



TITULO DEL PROYECTO

Análisis Comparativo del Rendimiento y Facilidad de Aprendizaje de los
Frameworks de Flutter y React Native para el Desarrollo de Aplicaciones Móviles
en Laptops Master

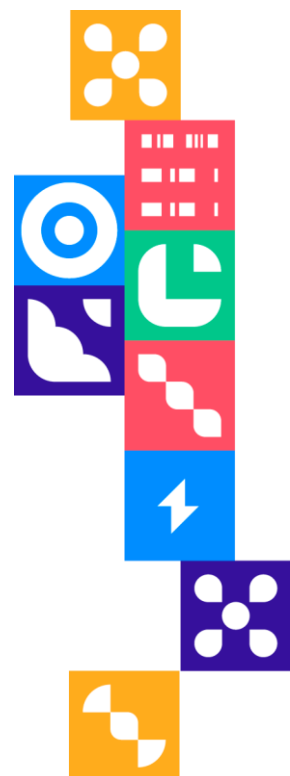
AUTOR

Hugo Armando Montaña Cuevas

No. Control: 200110351

CARRERA

Ingeniera en Sistemas Computacionales



AGRADECIMIENTOS

Expreso mi más profundo agradecimiento al profesor Miguel Ángel Gallardo Lemus, mi asesor, por su constante apoyo, guía y paciencia durante la realización de esta investigación. Su experiencia y consejos han sido esenciales para el desarrollo de este trabajo.

De igual forma, quiero agradecer a todos los profesores del Tecnológico Superior de Jalisco de la carrera de Ingeniería en Sistemas Computacionales, quienes han contribuido significativamente a mi formación con sus conocimientos y experiencias, marcando un impacto positivo en mi aprendizaje y crecimiento personal.

A mis padres, quienes con su amor, apoyo incondicional y palabras de aliento han sido mi pilar durante todo este proceso. Gracias por siempre creer en mí y por brindarme las herramientas necesarias para alcanzar mis metas.

A mis abuelos, por ser una fuente de inspiración y fortaleza. Sus valores, enseñanzas y cariño han dejado una huella imborrable en mi vida, y este logro también es un reflejo de su influencia en mi camino.

Gracias a todos aquellos que, de una u otra forma, han contribuido a que este proyecto sea una realidad.

RESUMEN

Este trabajo presenta un análisis comparativo entre los frameworks Flutter y React Native, enfocado en dos dimensiones principales: rendimiento y facilidad de aprendizaje. La investigación se basa en el desarrollo de aplicaciones móviles funcionales desarrolladas en Flutter y React Native, evaluadas bajo métricas como velocidad de ejecución, consumo de recursos (CPU y RAM), y la experiencia del desarrollo en ambos frameworks.

Los resultados se obtuvieron mediante herramientas como Android Profiler, Xcode Instruments y Flutter DevTools. Se observaron diferencias significativas en el rendimiento de las aplicaciones y en la curva de aprendizaje de los frameworks, lo que permitió identificar fortalezas y debilidades específicas de cada tecnología. El estudio concluye con recomendaciones basadas en datos objetivos, orientadas a optimizar la selección del framework según las necesidades de Laptops Master.

INDICE

AGRADECIMIENTOS.....	2
RESUMEN.....	3
INDICE	4
INDICE DE IMÁGENES	9
INDICE DE TABLAS	13
INTRODUCCION.....	14
PLANTEAMIENTO DEL PROBLEMA.....	15
PROBLEMA PRIORIZADO.....	16
CONSTRUCTO DE ALINEACION COHERENCIA	17
Pregunta de Investigación.....	17
Hipótesis	17
Objetivo General.....	17
Objetivos Específicos.....	17
JUSTIFICACION.....	19
CRITERIOS DE ÉXITO	21
ALCANCE Y DELIMITACIONES DEL PROYECTO	24
Alcance del proyecto.....	24
Delimitaciones del proyecto.....	24
MATRIZ DE TECNICAS E INSTRUMENTOS	25
Matriz de Congruencia.....	25
Matriz de Instrumentos	27
MARCO TEÓRICO.....	28
Dispositivo móvil	28
Teléfonos Móviles.....	28
Sistema Operativo (SO)	28
Android	29
El núcleo Linux.....	30
Runtime de Android.....	30
Librerías Nativas.....	31
Entorno de aplicación	32
Aplicaciones	32

IOS.....	32
Aplicación Móvil	34
Tipos De Aplicaciones	35
Arquitectura de Software.....	36
Tipos de arquitecturas de software.....	37
MVC (Modelo-Vista-Controlador).....	37
MVP (Modelo-Vista-Presentador)	38
MVVM (Model-View-ViewModel)	39
Arquitectura Limpia (Clean Architecture)	40
Arquitectura Cliente-Servidor.....	42
Arquitectura REST.....	43
Paradigmas y Lenguajes de Programación	44
Tipos de Paradigmas de Programación.....	45
Paradigma Imperativo.....	45
Paradigma Orientado a Objetos.....	45
Paradigma Funcional	46
Paradigma Declarativo	46
Paradigma Lógico.....	47
Paradigma Multiparadigma	48
Paradigmas de Lenguajes de Programación de este Proyecto	49
Dart	49
JavaScript	49
Frameworks.....	49
Framework React Native	49
Modelo de Hilo	50
Arquitectura antigua	50
Arquitectura Nueva.....	51
JavaScript	52
Virtual DOM	53
Framework Flutter	54
Dart	55
Widgets	56
APIS.....	57

Rendimiento	58
Android Profiler	59
Xcode Instruments.....	59
Flutter DevTools.....	59
Facilidad de Aprendizaje	60
METODOLOGÍA	61
Ciclo de Vida de Desarrollo de Software para la Aplicación de Flutter.....	61
Análisis	62
Requisitos funcionales.....	62
Requisitos no funcionales	62
Requisitos Técnicos	62
Diseño	64
Diseño de interfaz de usuario UI.....	64
Diagrama de componentes.....	66
Diagrama de Objetos	70
Implementación.....	78
Configuración del entorno	78
Módulos.....	84
Módulo de Consumo de API de TheMovieDB	84
Módulo de Favoritos con ISAR.....	95
Pruebas	103
Velocidad de Ejecución	103
Conclusión	104
Pruebas de Consumo de CPU y RAM	106
Android	106
Análisis del Reporte de Uso de CPU.....	106
Análisis del Reporte de Uso de Memoria	108
Conclusión	110
iOS	111
Análisis del Reporte de Uso del CPU.....	111
Análisis del Reporte de Uso de Memoria	112
Conclusiones.....	113
Ciclo de Vida de Desarrollo de Software para la Aplicación de React Native	114

Análisis	115
Requisitos funcionales	115
Requisitos no funcionales	115
Requisitos Técnicos	115
Diseño	117
Diseño de interfaz de usuario UI	117
Diagrama de componentes	118
Diagrama de Objetos	121
Implementación.....	125
Configuración del entorno de desarrollo	125
Módulos.....	130
Módulo de Consumo de API de TheMovieDB	130
Módulo de Favoritos con AsyncStorage	137
Pruebas	143
Velocidad de Ejecución	143
Conclusión	144
Pruebas de consumo de CPU y RAM	145
Android	145
Análisis de Reporte de Uso de CPU	145
Análisis de Reporte de Uso de Memoria.	147
Conclusiones.....	149
iOS	149
Análisis de Reporte de Uso de CPU	150
Análisis del Reporte de Uso de RAM	151
Conclusiones.....	153
RESULTADOS.....	154
Análisis Comparativo de los Frameworks de Flutter y React Native.....	154
Velocidad de Ejecución: Flutter vs React Native.....	154
Conclusión	156
Consumo de CPU y RAM: Flutter vs React Native	159
Consumo de CPU.....	159
Consumo de Memoria.....	160
Conclusión	161

Facilidad de Aprendizaje de los Frameworks de Flutter y React Native	163
Facilidad de aprendizaje de Flutter.....	163
Facilidad de aprendizaje de React Native	164
Conclusión	165
Conclusión General.....	167
Referencias	169

INDICE DE IMÁGENES

Imagen 1: Arquitectura de Android [6].....	30
Imagen 2: Comparativa de arquitecturas de Dalvik y ART	31
Imagen 3: Arquitectura en Capas de iOS [7]	33
Imagen 4: Estructura del patrón MVC [14]	38
Imagen 5: Componentes MVP [14]	39
Imagen 6: Componentes MVVM [14]	40
Imagen 7: Capas de la Arquitectura Limpia [15].....	41
Imagen 8: Capas de la Arquitectura Limpia [15].....	42
Imagen 9: Esquema Básico de Comunicación REST entre Cliente y Servidor [19]	44
Imagen 10: Flujo de Compilación en React Native [30].....	50
Imagen 11: Arquitectura Antigua de Ejecución en React Native [31]	51
Imagen 12: Arquitectura Moderna de React Native con JSI [31].....	52
Imagen 13: Ciclo de Reconciliación del Virtual DOM en React [33]	54
Imagen 14: Arquitectura de Flutter [34]	55
Imagen 15: Diseño de la Aplicación Flutter	64
Imagen 16: Diagrama de Componentes de la Arquitectura del Proyecto en Flutter	66
Imagen 17: Estructura Interna de las Capas de la Arquitectura del proyecto en Flutter	67
Imagen 18: Diagrama de Objetos de Domain	70
Imagen 19: Diagrama de Objetos Infraestructure.....	71
Imagen 20: Diagrama de Objetos de Presentation.....	72
Imagen 21: Diagrama de Objetos General	73
Imagen 22: Imagen visual de la estructura del proyecto domain.....	76
Imagen 23: Imagen visual de la estructura del proyecto infrastructure	76
Imagen 24: Imagen visual de la estructura del proyecto presentation.....	77
Imagen 25: Salida de Flutter Doctor para Diagnóstico del Entorno de Desarrollo.	80
Imagen 26: Resultado de Flutter Doctor: Entorno Configurado Correctamente	81
Imagen 27: Comando para crear un proyecto en Flutter	82

Imagen 28: Formulario de Registro de Usuario en TMDB.....	83
Imagen 29: Visualización de Clave de API en TMDB.....	83
Imagen 30: Implementación de un MovieDatasource para consumir TheMovieDB API con Dio en Flutter	85
Imagen 31: Métodos de MoviesDbDatasource para Obtener Películas Populares y en Cartelera.....	87
Imagen 32: Mapper para transformar objetos de TheMovieDB a entidades de dominio en Flutter.....	89
Imagen 33: Mapper para convertir un objeto Cast a una entidad Actor en Flutter	90
Imagen 34: Modelo MovieDbResponse para manejar datos de la API de TheMovieDB en Flutter	91
Imagen 35: Factory constructor para deserializar MovieDetails desde JSON en Flutter	93
Imagen 36: Método toJson para serializar un objeto MovieDetails a formato JSON en Flutter	94
Imagen 37: Datasource para gestionar la base de datos local con Isar en Flutter	96
Imagen 38: Método para verificar si una película es favorita en la base de datos Isar en Flutter	97
Imagen 39: Implementación del repositorio de almacenamiento local en Flutter ..	99
Imagen 40: Definición del proveedor para el repositorio de almacenamiento local con Riverpod en Flutter	101
Imagen 41: Reporte de análisis de rendimiento de CPU con Flutter DevTools de la aplicación de Flutter en Android	106
Imagen 42: Gráfica de uso de memoria en tiempo real con Flutter DevTools de la aplicación de Flutter en Android	108
Imagen 43: Detalle de memoria en un evento de recolección de basura (GC) ...	108
Imagen 44: Reporte de uso de CPU en Xcode Instruments de la aplicación de Flutter en iOS	111
Imagen 45: Reporte de uso de memoria en Xcode Instruments de la aplicación de Flutter en iOS	112
Imagen 46: Diseño de la Aplicación en React Native	117

Imagen 47: Diagrama de Componentes de Aplicación Móvil en React Native....	118
Imagen 48	121
Imagen 49: Finalización de la instalación de Expo CLI con npm.....	126
Imagen 50: Creación de proyecto en React Native con Expo: punto 1 y 2	127
Imagen 51: Creación de proyecto en React Native con Expo: punto 3	128
Imagen 52: Creación de proyecto en React Native con Expo: punto 4	128
Imagen 53: Creación de proyecto en React Native con Expo: punto 5	129
Imagen 54: Función genérica para realizar peticiones GET con Axios en JavaScript.....	131
Imagen 55: Definición de endpoints de la API de TheMovieDB en JavaScript....	133
Imagen 56: Funciones para consumir los endpoints de la API de TheMovieDB utilizando una capa de abstracción en JavaScript.....	135
Imagen 57: Pantalla MovieScreen con gestión de favoritos y verificación de datos existentes	138
Imagen 58: Función checkExistingItem para verificar y gestionar favoritos en AsyncStorage	139
Imagen 59: Función addNewItem para agregar o eliminar elementos de la lista de favoritos en AsyncStorage	140
Imagen 60: Pantalla FavoriteScreen con gestión de datos de favoritos.....	141
Imagen 61: Función getData para recuperar y actualizar datos de favoritos desde AsyncStorage	141
Imagen 62: Hook useEffect para sincronizar datos de favoritos al enfocar la pantalla en React	142
Imagen 63: Gráfico de uso de CPU en Android Profiler para la aplicación Android	146
Imagen 64: Gráfico de Android Profiler mostrando el uso máximo del CPU y distribución de threads	146
Imagen 65: Gráfico de Android Profiler destacando el uso máximo otros procesos	147
Imagen 66: Gráfico de uso de memoria en Android Profiler mostrando un pico de 61.9 MB.....	147

Imagen 67: Gráfico de uso de memoria segmentada en Android Profiler	148
Imagen 68: Reporte de uso de CPU en Xcode Instruments para la aplicación de React Native en iOS.....	150
Imagen 69: Reporte de uso de memoria en Xcode Instruments para la aplicación de React Native en iOS	151

INDICE DE TABLAS

Tabla 1: Matriz de Congruencia	27
Tabla 2: Matriz de Instrumentos	27
Tabla 3: Tiempos de Velocidad de Ejecución de la App Flutter en Android e iOS	103
Tabla 4: Tiempos de Velocidad de Ejecución de la App en React Native en Android e iOS	143

INTRODUCCION

En la actualidad, el desarrollo de aplicaciones móviles se ha convertido en un elemento estratégico para empresas que buscan mantenerse competitivas en un mercado dinámico y en constante evolución. Entre las diversas herramientas disponibles, los frameworks multiplataforma como Flutter y React Native han ganado popularidad debido a su capacidad para optimizar recursos y acelerar los tiempos de desarrollo. Sin embargo, la elección del framework adecuado puede ser un desafío, especialmente cuando se busca maximizar el rendimiento y minimizar la curva de aprendizaje.

