilidad de los frameworks, junto con la experiencia del desarrollador y la calidad de la interfaz de las aplicaciones resultantes. Se analizará también la facilidad de integración con otros sistemas y servicios, además, se considerarán aspectos como la robustez del soporte comunitario y la disponibilidad de recursos de aprendizaje y de documentación, que pueden influir significativamente en la velocidad de desarrollo y en la resolución de problemas. Esta evaluación busca ofrecer una visión completa que no solo mida el rendimiento técnico, sino también la practicidad y eficiencia de los frameworks en escenarios de uso real.

**Objetivos**

* Implementar métricas de calidad básicas y generales que apliquen a todos los frameworks.
* Desarrollar métricas de calidad específicamente adaptadas al entorno del desarrollo móvil.
* Realizar una comparación objetiva de los lenguajes mediante el desarrollo de un proyecto común, permitiendo así evaluar su funcionalidad en condiciones uniformes.
* Facilitar un entorno donde otros desarrolladores puedan replicar o ampliar las comparativas realizadas, promoviendo una colaboración continua y el enriquecimiento del estudio.
* Conducir pruebas controladas para asegurar la precisión y relevancia de los resultados obtenidos.
* Ejecutar pruebas en ambientes de producción reales para validar la aplicabilidad de los frameworks en escenarios operativos auténticos.

**Metodología**

Como se mencionó anteriormente, el objetivo es evaluar la eficiencia tanto del framework como del lenguaje de programación base. Para ello, dividiremos el benchmark en dos partes esenciales. La primera parte se enfocará exclusivamente en los lenguajes de programación. Se llevarán a cabo pruebas de velocidad, monitoreando el tiempo que cada lenguaje requiere para resolver un problema matemático complejo y computacionalmente exigente. Posteriormente, se registrarán y analizarán estos tiempos para determinar qué lenguajes exhiben un rendimiento superior.

En la segunda parte, el análisis se centrará completamente en los frameworks. Esta evaluación abarcará una gama más amplia de variables, incluyendo el tiempo necesario para crear y compilar un nuevo proyecto, así como la calidad y estructura de la documentación y los proyectos internos de los frameworks. También se examinará la integración con servicios web externos, la implementación y gestión de sensores, la precisión en la exhibición de colores, el impacto en el rendimiento de la batería de las aplicaciones y, finalmente, la capacidad de cada framework para desarrollar una aplicación híbrida que sea visualmente atractiva y que incorpore todos los elementos mencionados anteriormente. Este enfoque integral pretende proporcionar una evaluación detallada que refleje la utilidad y la eficacia de los frameworks en condiciones prácticas y operativas reales.

**Primera Etapa**

Antes de empezar con las pruebas es necesario hacer una breve explicación acerca de como funcionan los 3 lenguajes de programación y cual es su uso principal.

Kotlin:

Kotlin es un lenguaje de programación de código abierto de tipo estático dirigido a JVM, Android, JavaScript, Wasm y Native. Está desarrollado por JetBrains. El proyecto comenzó en 2010 y fue de código abierto desde el principio. El primer lanzamiento oficial 1.0 fue en febrero de 2016. (JETBRAINS, 2024)

JavaScript:

JavaScript es un lenguaje de programación que los desarrolladores utilizan para hacer páginas web interactivas. Desde actualizar fuentes de redes sociales a mostrar animaciones y mapas interactivos, las funciones de JavaScript pueden mejorar la experiencia del usuario de un sitio web. Como lenguaje de scripting del lado del servidor, se trata de una de las principales tecnologías de la World Wide Web. (Amazon Web Services, 2023)

Dart:

Dart es un lenguaje optimizado para el cliente para desarrollar aplicaciones rápidas en cualquier plataforma. Su objetivo es ofrecer el lenguaje de programación más productivo para el desarrollo multiplataforma, junto con una plataforma de ejecución flexible para marcos de aplicaciones. (Dart, 2024)

Para realizar una comparación con respecto al rendimiento de los 3 lenguajes de programación se decidió optar por una métrica bastante usada en el mundo de la programación, esta se basa en medir el tiempo de ejecución de un programa, normalmente

un programa computacionalmente exigente para obtener tiempos de medición significativos.

Con esto en mente, definimos los siguientes 3 problemas que involucran cálculos matemáticos extensos:

El Conjunto de Mandelbrot:

La ecuación de Mandelbrot describe un conjunto de números complejos que generan un patrón fractal, conocido como el conjunto de Mandelbrot. Este patrón es una imagen muy famosa que parece un objeto con formas repetitivas y detalles infinitos. La idea básica es que tomas un número complejo 𝑐c y lo usas en una fórmula iterativa: 𝑧𝑛+1=𝑧𝑛2+𝑐zn+1​=zn2​+c. (Coordinador Comunicaciones UniSabaneta, 2024)

En nuestro programa, inicialmente establecemos dos números que representan las dimensiones (altura y anchura) de nuestro plano cartesiano, correspondientes a los ejes Y y X, respectivamente. A continuación, comenzamos a iterar sobre cada punto de este plano. Al llegar a un punto, fijamos un límite máximo de iteraciones y procedemos a determinar si este punto, y los subsiguientes generados por las iteraciones, pertenecen al conjunto de Mandelbrot. Este proceso se repite hasta alcanzar el número máximo de iteraciones establecido. Luego, avanzamos al siguiente punto inicial y repetimos el procedimiento.