Este documento tiene como objetivo comparar de manera empírica y objetiva los frameworks Flutter y React Native, evaluando su rendimiento, consumo de recursos y facilidad de aprendizaje. A través del desarrollo de aplicaciones funcionales y pruebas controladas, se pretende ofrecer una guía clara para seleccionar el framework que mejor se adapte a las necesidades específicas de la empresa ficticia Laptops Master en Puerto Vallarta, México. Además, el análisis incluye la experiencia del desarrollador, proporcionando una visión integral que considera tanto los aspectos técnicos como humanos del desarrollo de software.

PLANTEAMIENTO DEL PROBLEMA

La elección del framework de desarrollo adecuado se ha convertido en un desafío estratégico para Laptops Master. La empresa se enfrenta al dilema de seleccionar entre dos de los principales frameworks del mercado: Flutter y React Native. Ambos ofrecen ventajas significativas en términos de rendimiento y facilidad de aprendizaje, pero la falta de información específica sobre cómo se traducen estas ventajas en el contexto particular de Laptops Master dificulta la toma de decisiones. El problema radica en la falta de evidencia empírica y comparativa sobre las diferencias en rendimiento y facilidad de desarrollo entre Flutter y React Native, específicamente adaptadas a las necesidades y requisitos de Laptops Master en Puerto Vallarta. Esta carencia de datos conduce a una situación de incertidumbre que obstaculiza la capacidad de la empresa para tomar decisiones informadas y estratégicas con respecto a la adopción de un framework de desarrollo de aplicaciones móviles.

Por lo tanto, surge la necesidad de llevar a cabo una investigación exhaustiva que permita evaluar y comparar de manera objetiva y cuantitativa el rendimiento y facilidad de aprendizaje de los frameworks de Flutter y React Native. Solo mediante la realización de experimentos controlados y la recopilación de datos empíricos será posible determinar con precisión cuál de estos frameworks ofrece el mejor equilibrio entre rendimiento y facilidad de aprendizaje para las necesidades de la empresa.

PROBLEMA PRIORIZADO

El problema prioritario consiste en determinar cuál de los frameworks de desarrollo de aplicaciones móviles, Flutter o React Native, ofrece un mejor equilibrio entre rendimiento y facilidad de aprendizaje para las necesidades específicas de Laptops Master en Puerto Vallarta. Esta decisión es crucial para optimizar recursos, mejorar la competitividad y ofrecer soluciones móviles de alta calidad a los clientes de la empresa.

CONSTRUCTO DE ALINEACION COHERENCIA

Pregunta de Investigación

¿Cuáles son las diferencias en el rendimiento y facilidad de aprendizaje entre los frameworks de Flutter y React Native en el desarrollo de aplicaciones móviles que se pretende construir en Laptops Master de Puerto Vallarta en el año 2024?

Hipótesis

H0. - Usando el framework de FLUTTER y con el lenguaje de DART se obtiene un mayor rendimiento en el desarrollo de aplicaciones móviles para ambas plataformas y tiene mayor facilidad de aprendizaje comparado con el framework de REACT NATIVE y con su lenguaje principal de JAVASCRIPT.

H1. - Usando el framework de REACT NATIVE y con el lenguaje de JAVASCRIPT se obtiene un mayor rendimiento en el desarrollo de aplicaciones móviles para ambas plataformas y tiene mayor facilidad de aprendizaje comparado con el framework de FLUTTER y con su lenguaje principal de DART.

Objetivo General

Generar un análisis comparativo del rendimiento y facilidad de aprendizaje de los frameworks de FLUTTER Y REACT NATIVE para el desarrollo de aplicaciones híbridas en LAPTOPS MASTER y determinar cuál es mejor para la empresa.

Objetivos Específicos

1. Desarrollar una App en el framework de Flutter con el lenguaje de Dart.
2. Evaluar el rendimiento de la aplicación desarrollada con el framework de Flutter en términos de velocidad de ejecución y consumo de recursos.
3. Desarrollar una App en el framework de React Native con el lenguaje de JavaScript.
4. Evaluar el rendimiento de la aplicación desarrollada con el framework de React Native en términos de velocidad de ejecución y consumo de recursos.
5. Analizar la facilidad de aprendizaje del framework de Flutter en base a la curva de aprendizaje, la documentación y soporte existente del framework.
6. Analizar la facilidad de aprendizaje del framework de React Native en base a la curva de aprendizaje, la documentación y soporte existente del framework.

7. Analizar la experiencia del usuario(desarrollador) basado en sus preferencias, satisfacción y problemas comunes con el framework de Flutter.
8. Analizar la experiencia del usuario(desarrollador) basado en sus preferencias, satisfacción y problemas comunes con el framework de React Native.

JUSTIFICACION

La investigación propuesta se centra en la comparación del rendimiento y la facilidad de aprendizaje entre los frameworks de FLUTTER Y REACT NATIVE para el desarrollo de aplicaciones móviles que se pretenden construir en Laptops Master en Puerto Vallarta. Esta comparación es fundamental por las siguientes razones:

- **Optimización de recursos:** La selección del framework de desarrollo adecuado puede influir significativamente en la optimización de recursos, como el tiempo y los esfuerzos dedicados al desarrollo de aplicaciones móviles. Al identificar cuál de los dos frameworks ofrece un mejor rendimiento y una mayor facilidad de aprendizaje, Laptops Master podrá asignar sus recursos de manera más eficiente y obtener resultados más efectivos.
- **Mejora de la competitividad:** En un mercado donde la velocidad y la calidad de desarrollo son críticas, la capacidad de ofrecer aplicaciones móviles con un rendimiento superior puede marcar la diferencia en la satisfacción del cliente y la competitividad empresarial. Al determinar cuál de los frameworks proporciona un mejor rendimiento, Laptops Master podrá ofrecer soluciones más robustas y atractivas para sus clientes, lo que contribuirá a fortalecer su posición en el mercado.
- **Reducción de costos y tiempos de desarrollo:** La facilidad de aprendizaje es un factor clave ya que puede reducir los costos y tiempos para comenzar a desarrollar aplicaciones móviles. Al identificar el framework que ofrezca una curva de aprendizaje más suave y permita a los trabajadores adaptarse rápidamente, Laptops Master podrá minimizar los costos asociados con la capacitación y el desarrollo, y acelerar la entrega de las aplicaciones al mercado. Esto brindará una ventaja competitiva significativa al optimizar tanto el tiempo como los recursos necesarios para el desarrollo.
- **Toma de decisiones informada:** La investigación proporcionará datos objetivos y comparativos sobre el rendimiento y la facilidad de aprendizaje de FLUTTER y REACT NATIVE, lo que permitirá a Laptops Master tomar decisiones informadas y basadas en evidencia sobre qué framework se

ajusta mejor a sus necesidades y objetivos específicos de desarrollo de aplicaciones móviles.

CRITERIOS DE ÉXITO

El éxito de este análisis comparativo se definirá en función de si los resultados obtenidos respaldan la hipótesis H0, que establece que Flutter, utilizando el lenguaje Dart, ofrece un mayor rendimiento y facilidad de aprendizaje en el desarrollo de aplicaciones móviles multiplataforma en comparación con React Native y su lenguaje principal, JavaScript.

1. Validación de la Hipótesis H0

- a. Los resultados de las pruebas deben demostrar que las aplicaciones desarrolladas con Flutter tienen:
 - i. **Mejor rendimiento:** Evaluado mediante métricas de velocidad de ejecución, consumo de recursos (CPU y RAM).
 - ii. **Mayor facilidad de aprendizaje:** Evaluada mediante la curva de aprendizaje, calidad de la documentación y soporte.

2. Indicadores de Rendimiento

a. Velocidad de ejecución:

- i. La aplicación desarrollada en Flutter debe presentar tiempos de respuesta más rápidos en operaciones clave como apertura de pantallas, carga de datos, y transiciones visuales, en comparación con React Native.

b. Consumo de recursos (CPU y RAM):

- i. Flutter debe mostrar un consumo más eficiente de recursos en dispositivos de prueba equivalentes.

3. Indicadores de Facilidad de Aprendizaje

a. Curva de aprendizaje:

- i. La facilidad de aprendizaje de Flutter será evaluada mediante la experiencia del desarrollador principal al implementar una aplicación funcional básica. Se considerará el tiempo requerido para comprender conceptos clave, implementar funcionalidades principales y superar obstáculos iniciales.

b. Documentación y soporte:

- i. La utilidad de la documentación oficial y los recursos de soporte de Flutter será analizada en función de su efectividad para resolver problemas, facilitar el avance en el desarrollo y cubrir aspectos esenciales como configuración inicial y manejo de errores.

c. Satisfacción del desarrollador:

- i. Se analizará el nivel de satisfacción del desarrollador con Flutter, considerando aspectos como facilidad de uso, claridad de la documentación, soporte de la comunidad y capacidad para resolver problemas comunes de manera autónoma.

4. Confirmación de la Hipótesis H0

- a. La hipótesis H0 se considerará confirmada si los resultados obtenidos muestran que Flutter supera a React Native en términos de rendimiento y facilidad de aprendizaje. Esto incluirá evidencia de que Flutter ofrece un mejor desempeño en la mayoría de los indicadores definidos, tales como velocidad de ejecución, consumo de recursos, y satisfacción del desarrollador durante el proceso de aprendizaje y desarrollo.

Para garantizar una comparación rigurosa entre los dos frameworks, todas las pruebas se llevarán a cabo en condiciones equitativas. Esto implica que se utilizarán dispositivos móviles físicos con características idénticas para probar cada aplicación desarrollada con ambos frameworks. El objetivo principal de estas pruebas es determinar cuál de los dos frameworks ofrece un mejor rendimiento y, en última instancia, cuál de los 2 frameworks tiene mayor facilidad de aprendizaje para comenzar a desarrollar aplicaciones móviles.

Es esencial que todas las pruebas se realicen en un entorno controlado y consistente. Por lo tanto, se emplearán dispositivos con las mismas especificaciones técnicas, incluyendo CPU, memoria RAM, almacenamiento y otros componentes relevantes. Además, se asegurará la uniformidad en el sistema operativo y su versión en todos los dispositivos utilizados para las pruebas.

Cada aplicación será sometida a una serie de mediciones cuantitativas, incluyendo el tiempo de inicio desde que se hace click en el ícono hasta que la aplicación está completamente cargada, la cantidad de código necesario para lograr funcionalidades similares en ambos frameworks

Al mantener un enfoque equitativo en todas las pruebas, se espera obtener resultados confiables y significativos que permitan una evaluación objetiva de los dos frameworks en términos de rendimiento y facilidad de aprendizaje.

ALCANCE Y DELIMITACIONES DEL PROYECTO

Alcance del proyecto

El presente proyecto tiene como objetivo llevar a cabo un análisis comparativo del rendimiento y la facilidad de desarrollo de los frameworks de Flutter y React Native para la creación de aplicaciones móviles destinadas a Laptops Master en Puerto Vallarta durante el año 2024.

Delimitaciones del proyecto

1. **Alcance temporal:** El Estudio se limita al año 2024, lo que significa que las conclusiones y recomendaciones se basaran en las condiciones y tecnologías disponibles hasta ese momento.
2. **Exclusión de otros frameworks:** El estudio se centra únicamente en comparar Flutter y React Native, lo que significa que otros frameworks alternativos para el desarrollo multiplataforma no se consideraran en este análisis comparativo.
3. **Versiones específicas:** El estudio tomara en cuenta las últimas versiones de cada framework como sus respectivos lenguajes en sus últimas actualizaciones en el año 2024.

MATRIZ DE TECNICAS E INSTRUMENTOS

Matriz de Congruencia

Pregunta de Investigación	Objetivo General	Objetivos Específicos	Hipótesis	Dimensiones
¿Cuáles son las diferencias en el rendimiento y facilidad de aprendizaje entre los frameworks de Flutter y React Native en el desarrollo de aplicaciones móviles que se pretende construir en Laptops Master de Puerto Vallarta en el año 2024?	Generar un análisis comparativo del rendimiento y facilidad de aprendizaje de los frameworks de FLUTTER Y REACT NATIVE para el desarrollo de aplicaciones híbridas en LAPTOPS MASTER y determinar cuál es mejor para la empresa.	<p>1-Desarrollar una App en Flutter con el lenguaje de Dart.</p> <p>2-Evaluar el rendimiento de la aplicación desarrollada con el framework de Flutter en términos de velocidad de ejecución y consumo de recursos.</p> <p>3-Desarrollar una App en React Native con el lenguaje de JavaScript.</p> <p>4-Evaluar el rendimiento de la aplicación desarrollada con el framework de React Native en términos de velocidad de ejecución y consumo de recursos.</p>	<p>H0. - Usando el framework de FLUTTER y con el lenguaje de DART se obtiene un mayor rendimiento en el desarrollo de aplicaciones móviles para ambas plataformas y tiene mayor facilidad de aprendizaje comparado con el framework de REACT NATIVE y con su lenguaje principal de JAVASCRIPT.</p> <p>H1. - Usando el framework de REACT NATIVE y con el lenguaje</p>	<p>1- Rendimiento de las apps en cada framework:</p> <ul style="list-style-type: none"> • Velocidad de ejecución. • Consumo de recursos (CPU y RAM) <p>2- Facilidad de aprendizaje:</p> <ul style="list-style-type: none"> • Curva de aprendizaje para desarrolladores. • Documentación y soporte. <p>3- Experiencia del usuario(desarrollador):</p> <ul style="list-style-type: none"> • Satisfacción con el framework. • Desafíos comunes enfrentados.

		<p>5-Analizar la facilidad de aprendizaje del framework de Flutter en base a la curva de aprendizaje, la documentación y soporte existente del framework.</p> <p>6-Analizar la facilidad de aprendizaje del framework de React Native en base a la curva de aprendizaje, la documentación y soporte existente del framework.</p> <p>7-Analizar la experiencia del usuario(desarrollador) basado en sus preferencias, satisfacción y problemas comunes con el framework de Flutter.</p> <p>8-Analizar la experiencia del usuario(desarrollador) basado en sus preferencias, satisfacción y</p>	<p>de JAVASCRIPT se obtiene un mayor rendimiento en el desarrollo de aplicaciones móviles para ambas plataformas y tiene mayor facilidad de aprendizaje comparado con el framework de FLUTTER y con su lenguaje principal de DART.</p>	
--	--	---	--	--

		problemas comunes con el framework de React Native.		
--	--	---	--	--

Tabla 1: Matriz de Congruencia

Matriz de Instrumentos

Dimensiones	Herramientas o Instrumentos
Rendimiento de las apps en cada framework.	<ul style="list-style-type: none"> • Android Profiler • Xcode Instruments • Flutter DevTools
Facilidad de aprendizaje y experiencia del usuario (desarrollador).	<ul style="list-style-type: none"> • Relato de experiencia personal en el desarrollo de las aplicaciones móviles

Tabla 2: Matriz de Instrumentos

MARCO TEÓRICO

Dispositivo móvil

Un dispositivo móvil se puede definir como un aparato tan pequeño con características tales como conexión a red, capacidad de procesamiento con cierta memoria limitada y el cual ha sido diseñado específicamente para una función pero que puede realizar otras funciones generales. [1]. Con dicha definición se da a entender que un dispositivo móvil es cualquier dispositivo que es fácil de transportar y que puede procesar información. Entonces aquellos que entran en esta categoría van desde reproductores de música, teléfonos móviles e incluso hasta computadoras portátiles.

Teléfonos Móviles

Los Smartphones son teléfonos inteligentes en donde su nivel de ubicuidad hace que cualquier usuario pueda estar conectado con otro individuo e incluso tener acceso a la red Internet desde cualquier lugar. [2]. En un principio fueron creados para realizar solamente llamadas sin necesidad de estar conectados a una terminal por medio de un cable, con el paso de tiempo se fueron añadiendo nuevas funcionalidades tales como la mensajería instantánea, juegos, cámaras, linternas entre otras más.

Una de las características que tiene los teléfonos móviles es permitir la instalación de programas o aplicaciones que realizan el procesamiento de información ya que poseen un sistema operativo que facilita la interacción con el dispositivo además de contar con bastantes sensores.

Por las facilidades que nos brindan estos dispositivos y el fácil acceso que se tiene ha provocado que los programadores se enfoquen en desarrollar aplicaciones para estos dispositivos, dando como resultado que estén disponibles 2.7 millones de aplicaciones en la Play Store y 1.8 millones de aplicaciones en la App Store. [3]

Sistema Operativo (SO)

El sistema operativo o por sus siglas SO es una capa compleja entre el hardware y el usuario el cual le facilita las herramientas e interfaces adecuadas para llevar a

cabo sus tareas abstrayéndole de procesos complejos para llevarlas a cabo. [1]. Entonces se entiende que el sistema operativo es aquel que interactúa directamente con el dispositivo con la finalidad de facilitar la interacción del usuario con el dispositivo.

Cada creador de dispositivos móviles decide el sistema operativo que instalara en su producto por este motivo las empresas a lo largo de estos años han desarrollado sus propios sistemas operativos de los cuales hoy en día los líderes de los sistemas operativos son Android e iOS, sin embargo existieron otros tipos de sistemas operativos tales como Windows Phone y otros no tan usados como BlackBerry OS Firefox OS entre otros más de los cuales ya han sido descontinuados por su falta de popularidad y no cumplir con las necesidades del usuario.

Android

Es un sistema operativo para teléfonos móviles basado en Linux el cual fue desarrollado por Google y presentado en 2007, además el sistema permite desarrollos de aplicaciones móviles con los lenguajes de Java y Kotlin. [4]

El sistema operativo de Android permite ejecutar aplicaciones desarrolladas en el lenguaje de java, aunque hoy en día se ha optado por comenzar a desarrollar aplicaciones con el lenguaje de Kotlin el cual fue creado por JetBrains, la empresa que está detrás de IntelliJ IDEO el cual es uno de los mejores IDE de desarrollo para java. Además, han desarrollado un módulo Kotlin el cual es Kotlin/Native capaz de compilar un único código base para aplicaciones tanto para iOS y Android [5]

La arquitectura de Android está conformada por el Núcleo de Linux, el Runtime de Android, las librerías nativas, el entorno de aplicación y las aplicaciones como se muestra en el siguiente gráfico.

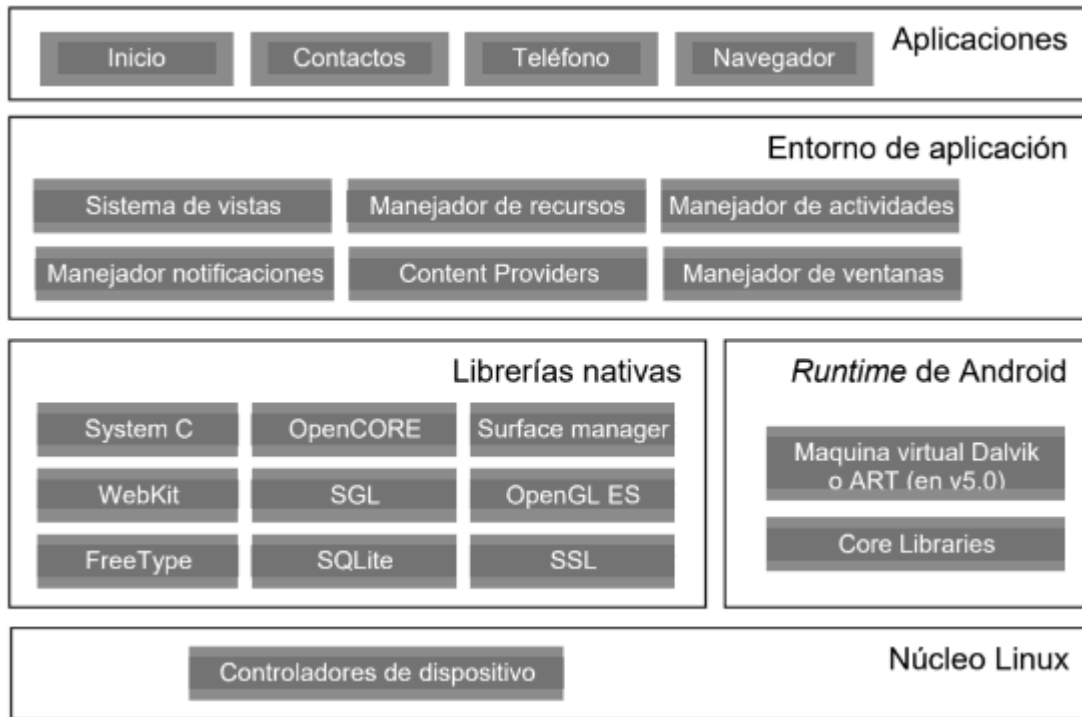


Imagen 1: Arquitectura de Android [6]

El núcleo Linux

El núcleo principal de Android está formado por el propio sistema operativo de Linux el cual proporciona una gran variedad de servicios como la seguridad, manejo de memoria, el multiproceso entre otras más. [6]

Runtime de Android

El Runtime de Android o mejor conocido como ART es una máquina virtual que remplazo la máquina virtual que utilizaba antes Android llamada Dalvik, el cual optimizaba los recursos de una mejor manera sin embargo el nuevo entorno de ejecución lograba reducir el tiempo de ejecución del código de java hasta un 33%. [6]

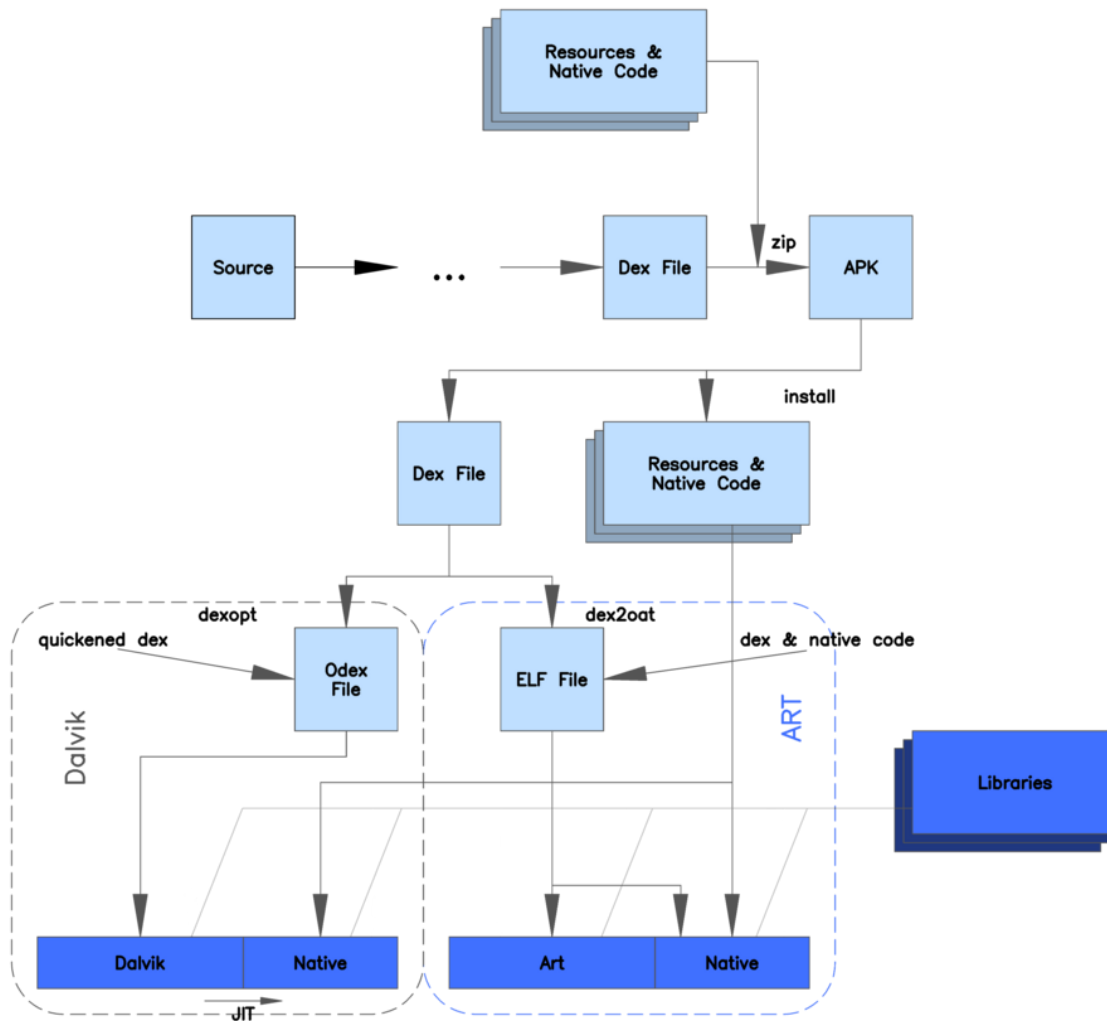


Imagen 2: Comparativa de arquitecturas de Dalvik y ART [7]

Librerías Nativas

Android cuenta con un conjunto de librerías en C/C++ que están compiladas en el propio código nativo del procesador para ser usadas por varios componentes de Android [6], algunas de estas son:

- System C library
- Media Framework
- Surface Manager
- WebKit/Chromium
- SGL
- Librerías 3D

- FreeType
- SQLite
- SSL

Entorno de aplicación

El entorno de aplicación de Android ofrece una plataforma libre de desarrollo para aplicaciones que usen una gran parte de sensores, servicios, localización, notificaciones etc. Además, fue diseñada para simplificar la reutilización de componentes [6]. Algunos servicios importantes son:

- **Views:** Es el conjunto visual de los componentes de la vista.
- **Resourcec Manager:** Proporciona acceso a recursos que no son código.
- **Activity Manager:** Maneja el ciclo de vida de las aplicaciones y un sistema de navegación entre estas.
- **Notification Manager:** Permite a las aplicaciones mostrar alertas en la barra de estado.
- **Content Providers:** Es un mecanismo que permite acceder a datos de otras aplicaciones del sistema.