Como se puede ver, este algoritmo propone varias iteraciones y cálculos, presentando una complejidad de: . Los 3 lenguajes comparten el mismo algoritmo por ende el tiempo de ejecución recae únicamente en el desempeño del lenguaje.

Criba de Eratóstenes: Este algoritmo se emplea para identificar los números primos dentro de un intervalo específico, X. Comienza con una iteración a partir del número 2 y continúa hasta que cada número ha sido evaluado. La metodología consiste en crear una lista o tabla booleana inicialmente establecida en valores verdaderos. Durante la iteración, utilizando una variable 𝑖, se marcan como falsos todos los múltiplos de 𝑖 que sean menores que X. De esta manera, se van excluyendo progresivamente los números compuestos del conjunto de posibles primos. La eficiencia del algoritmo, cuando se implementa en los tres lenguajes de programación, presenta la siguiente complejidad: .

Secuencia Fibonacci:

Es una secuencia infinita de números naturales cuyos dos primeros términos son 1 y 1 y tal que, cualquier otro término se obtiene sumando los dos inmediatamente anteriores. Por tanto, se cumple la relación: (Universidad de Almería, s.f.)

El algoritmo que hemos implementado está su forma tradicional y recursiva para encontrar el trigésimo quinto número primo no incluye optimizaciones avanzadas, como las que proporcionaría la programación dinámica. Este enfoque básico sigue un método directo sin almacenar los resultados de cálculos previos, lo cual podría reducir significativamente el número de operaciones necesarias al reutilizar estos resultados. Debido a la ausencia de tales mejoras, el algoritmo se enfrenta a una escalada exponencial en el número de operaciones requeridas a medida que aumenta el valor de entrada, lo que se refleja en su complejidad computacional de: . Decidimos ir por la implementación más básica ya que proporcionará tiempo de ejecución mayores, lo cual puede ser beneficioso para la comparación.

Resultados:

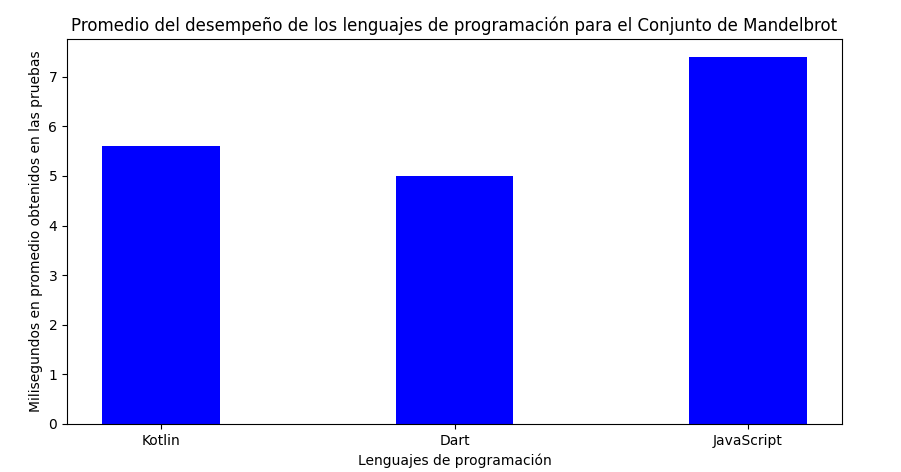
Teniendo en mente que la idea es que cualquier usuario pueda ejecutar estas pruebas en sus propios dispositivos y entornos locales, todos los algoritmos descritos anteriormente, para cada uno de los 3 lenguajes, fueron convertidos en imágenes Docker y posteriormente subidos a Docker Hub, de esta forma, cualquier usuario que cuente con Docker instalado en su respectivo dispositivo puede correr las pruebas de ejecución.

|  |  |  |  |
| --- | --- | --- | --- |
|  | El Conjunto de Mandelbrot | Criba de Eratóstenes | Secuencia Fibonacci |
| Kotlin | pelucapreb/tesis2024-dk:k.c.1 | pelucapreb/tesis2024-dk:k.c.2 | pelucapreb/tesis2024-dk:k.c.3 |
| JavaScript | pelucapreb/tesis2024-dk:d.c.1 | pelucapreb/tesis2024-dk:d.c.2 | pelucapreb/tesis2024-dk:d.c.3 |
| Dart | pelucapreb/tesis2024-dk:js.c.1 | pelucapreb/tesis2024-dk:js.c.2 | pelucapreb/tesis2024-dk:js.c.3 |

Para poder ejecutar las pruebas desde una terminal, es necesario poner el comando “Docker run” y añadir la imagen que se quiere ejecutar, en la tabla se pueden ver los nombres de las imágenes de cada lenguaje y de cada problema a resolver:

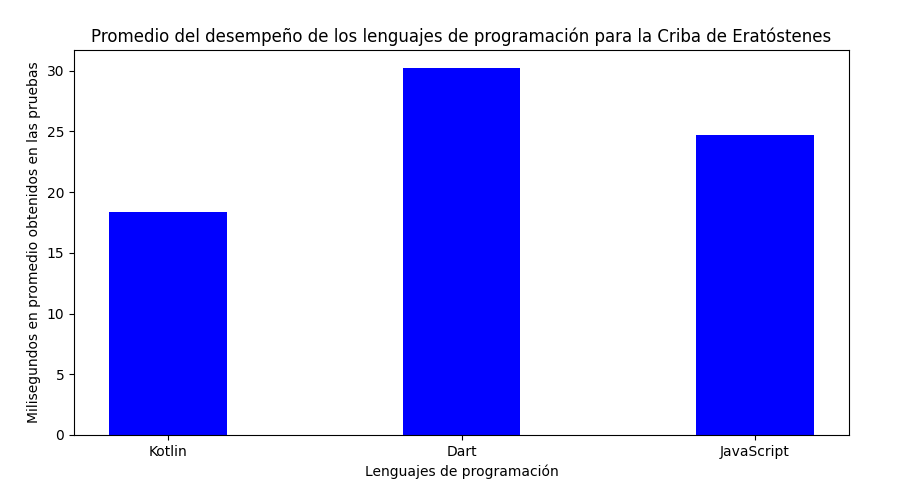
**Tabla #1 nombre imágenes Docker**

Posteriormente, implementamos tres scripts, uno para cada problema, escritos en Python. Estos scripts se encargan de ejecutar cada imagen cinco veces y calcular un promedio de los tiempos de ejecución para cada lenguaje de programación. Adicionalmente, cada vez que se ejecuta el script de Python, el orden de los lenguajes se selecciona de manera aleatoria. Esto se hace para mitigar cualquier latencia que pueda surgir del uso continuo de recursos computacionales, considerando que el rendimiento del último programa ejecutado podría verse ligeramente afectado. Finalmente, usando una librería llamada Matplotlib, se realiza una gráfica comparativa del promedio de los tiempos de ejecución de los 3 lenguajes.

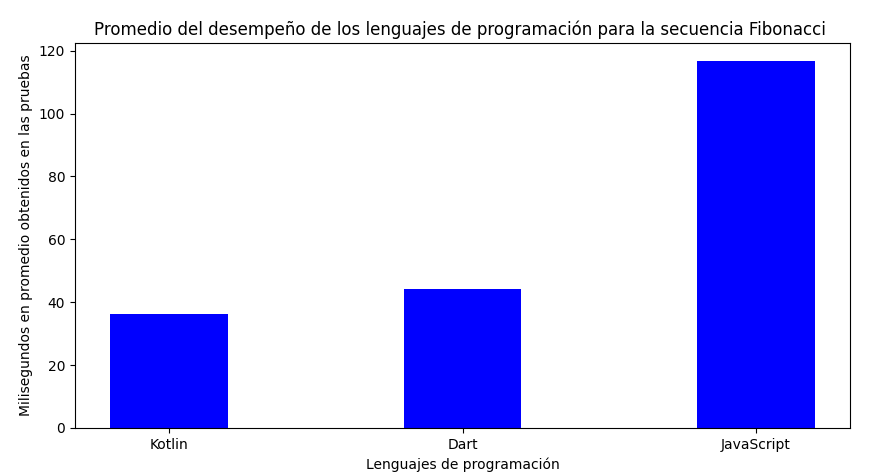
El Conjunto de Mandelbrot:

**Imagen #1 gráfica comparativa tiempos de ejecución conjunto de Mandelbrot**

Criba de Eratóstenes:



**Imagen #2 gráfica comparativa tiempos de ejecución de la criba de Eratóstenes.**

Secuencia Fibonacci:

**Imagen #3 gráfica comparativa tiempos de ejecución de la secuencia Fibonacci**

Como se puede ver en las gráficas de los resultados, Kotlin es el lenguaje que mayor velocidad tiene en la ejecución de los programas, seguido de Dart y, finalmente, de JavaScript. Esta jerarquía en la velocidad de ejecución puede atribuirse a varios factores propios de cada lenguaje. Kotlin, diseñado para ser tanto conciso como eficiente, aprovecha la Java Virtual Machine (JVM) para optimizar la ejecución, lo cual resulta en tiempos de procesamiento más rápidos y un mejor rendimiento en comparación con otros lenguajes que no utilizan compilación a bytecode. Por su parte, Dart, aunque inicialmente interpreta el código en tiempo de desarrollo, compila este a código nativo cuando se despliega la aplicación, lo que le permite superar en rendimiento a lenguajes puramente interpretados como JavaScript. JavaScript, siendo un lenguaje de scripting interpretado y con un enfoque en flexibilidad más que en velocidad, tiende a ser más lento en comparación con Kotlin y Dart, especialmente en tareas computacionalmente intensivas. Estas diferencias resaltan la importancia de elegir el lenguaje adecuado basado en las necesidades específicas de rendimiento y eficiencia para cada proyecto de software.