Aplicaciones

Este es el último nivel de la arquitectura el cual tiene un conjunto de aplicaciones instaladas en Android en donde todas estas han de correr en la máquina virtual ART para garantizar la seguridad del propio sistema. [6]

IOS

Es un sistema operativo de Apple lanzado en 2007 siendo un sistema operativo de tipo Unix del cual es derivado de macOS que a su vez deriva de Darwin BSD y el cual se encuentra únicamente en los dispositivos creados por Apple tales como iPhone, iPod, iPad o Apple Watch [4]

La jerarquía de la plataforma iOS consta de 4 capas las cuales contienen una serie de componentes los cuales a través de estos completan las funciones correspondientes como se muestra en la Figura:

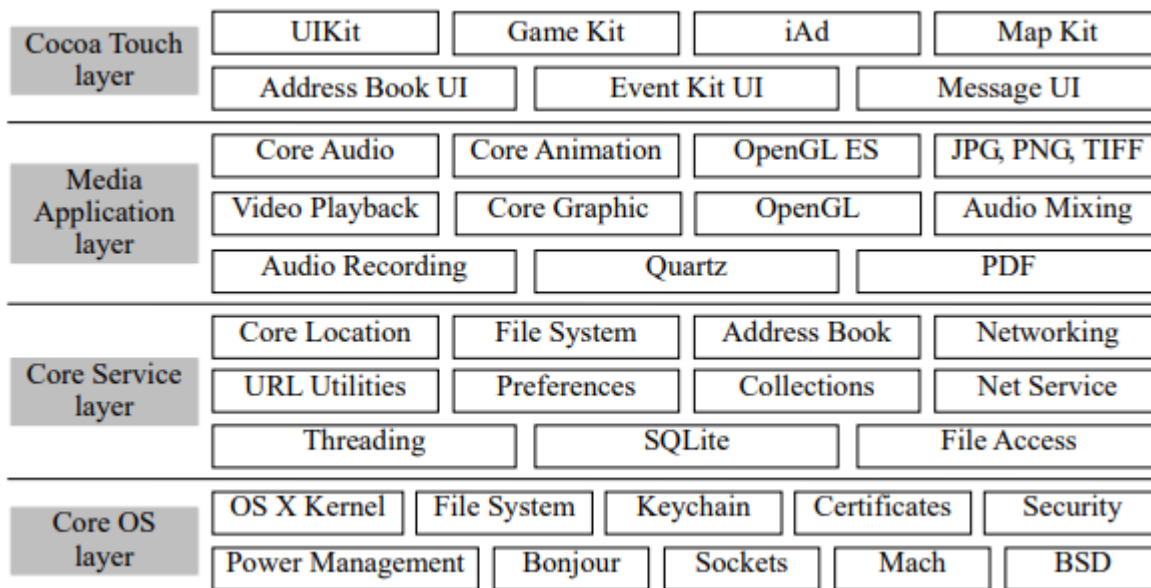


Imagen 3: Arquitectura en Capas de iOS [8]

- **Cocoa Touch:** El framework Cocoa Touch utiliza gran parte del modo maduro de Mac para centrarse mucho más en la interfaz touch y su optimización. Además, UIKit proporciona una gran variedad de herramientas básicas para iOS como gráficos, programas controlados por eventos, procesamiento de archivos, manipulación de redes y cadenas entre otras más, lo que proporciona que Cocoa Touch y la interfaz del usuario del iPhone sea compatibles con el diseño especial que UIKit puede proporcionar. [8]
- **Media Application:** Esta capa proporciona una aplicación completa de todo tipo de imágenes, audio y video, el marco de alto nivel del sistema operativo de iPhone se puede utilizar para poder crear gráficos y animaciones avanzados mucho más rapido y eficientes. Una parte importante de las aplicaciones de iPhone son las imágenes de alta calidad, incluida la reproducción de audio, su calidad de grabación además de admitir MediaPlayer un framework que permite la reproducción de pantalla completa. [8]
- **Core Services:** El Core Services se encuentra ubicada en la capa del sistema operativo básico superior el cual proporciona una base para todos

los servicios del sistema de aplicaciones. Algunos marcos que existen en Core Services son los siguientes:

- **AddressBook:** Este marco proporciona una interfaz de programación almacenada en la guía telefónica del dispositivo móvil.
- **CoreFoundation:** Este marco se basa en un conjunto de interfaces en el lenguaje de C, el cual nos da funciones básicas de administración y servicios de datos.
- **CFNetwork:** Este marco nos proporciona una abstracción orientada a objetos de protocolo de red.
- **CoreLocation:** Este marco nos permite obtener la latitud y longitud actual del dispositivo [8]
- **Core OS:** La capa del sistema operativo central contiene el entorno del propio núcleo del sistema operativo, la unidad y la interfaz básica. El núcleo se basa en el propio sistema operativo que tiene Mac, este se responsabiliza de todos los aspectos del propio sistema tales como la gestión de memoria virtual, hilos, sistemas de archivos, red y la comunicación interna. El controlador de la capa de sistema proporciona la interfaz entre el hardware y el marco del sistema. [8]

Aplicación Móvil

Una aplicación móvil es una aplicación informática el cual está diseñada para funcionar solo en teléfonos inteligentes, específicamente en el sistema operativo para el cual fue diseñado ya sea para el sistema operativo de Android o para iOS. Por lo general se encuentran disponibles a través de plataformas de distribución, operadas por las compañías propietarias de los sistemas operativos móviles como Android o iOS. [9]

Estos tipos de programas o aplicaciones son creados con diferentes lenguajes de programación y entornos de desarrollos diferentes, e incluso con las propias herramientas de cada sistema operativo en que él se desarrolla.

La tecnología de los dispositivos móviles ha revolucionado la comunicación interpersonal, por lo que las plataformas de comunicación digital (aplicaciones móviles) nos facilita la transferencia de datos entre individuos. [10]

Tipos De Aplicaciones

Existen varias formas de crear una aplicación móvil ya que hay más de una sola forma para desarrollarlas. La creciente demanda de software para dispositivos móviles ha generado nuevos desafíos para los desarrolladores ya que este tipo de aplicaciones tienen sus propias características, restricciones y necesidades únicas. [11]. Sin embargo, el usuario final no siempre nota estas diferencias, pero desde el punto de vista técnico cada una de las aplicaciones puede tener limitaciones al momento de realizar alguna función.

- **Aplicaciones Nativas:** Las aplicaciones nativas son solo aquellas que se pueden ejecutar en la plataforma o sistema operativo para el cual fueron creadas, es decir, se debe considerar el tipo de dispositivo su propio sistema y su versión. La principal ventaja de estas aplicaciones es la posibilidad de interactuar con todas las funcionalidades que brinda el dispositivo sin ninguna limitación o faltas de restricción, sin embargo, dado que solo está desarrollado para un solo sistema operativo y se desea abarcar otros sistemas es necesario crear una aplicación para cada sistema operativo. [11]
- **Aplicaciones Web:** Las aplicaciones web para dispositivos móviles son diseñadas solo para ser ejecutadas en navegadores de internet del propio dispositivo móvil, tales como Google, Safari Firefox entre otras más. Por lo general estas aplicaciones son desarrolladas con las mismas tecnologías que se usan para crear páginas web como HTML, CSS y JavaScript. Una de las ventajas de estas aplicaciones es que no necesitan ser instaladas en el propio dispositivo, además, no requieren aprobación del sistema u otros componentes para poder usarlas. La ventaja principal de estas aplicaciones es su independencia de la plataforma o sistema operativo ya que solo necesita un navegador, la contraparte de esto es que la velocidad de

ejecución es mucho menor que una aplicación nativa instalada en el propio dispositivo. [11]

- **Aplicaciones Híbridas:** Las aplicaciones Híbridas combinan lo mejor de los 2 tipos de aplicaciones anteriores utilizando las tecnologías multiplataforma como HTML, CSS y JavaScript, pero con acceso a una gran parte de las capacidades específicas del dispositivo, es decir, son desarrolladas utilizando tecnologías web, pero son ejecutadas en un contenedor web sobre el dispositivo móvil. Una de las principales ventajas es la distribución de la aplicación ya que al ser híbridas pueden abarcar varios sistemas operativos con un mismo código fuente y la posibilidad de utilizar las características del hardware del propio dispositivo. Una de sus desventajas es que, al utilizar la misma interfaz gráfica, su apariencia no será similar a una aplicación nativa, además de que su ejecución será un poco más lenta que la de una aplicación nativa. [11]

Arquitectura de Software

La Arquitectura de Software (AS) es un concepto fundamental en el desarrollo de sistemas complejos, pero carece de una definición unánime debido a la diversidad de perspectivas que existen sobre su naturaleza. Las definiciones pueden superar las centenas, reflejando una falta de consenso entre los profesionales. En general, estas definiciones tienden a entremezclar distintos aspectos, como el proceso dinámico de definir la arquitectura dentro del ciclo de vida del software, la configuración estática de los componentes del sistema desde un nivel de abstracción elevado, y la disciplina que estudia estas dos áreas. “Una definición reconocida es la de Clements, quien considera la AS como una vista del sistema que incluye sus componentes principales, su conducta e interacción para alcanzar los objetivos del sistema”. [12] Pero también la arquitectura de software se puede definir como una colección de patrones y abstracciones que permiten resolver problemas de manera eficiente. Además, una AS tiene como propósito cumplir con ciertos atributos de calidad, tales como la mantenibilidad, la extensibilidad, la flexibilidad y la capacidad de interactuar con otros sistemas de información. [13]

Tipos de arquitecturas de software

En esencia, la Arquitectura de Software proporciona una comprensión abstracta del sistema, permitiendo su análisis y diseño sin entrar en los detalles minuciosos que se tratarán en etapas posteriores. A continuación, se describen algunas de las arquitecturas más relevantes:

MVC (Modelo-Vista-Controlador)

El patrón de arquitectura Modelo/Vista/Controlador (MVC) fue descrito por primera vez en 1979 por Trygve Reenskaug e introducido en el lenguaje Smalltalk-80. MVC está diseñado para reducir el esfuerzo de programación en sistemas con múltiples vistas sincronizadas de los mismos datos. En este patrón, el Modelo, las Vistas y los Controladores se tratan como entidades separadas, lo cual permite que cualquier cambio en el Modelo se refleje automáticamente en las Vistas. Las principales ventajas del patrón MVC incluyen la separación clara de los componentes del programa, una interfaz API bien definida que permite reemplazar sus partes sin problemas, y la conexión dinámica entre el Modelo y las Vistas en tiempo de ejecución. Esto facilita la construcción modular del programa y hace que los componentes puedan ser reemplazados sin afectar el resto del sistema, lo cual es una mejora respecto a las aproximaciones monolíticas. [14]

El patrón MVC (Modelo-Vista-Controlador) divide la aplicación en tres componentes esenciales.

- El Modelo gestiona los datos y sus transformaciones, sin conocimiento específico de las Vistas o Controladores.
- La Vista es responsable de mostrar los datos del Modelo al usuario.
- Y el Controlador interpreta las acciones del usuario y coordina la comunicación entre la Vista y el Modelo, asegurando una separación clara que facilita el mantenimiento y escalabilidad del sistema.

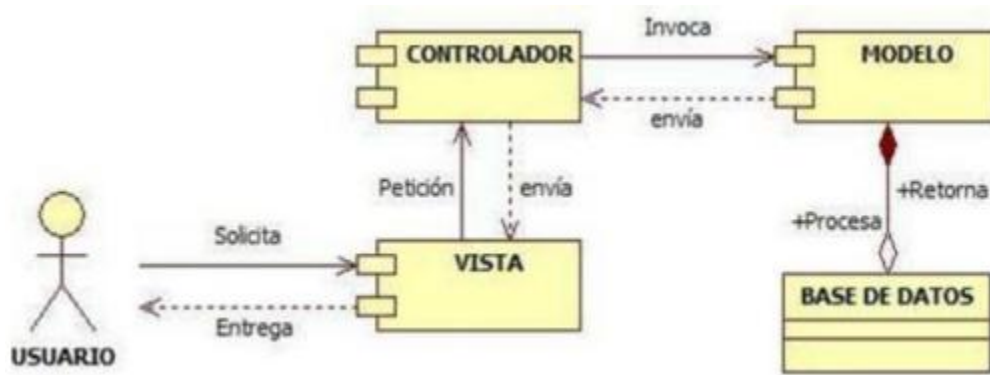


Imagen 4: Estructura del patrón MVC [15]

MVP (Modelo-Vista-Presentador)

El MVP es un patrón arquitectónico utilizado en la capa de presentación de aplicaciones de software. Fue desarrollado por Taligent en los años 90 y se implementó inicialmente en C++ y Java. Este patrón se basa en los principios de MVC (Modelo-Vista-Controlador). [15]

Este modelo tiene mucha más flexibilidad al implementarse una función en la clase especificada, además de contar con la facilidad de realizar varios escenarios de pruebas mejorando la capacidad de mantenimiento y permitiendo crear y ejecutar con facilidad los escenarios de prueba.

El patrón MVP se compone de 3 partes esenciales tales como:

- El Modelo almacena los datos y la lógica de negocio, exponiendo solo interfaces para conectarse con el presentador, ocultando detalles internos.
- La Vista es la interfaz de usuario, recibe las acciones del usuario y las envía al presentador, mostrando luego el resultado.
- Y el Presentador actúa como intermediario entre la vista y el modelo, procesando las peticiones del usuario, obteniendo información del modelo y actualizando la vista.

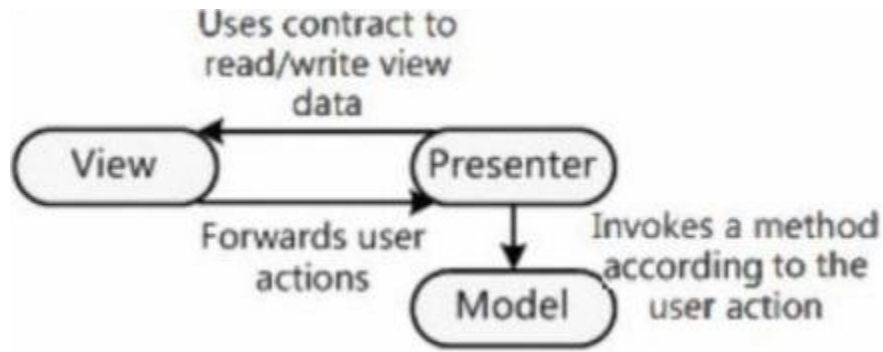


Imagen 5: Componentes MVP [15]

MVVM (Model-View-ViewModel)

El patrón Modelo-Vista-VistaModelo (MVVM) permite separar claramente la lógica de negocio y la presentación de la interfaz de usuario, lo que facilita el mantenimiento, pruebas, escalabilidad y desarrollo colaborativo. Este patrón fue desarrollado como alternativa a MVC, destacando por desligar la vista del modelo para facilitar las pruebas unitarias y mantener la programación modular. [15]

Los componentes que forman parte del patrón MVVM son:

- El modelo es el responsable del acceso a la fuente de datos y de trabajar con esos datos.
- La vista representa los datos de forma pertinente, reflejando el estado de los datos y recibiendo los eventos y las interacciones del usuario.
- Y la Vista del modelo es responsable de representar la forma en que se espera que sea una vista y como se comportará ante las interacciones que tiene con el usuario. Además, describe el conjunto de principios y estructuras que presentan los datos recuperados del modelo. Como último trabajo, es el puente de comunicación entre el modelo y la vista.

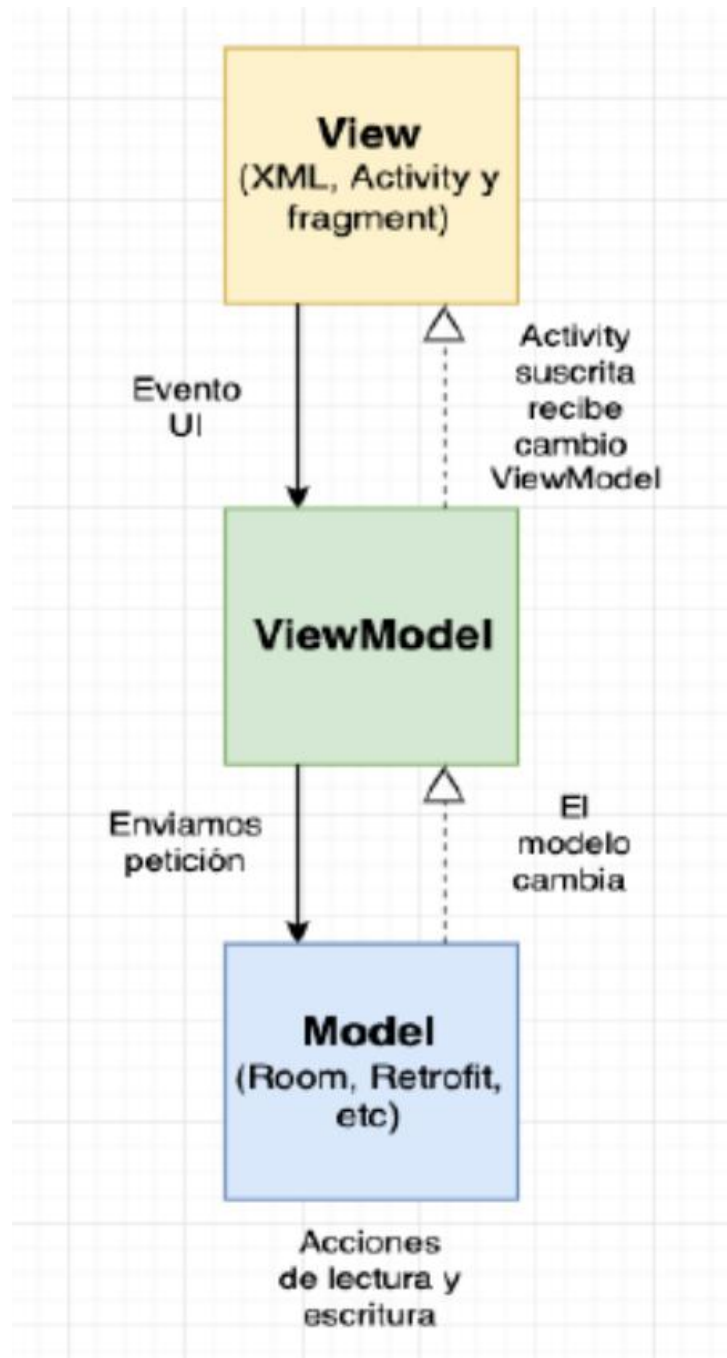


Imagen 6: Componentes MVVM [15]

Arquitectura Limpia (Clean Architecture)

La Arquitectura Limpia es una filosofía de diseño que organiza el software en capas concéntricas, donde cada capa tiene diferentes responsabilidades. La regla principal, llamada "regla de dependencia", establece que las dependencias de

código solo pueden ir de los niveles externos hacia los internos, evitando que el círculo interno conozca los detalles de los niveles externos.

Los principios fundamentales que sigue esta arquitectura se resumen en las siglas SOLID:

- **S:** Principio de responsabilidad única, cada clase debe tener una única responsabilidad.
- **O:** Principio abierto/cerrado, las entidades deben ser abiertas para la extensión, pero cerradas para la modificación.
- **L:** Principio de sustitución de Liskov, los objetos deben ser reemplazables por instancias de sus subtipos sin alterar el funcionamiento.
- **I:** Principio de segregación de interfaces, es mejor utilizar muchas interfaces específicas en lugar de una sola interfaz general.
- **D:** Principio de inversión de dependencia, se debe depender de abstracciones, no de implementaciones.

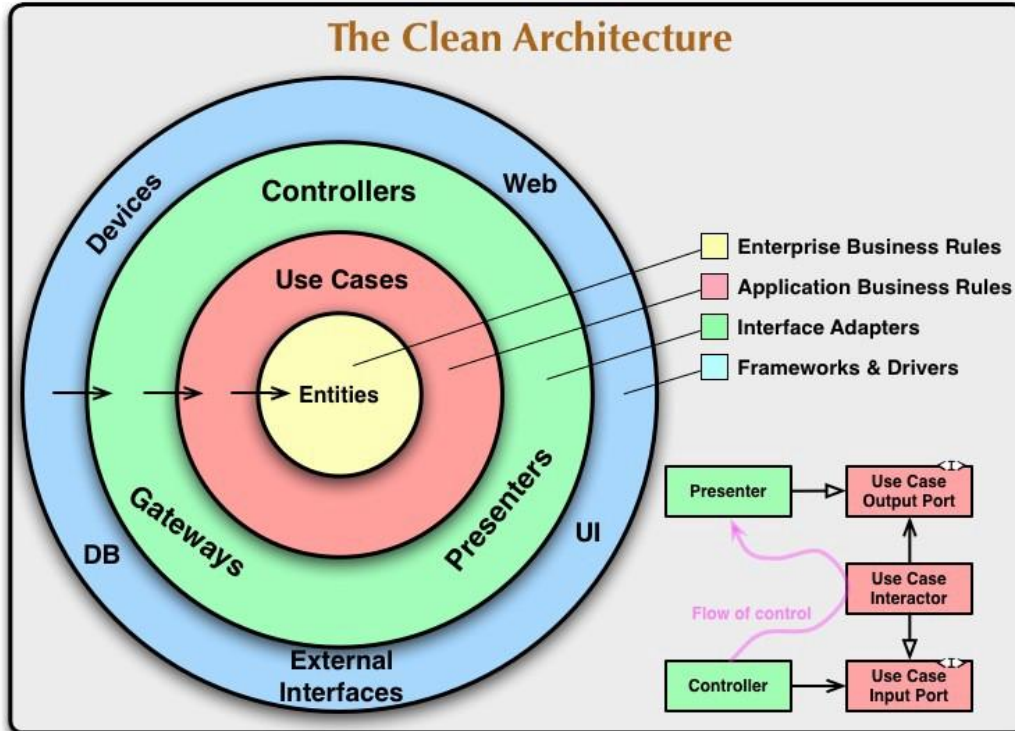


Imagen 7: Capas de la Arquitectura Limpia [15]

Arquitectura Cliente-Servidor

La arquitectura cliente-servidor es un modelo para el desarrollo de sistemas de información, en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información servicios o recursos. Un cliente se puede definir como aquel que solicita un servicio y un servidor es definido como proveedor de servicios. [16]

Los principales componentes de la arquitectura de Cliente-Servidor son los siguientes:

- El Servidor se encargar de esperar continuamente las peticiones del cliente, por lo que mientras el servidor no reciba ninguna petición este estará en modo de espera y cuando este reciba alguna petición, el servidor despierta y atiende al cliente. Una vez terminando el servicio el servidor vuelve a su estado de espera.
- El Cliente es aquella entidad activa de una conexión puesto que este mismo es el que toma iniciativa para demandar un servicio del servidor.
- La Infraestructura se refiere a los elementos tantos de hardware como de software los cuales garantizan la conexión física y la transferencia de datos entre los equipos de red permitiendo la comunicación entre el cliente y servidor. [16]

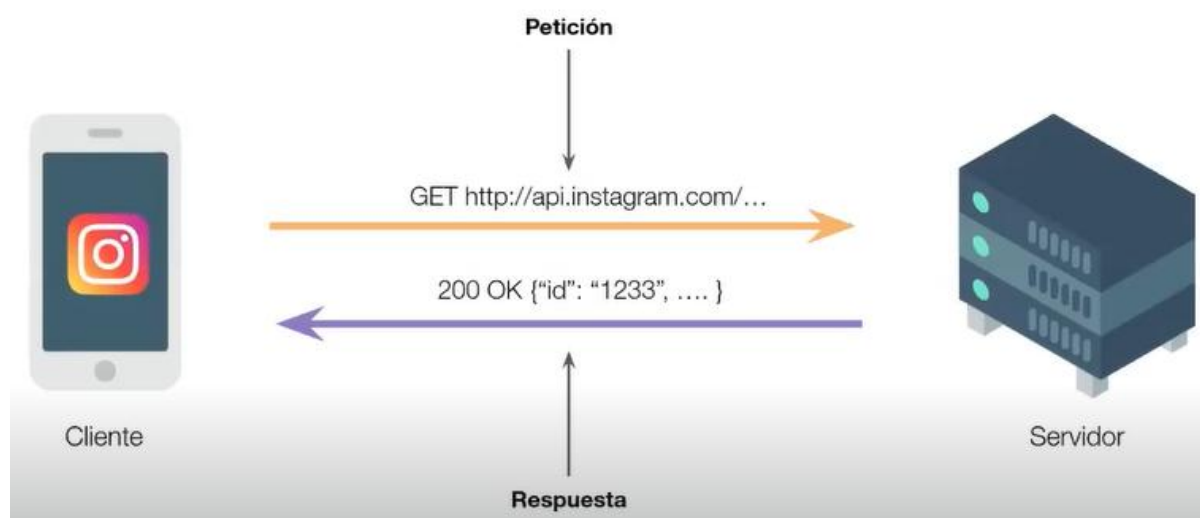


Imagen 8: Capas de la Arquitectura Limpia [15]

Arquitectura REST

La arquitectura REST (Representational State Transfer) es un estilo arquitectónico ampliamente utilizado para el diseño de servicios web, que se centra en la simplicidad, escalabilidad y separación entre cliente y servidor. Se organiza en niveles de madurez, cada uno aportando características adicionales que mejoran su funcionalidad y flexibilidad [17].

Niveles de la arquitectura REST

1. Nivel 0 (Swamp of POX):

- a. Intercambio de mensajes en formato XML (POX = Plain Old XML) utilizando el método POST de HTTP.
- b. Similar a SOAP, pero es complejo y tiene un fuerte acoplamiento entre cliente y servidor, dependiendo de WSDL para definir cómo interactuar.

2. Nivel 1 (Recursos):

- a. Introduce el concepto de "Recurso" (e.g., Facturas, Clientes).
- b. Cada recurso tiene un grupo de URLs para realizar operaciones básicas como inserción, borrado, actualización y consulta, generalmente usando JSON.
- c. Carece de convenciones claras para nombrar URLs, lo que puede generar APIs caóticas.

3. Nivel 2 (Verbos HTTP):

- a. Define convenciones para usar los métodos HTTP estándar:
 - i. GET: Obtener datos.
 - ii. POST: Insertar datos.
 - iii. PUT: Actualizar datos.
 - iv. DELETE: Eliminar datos.
- b. Proporciona una estructura más organizada para las APIs, con URLs más intuitivas y consistentes.

4. Nivel 3 (HATEOAS - Hypermedia as the Engine of Application State):

- a. Introduce la relación entre recursos mediante hipervínculos en las respuestas.
- b. Permite a los clientes descubrir recursos relacionados y navegar dinámicamente por la API, reduciendo el acoplamiento entre cliente y servidor.

Características clave de REST

1. **Simplicidad y estandarización:** Uso de métodos HTTP y formatos ligeros como JSON.
2. **Separación de responsabilidades:** Cliente y servidor están desacoplados, lo que facilita el desarrollo independiente.
3. **Autoexploración (HATEOAS):** Los clientes pueden navegar por los recursos utilizando enlaces proporcionados en las respuestas.
4. **Escalabilidad:** REST está diseñado para ser eficiente en sistemas distribuidos y escalables. [17]

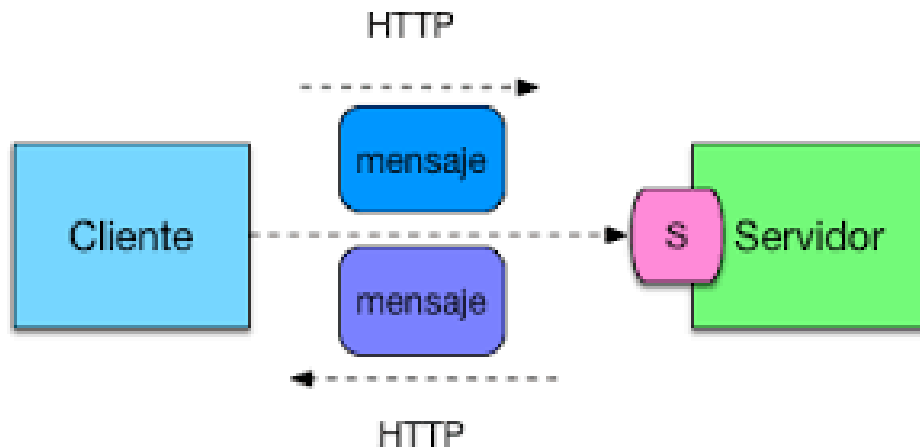


Imagen 9: Esquema Básico de Comunicación REST entre Cliente y Servidor [18]

Paradigmas y Lenguajes de Programación

Los paradigmas de programación representan modelos matemáticos que influyen en la manera de resolver problemas, frecuentemente con el uso de tecnología informática, aunque no siempre es necesario. Los lenguajes de programación, por su parte, son conjuntos de instrucciones utilizados para desarrollar programas

basados en diferentes paradigmas o combinaciones de ellos. Estos lenguajes son reconocidos a nivel internacional y disponen de herramientas como compiladores, editores y entornos de desarrollo integrados, que pueden ser de acceso libre o de uso privativo. [19]

Tipos de Paradigmas de Programación

En el ámbito del desarrollo de aplicaciones móviles, los paradigmas de programación juegan un papel crucial al influir en cómo los desarrolladores abordan la creación de soluciones innovadoras y eficientes para dispositivos móviles. Estos paradigmas permiten elegir el enfoque más adecuado para optimizar recursos, mejorar la experiencia del usuario y facilitar el mantenimiento del código.

Paradigma Imperativo

El paradigma imperativo fue el primer paradigma aceptado formalmente y el más representativo de la programación ya que está basado en enunciados imperativos, es decir, por comandos u órdenes. Al paso del tiempo llego Fortran que a partir de él la programación imperativa permitió la implementación de programas complejos. [20]

La programación imperativa se rige por 2 conceptos básicos fundamentales para la construcción de programas tales como el módulo y la estructura. Además, se señala que un programa debe dividirse en subprogramas para hacerlos legible y manejable. [20]

Paradigma Orientado a Objetos

La Programación Orientada a Objetos representa un cambio significativo en la forma de concebir el desarrollo de software, ofreciendo un nivel superior de abstracción que mejora notablemente la calidad del código final. De manera general, este paradigma introduce elementos clave como las clases y los objetos, que permiten modelar el mundo desde una perspectiva centrada en las entidades en lugar de las acciones. Los datos se encapsulan dentro de las clases, lo que garantiza un acceso controlado e independiente de cómo se representan internamente. Esto reduce las interdependencias entre las distintas partes del sistema, facilitando tanto su mantenimiento como su evolución. Además, herramientas como la herencia, la

composición y el polimorfismo simplifican el desarrollo al posibilitar la creación de nuevas clases a partir de otras, promoviendo así la reutilización de código y la eficiencia en el diseño de sistemas. [21]

Paradigma Funcional

El paradigma de programación funcional es un estilo de programación que se centra en el uso de funciones matemáticas para resolver problemas y evitar estados intermedios o modificación de datos durante la ejecución del programa. A diferencia del paradigma imperativo (como C o Java), que depende del uso de variables y secuencias de instrucciones, la programación funcional aplica funciones a datos de manera pura, lo cual significa que los resultados de las funciones dependen únicamente de sus entradas y no de un estado compartido. [22]

Este paradigma se basa en la evaluación continua de funciones sobre funciones, eliminando la necesidad de variables. Algunas de sus características principales incluyen:

- **Funciones de Orden Superior:** Las funciones pueden ser tratadas como datos, es decir, pueden ser pasadas como argumentos y devueltas como resultado.
- **Evaluación Perezosa:** Las expresiones son evaluadas solo cuando se necesitan, lo cual permite trabajar con estructuras como listas infinitas.
- **Transparencia Referencial:** Las funciones siempre devuelven el mismo resultado dado el mismo conjunto de entradas, lo cual facilita el razonamiento sobre el código.
- **Recursión:** La programación funcional suele depender de la recursión para iterar, en lugar de los bucles tradicionales.

Paradigma Declarativo

El paradigma de programación declarativa se enfoca en describir qué problema se quiere resolver sin especificar cómo resolverlo en términos de pasos o instrucciones detalladas. A diferencia del paradigma imperativo, donde se indica paso a paso cómo llegar a un resultado, el paradigma declarativo se centra en expresar la lógica

del problema, dejando la parte de control (cómo se debe realizar) a la implementación subyacente del lenguaje. [23]

En la programación declarativa:

- **Qué hacer:** El programador describe el resultado que espera sin definir una secuencia de acciones detalladas. Por ejemplo, en lugar de escribir un bucle para buscar un elemento en una lista, simplemente se especifica "quiero el elemento que cumpla con esta condición".
- **Cómo hacerlo:** No se especifica explícitamente cómo el programa debe gestionar la memoria, el orden de las instrucciones, o los bucles para lograr el objetivo. Estos detalles se manejan automáticamente. [23]

Este paradigma tiene dos subparadigmas principales:

- **Programación funcional:** Se enfoca en usar funciones para describir cómo transformar datos, sin modificar el estado ni utilizar variables. Por ejemplo, Haskell y Lisp son lenguajes funcionales que se basan en la aplicación de funciones.
- **Programación lógica:** Describe problemas en términos de reglas lógicas y hechos. Los lenguajes como Prolog permiten definir reglas lógicas y luego realizar consultas, dejando al motor de inferencia determinar la secuencia de pasos para llegar a una solución. [23]

Paradigma Lógico

El paradigma de programación lógica se basa en describir el problema utilizando hechos y reglas que expresan lo que se conoce como verdadero o falso respecto a una situación, en lugar de detallar los pasos específicos para llegar a una solución. Este enfoque se enfoca más en la lógica y el conocimiento sobre el problema, y se diferencia de otros paradigmas en que no se indica cómo debe realizarse la computación paso a paso. El control de flujo y la búsqueda de soluciones son manejados por un motor de inferencia, que se encarga de deducir la respuesta a partir de las reglas y hechos que se proporcionan. [24]

Algunas de las características clave de la programación lógica incluyen:

- **Declaración de Hechos y Reglas:** Un programa lógico está compuesto por un conjunto de hechos y reglas. Los hechos representan lo que es verdadero y las reglas definen relaciones y condiciones que deben cumplirse. [24]
- **Sin Instrucciones Secuenciales:** A diferencia de la programación imperativa, la programación lógica no necesita especificar el orden de ejecución ni los detalles de implementación, como los bucles o el manejo de memoria. En lugar de eso, se enfoca en describir las condiciones y las relaciones entre los elementos del problema. [24]

Paradigma Multiparadigma

La programación multiparadigma consiste en combinar diferentes paradigmas de programación en un mismo proyecto, seleccionando el paradigma que sea más natural y eficiente para cada subproblema específico. Esta combinación permite mejorar la producción en el desarrollo de software y facilita la resolución de problemas complejos, ya que cada paradigma tiene sus propias fortalezas y debilidades, y ningún enfoque por sí solo puede resolver todos los problemas posibles. [25]

En la práctica, los problemas suelen tener múltiples aspectos que pueden ser mejor abordados desde diferentes enfoques, y el uso de un único paradigma puede limitar la capacidad de adaptación a todas las necesidades del proyecto.

El enfoque multiparadigma permite a los desarrolladores utilizar la mejor herramienta para cada tarea, integrando enfoques que se complementan entre sí. Esto resulta especialmente útil cuando la realidad es muy compleja y requiere la combinación de distintas técnicas y metodologías para cubrir todos los aspectos del problema. [25]

Por ejemplo, algunas partes de un problema pueden beneficiarse del enfoque orientado a objetos, mientras que otras pueden requerir un enfoque funcional o concurrente.

Paradigmas de Lenguajes de Programación de este Proyecto

Dart

Dart es un lenguaje de programación orientado a objetos, notable por su enfoque de tipado fuerte y soporte a múltiples paradigmas. Ofrece la flexibilidad de un lenguaje multiparadigma, mientras adopta la rigidez de un tipado fuerte mediante el uso de variables de tipos específicos. [26]

JavaScript

En realidad, JavaScript es un lenguaje de programación de propósito general, dinámico y con características del paradigma de orientación a objetos. Es capaz de realizar prácticamente cualquier tipo de aplicación y se ejecutará en el navegador del cliente [27]

Frameworks

Un framework también conocido como marco de trabajo, es un entorno que simplifica y hace que la programación sea más sencilla para el desarrollador dado que le asegura buenas prácticas y obtiene un código consistente. [28]

Framework React Native

React Native es un framework basado en JavaScript el cual está hecho para desarrollar aplicaciones móviles híbridas de presentación nativa tanto para iOS como para Android. Los tiempos van cambiando y la idea de desarrollar aplicaciones para iOS como para Android con un solo lenguaje se está volviendo realidad. [28]

React Native se introdujo por primera vez en 2015 y continúa desarrollándose para optimizar el framework, esta plataforma fue construida sobre el framework de React el cual fue creado para el desarrollo de páginas web y ahora con React Native se pueden crear aplicaciones móviles para iOS y Android. [29]

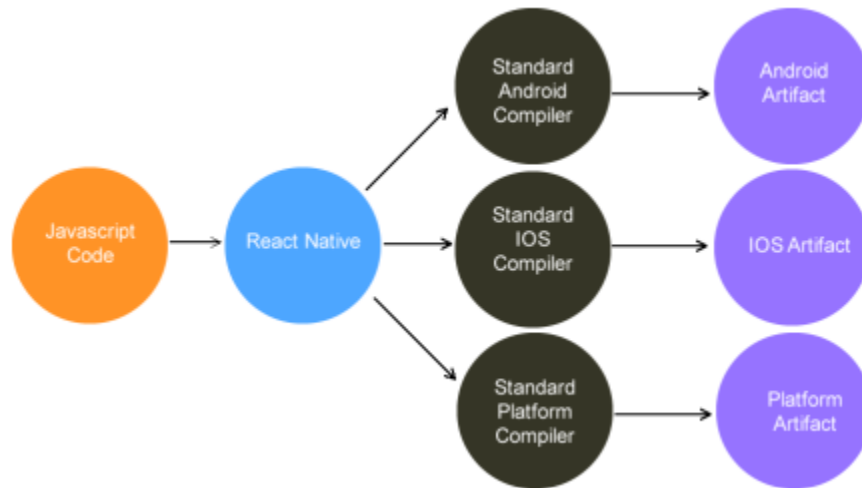


Imagen 10: Flujo de Compilación en React Native [29]

Modelo de Hilo

El renderizado de React Native utiliza 3 subprocesos los hilos son:

- A. *Subproceso de interfaz de usuario: Este subproceso es el único el cual puede manejar componentes nativos, además de ser responsable de representar las componentes en la pantalla del dispositivo.*
- B. *Hilo de JavaScript: Este hilo es responsable de ejecutar toda la lógica que el programador desarrollo.*
- C. *Hilo de fondo: Este hilo está dedicado solo para los cálculos de diseño.* [30]

Arquitectura antigua

El componente principal de lo que era la arquitectura antigua de React Native es el “Bridge”, el cual es un puente entre el código desarrollado en JavaScript y la parte nativa del dispositivo. Toda la comunicación se realiza mediante el puente el cual es capaz de enviar únicamente el contenido serializado como JSON, es decir, que después de recibir el mensaje es necesario decodificarlo para que el hilo nativo pueda ejecutar el código nativo, además de que la comunicación es siempre asíncrona.

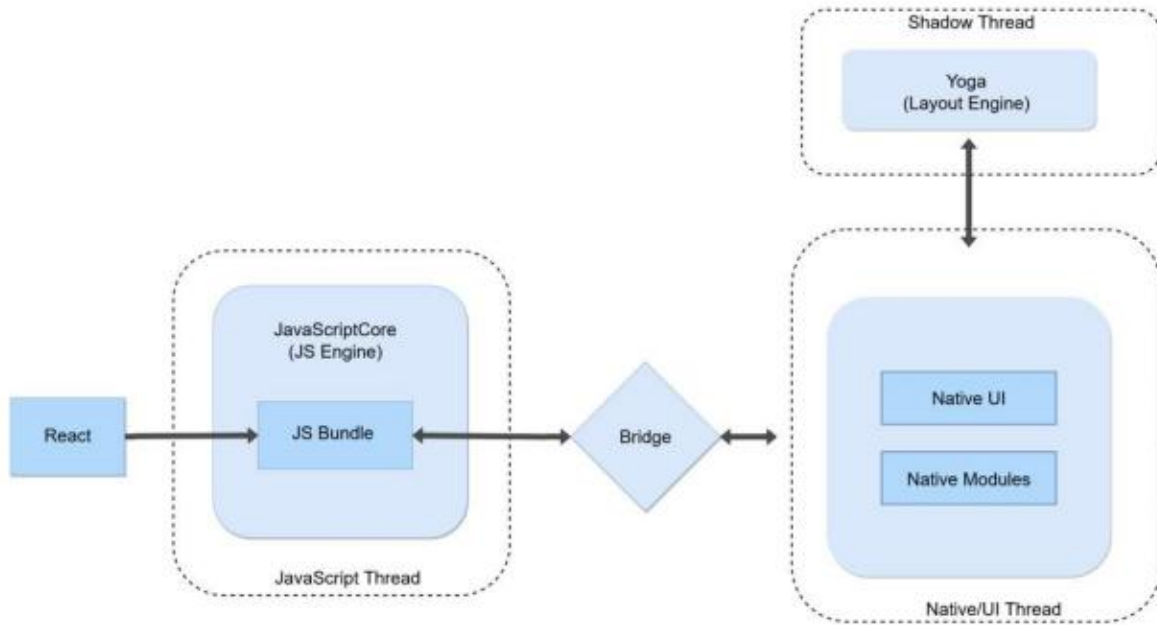


Imagen 11: Arquitectura Antigua de Ejecución en React Native [31]

Arquitectura Nueva

En la nueva arquitectura de React Native existe un componente muy importante para el renderizado llamado “Fabric”, este fue desarrollado en C++ y es el responsable de interactuar con el dispositivo para representar visualmente el contenido de la aplicación. La comunicación entre JavaScript y Fabric se realiza de manera sincrónica a través de la “Interfaz JavaScript” y de esta manera puede hacer referencia directamente con los métodos nativos.

A diferencia de la arquitectura anterior, ya no es necesario instalar módulos porque JavaScript cuenta con una referencia a estos módulos desde la interfaz “Turbo Modules”, de esta manera ahorra tiempo al iniciar la aplicación permitiendo una comunicación más eficiente con los módulos nativos.

Además, existe otro componente importante llamado “CodeGen” ya que JavaScript es un lenguaje de tipo dinámico y C++ es de tipo estático es de suma importancia mantener una comunicación fluida entre los 2 lenguajes y para lograr esto se utiliza el CodeGen el cual define las interfaces utilizadas en “Turbo Modules” y “Fabric”.
[30]

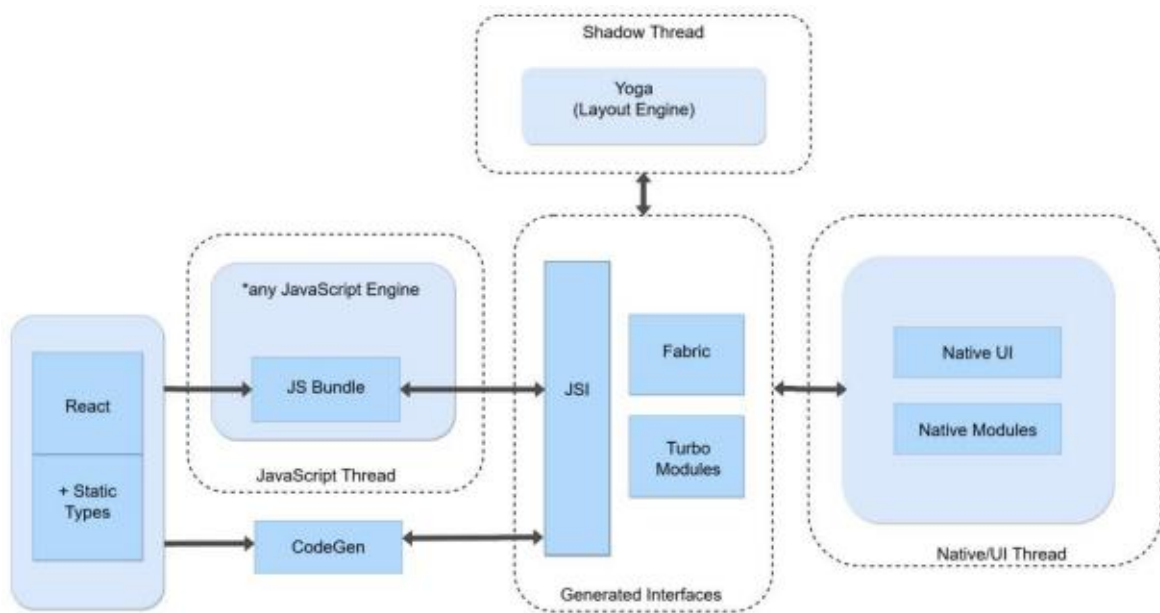


Imagen 12: Arquitectura Moderna de React Native con JSI [31]

JavaScript

JavaScript es un lenguaje de programación diseñado para añadir dinamismo e interactividad a las páginas web. A diferencia de otros lenguajes como Java, JavaScript es interpretado directamente por el navegador en lugar de ser compilado. Permite que las páginas respondan a acciones del usuario, como cambiar fotos, abrir ventanas y generar cálculos. [31]

JavaScript se estructura en tres partes principales:

- **Variables:** Los tipos de datos soportados por JavaScript son los numéricos, lógicos, cadenas de caracteres y tipos nulos.
- **Objetos:** Los objetos son un contenedor dado para una colección de valores e incluso puede estar construido de otros objetos.
- **Funciones:** Las funciones son procedimientos que una aplicación puede ejecutar cuando la llame.

Cabe resalta que JavaScript es un lenguaje de Scripts que permite a los usuarios personalizar aplicaciones mediante pequeños programas llamados scripts. Estos scripts se interpretan línea por línea, lo cual facilita su ejecución sin necesidad de compilar. JavaScript se parece a lenguajes como C, C++, Pascal y Delphi.

JavaScript está basado en objetos, lo que facilita la organización del código en partes reutilizables que realizan tareas específicas. A pesar de no ser un lenguaje orientado a objetos tradicional, tiene bibliotecas de objetos que permiten crear otros nuevos.

Además, JavaScript maneja eventos, es decir, responde a acciones del usuario como hacer clic, arrastrar, o apuntar con el mouse. También es independiente de la plataforma, ya que puede ejecutarse en cualquier navegador, sin importar el sistema operativo.

Virtual DOM

El Virtual DOM es una versión ligera del DOM real, que permite a React manipular y actualizar la interfaz de usuario de manera eficiente. Cuando el usuario interactúa con la aplicación, los eventos modifican la estructura del Virtual DOM. React toma una instantánea del Virtual DOM antes de hacer cualquier actualización, y luego usa un algoritmo de diferenciación para comparar la nueva versión con la anterior y detectar qué partes necesitan ser actualizadas en el DOM real. Este proceso se denomina conciliación. [32]

Además, React agrupa todas las operaciones de lectura/escritura en el DOM para minimizarlas y mejorar el rendimiento. Solo después de haber encontrado la cantidad mínima de cambios necesarios, React actualiza el DOM real en un solo paso, lo que optimiza el proceso de renderización.

La conciliación y la renderización son fases separadas en React, lo que permite que tanto React como React Native compartan el mismo conciliador, pero puedan tener diferentes mecanismos de renderización.

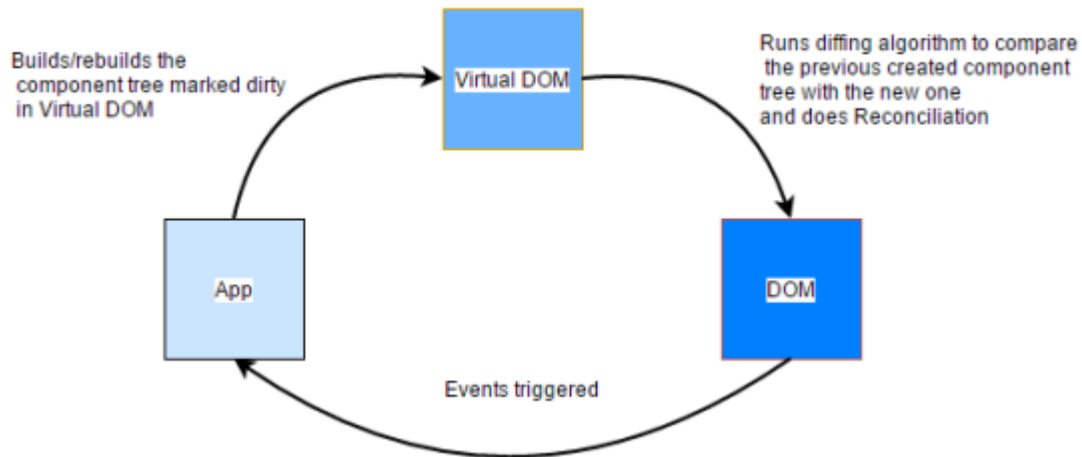


Imagen 13: Ciclo de Reconciliación del Virtual DOM en React [32]

Framework Flutter

Flutter es una tecnología híbrida creada por la empresa Google y basada en el lenguaje Dart, el cual permite crear aplicaciones nativas tanto para Android como para iOS en poco tiempo y con una interfaz amigable. [4]

Además, Flutter cuenta con un kit de herramientas de interfaz lo cual permite el desarrollo de aplicaciones nativas multiplataforma con una sola base de código. A diferencia de otras soluciones como las basadas en navegador (que utilizan WebView) o React Native (que usa componentes nativos), Flutter renderiza su propia interfaz de usuario mediante su motor interno, proporcionando flexibilidad total a los desarrolladores para crear y personalizar widgets. Este enfoque le permite a Flutter ofrecer una velocidad y capacidad de personalización sin compromisos, lo que lo diferencia de las soluciones tradicionales de desarrollo multiplataforma.

Flutter está diseñado como un sistema modular en capas. Consiste en una serie de bibliotecas independientes, cada una de las cuales depende de la capa inferior. Cada nivel del framework ha sido creado para ser opcional y puede ser reemplazado según sea necesario. [33]

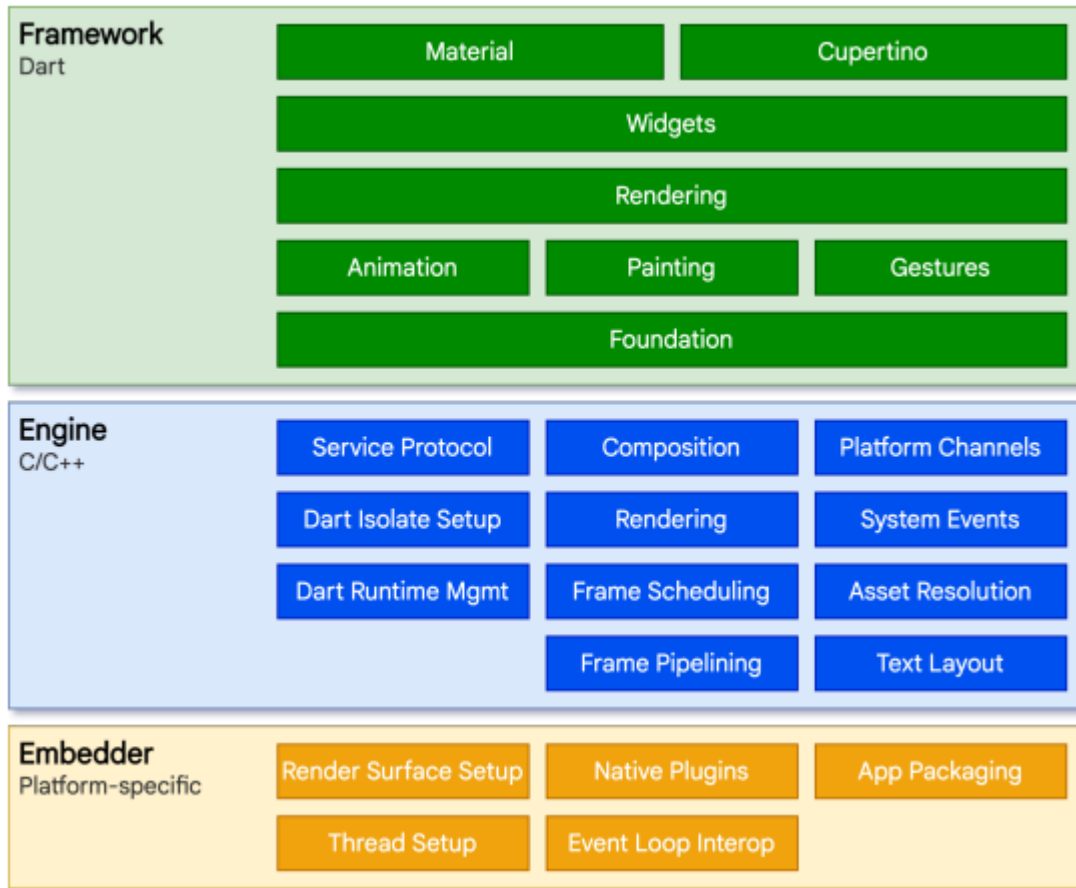


Imagen 14: Arquitectura de Flutter [33]

Dart

Dart es un lenguaje de programación orientado a objetos y con análisis estático de tipo creado por Google. La programación Dart fue creada como una alternativa a remplazar a JavaScript y convertirse en el lenguaje principal para los navegadores sin embargo este mismo sigue en mejoras y adaptaciones. [4]

Por otro lado, en mayo del 2017 Google lanzó públicamente Dart Flutter el cual es un marco para desarrollo multiplataforma el cual utiliza Dart para la creación de aplicaciones móviles tanto para iOS como para Android.

Características del lenguaje Dart:

- Lenguaje fácil de aprender, dado que es un lenguaje realmente sencillo y en su propio sitio web cuenta con una variedad de tutoriales para su entendimiento, además de permitir colaboraciones de otros desarrolladores.

- Lenguaje basado en POO (programación orientada a objetos) ya que gracias a sus basamentos en clases o en POO facilita la encapsulación y la reutilización de código.
- Adaptación a cualquier navegador ya que el lenguaje de programación Dart se puede ejecutar de dos maneras, en una máquina virtual o en un motor de JavaScript utilizando un compilador para traducir el código por lo que esto le permite adaptarse a cualquier navegador. [4]

Widgets

Un widget puede definir cualquier aspecto de la vista de una aplicación. Algunos widgets, como Row, definen aspectos del diseño. Otros son menos abstractos y definen elementos estructurales, como Button y TextField. Incluso la raíz de tu aplicación es un widget. [34]

Los widgets están optimizados por el equipo de Flutter para asegurar un buen rendimiento, recomendándose su uso. Por ejemplo, el widget ListView solo muestra los elementos visibles, descartando los no visibles para ahorrar recursos. Estas optimizaciones a nivel de widget contribuyen al rendimiento general de las aplicaciones, ya que Flutter ha sido diseñado con un enfoque en la eficiencia desde sus cimientos.

Si bien existe una gran variedad de widgets que pueden ser utilizados, particularmente existen 2 tipos widgets uno de estos es llamado StatefulWidget y el otro es llamado StatelessWidget y aunque los dos son widgets tiene un eso diferente tales como:

- Un **StatelessWidget** no mantiene ningún estado interno. Solo muestra información o interfaz de usuario, y no necesita cambios dinámicos. Si cambia algo, es el framework quien lo reconstruye. Un ejemplo es un botón simple como AddToCartButton, que solo espera ser presionado y no guarda información más allá de esa interacción.
- Un **StatefulWidget**, por otro lado, sí mantiene y gestiona un estado interno que puede cambiar durante el ciclo de vida del widget. Tiene un objeto State asociado que controla cuándo se debe repintar el widget al cambiar su

estado. Un ejemplo es QuantityCounter, que rastrea dinámicamente la cantidad de artículos en un carrito. [34]

APIS

Una API (Interfaz de Programación de Aplicaciones) es un conjunto de subrutinas, funciones y procedimientos que permiten la interacción entre diferentes aplicaciones o servicios. Actúa como una capa de abstracción que expone interfaces para ser utilizadas por otros programas, facilitando la comunicación y la orquestación de servicios. Las APIs son independientes del lenguaje de programación, lo que permite su portabilidad y uso en diversas tecnologías. [35]

Inicialmente, las APIs se crearon para cambiar la forma en que los usuarios y las empresas interactuaban en el ámbito digital, facilitando el manejo de información y el acceso a otros servicios. Sin embargo, su papel ha ido mucho más allá. Actualmente, las APIs no solo simplifican esas interacciones, sino que también son una herramienta clave para impulsar el crecimiento de los negocios. Ofrecen la posibilidad de gestionar datos, analizar métricas, generar reportes, optimizar el uso de recursos y administrar distintas áreas dentro de una empresa. Además, permiten evaluar la satisfacción de los usuarios, lo que contribuye a mejorar significativamente su experiencia.

Existen varios tipos de APIs, cada uno con un propósito específico:

- **APIs de servicios web o remotas:** Utilizadas en aplicaciones web para el intercambio de información a través de HTTP o HTTPS. Las más comunes incluyen SOAP, XML-RPC, JSON-RPC y REST.
- **APIs de código fuente o locales:** Proporcionan bibliotecas de objetos para que el software dentro de un mismo entorno se comunique, como en J2EE o .NET.
- **APIs heredadas:** Usan protocolos antiguos como CORBA, pero siguen siendo utilizadas en algunas interfaces de aplicaciones.
- **APIs de producción:** Gestionan datos reales desde una base de datos y aplican la lógica comercial, asegurando que los usuarios no accedan a información no autorizada.

- **APIs de desarrollo:** También llamadas "APIs falsas", permiten el desarrollo del front-end sin datos reales o bases de datos.
- **APIs abiertas:** También conocidas como APIs públicas, tienen medidas de seguridad relajadas para facilitar el acceso a desarrolladores externos.
- **APIs de socios:** Restringidas a desarrolladores autorizados, con seguridad y autenticación más estrictas.
- **APIs internas:** Usadas solo dentro de una organización para intercambiar datos de manera segura.
- **APIs compuestas:** Combinan múltiples APIs, facilitando la integración de datos y mejorando el rendimiento. [36]

APIs REST (Representational State Transfer) son un estilo de arquitectura que facilita la comunicación entre sistemas a través de solicitudes HTTP. Son flexibles y compatibles con múltiples lenguajes de programación y formatos de datos como JSON. Las características clave de las APIs REST incluyen:

- **Interfaz uniforme:** Las solicitudes son consistentes y utilizan identificadores de recursos únicos (URI).
- **Separación cliente-servidor:** El cliente solo necesita el URI del recurso; el servidor maneja las solicitudes.
- **Sin estado:** Cada solicitud incluye toda la información necesaria, sin necesidad de mantener sesiones en el servidor.
- **Almacenamiento en caché:** Permite guardar recursos para optimizar el rendimiento.
- **Sistema de capas:** Las APIs REST están organizadas en capas para gestionar llamadas y respuestas entre cliente y servidor.
- **Código bajo demanda (opcional):** Pueden enviar código ejecutable al cliente cuando se necesite. [36]

Rendimiento

El rendimiento en el desarrollo de software se refiere a la capacidad de los componentes de operar eficientemente sin comprometer la estabilidad del sistema completo. Evaluar esta característica es fundamental en el desarrollo de

aplicaciones móviles, ya que un componente con bajo rendimiento puede degradar el desempeño global de la aplicación. Para garantizar la calidad, se realizan pruebas de rendimiento, las cuales verifican que los componentes cumplan con las expectativas de eficiencia y respuesta. [37] Diversas herramientas, como Xcode Instruments, Flutter DevTools y Android, permiten evaluar el rendimiento en aplicaciones móviles, aunque dependerá el uso correspondiente al lenguaje aplicado para el desarrollo.

Android Profiler

Android Profiler es una herramienta de Android Studio utilizada para identificar y corregir problemas de rendimiento en aplicaciones. Permite detectar áreas de ineficiencia en el uso de recursos, como CPU, memoria, gráficos y batería, proporcionando una visión detallada del comportamiento de la aplicación. Se pueden realizar perfiles en dos tipos de aplicaciones: profileable (para tareas comunes de profiling sin sobrecarga de rendimiento) y debuggable (para tareas más avanzadas con un costo de rendimiento adicional). Android Profiler ayuda a mejorar el rendimiento general de una aplicación al ofrecer información precisa sobre cómo los recursos son utilizados durante la ejecución. [38]

Xcode Instruments

Xcode Instruments es una herramienta que permite realizar perfiles detallados de una aplicación utilizando plantillas específicas para distintos tipos de problemas de rendimiento, como problemas de memoria, CPU, consumo de energía, I/O, y redes. Proporciona información detallada sobre el comportamiento de la aplicación en dispositivos reales para obtener mediciones más precisas. Instruments ayuda a identificar el código que está causando problemas de rendimiento, permitiendo al desarrollador crear un plan para mejorarlo, implementar los cambios y comparar los resultados antes y después para asegurar mejoras. [39]

Flutter DevTools

Flutter DevTools es un conjunto de herramientas de depuración y rendimiento para aplicaciones Flutter y Dart. Permite inspeccionar el diseño y estado de la interfaz de usuario, diagnosticar problemas de rendimiento relacionados con la fluidez de la UI

(UI jank), realizar perfiles de CPU y red, depurar problemas de memoria, y analizar el tamaño del código y la aplicación. También se pueden ver registros y diagnósticos generales y validar enlaces profundos (deep links) en aplicaciones Android. Flutter DevTools está diseñado para ser utilizado junto con el flujo de trabajo del entorno de desarrollo integrado (IDE) o la línea de comandos, ayudando a los desarrolladores a identificar y solucionar problemas de rendimiento y depuración de manera eficiente. [40]

Facilidad de Aprendizaje

La facilidad de Aprendizaje se refiere al nivel de esfuerzo, tiempo y recursos necesarios para que los desarrolladores adquieran el conocimiento necesario para utilizar un framework de desarrollo de manera efectiva. Además, la curva de aprendizaje de un framework es un factor crítico en su adopción, ya que influye directamente en la productividad y eficiencia del equipo. Un framework con una curva de aprendizaje demasiado pronunciada puede desmotivar a los desarrolladores, mientras que uno más accesible permite una adopción más rápida, lo que incide de forma positiva en variables relacionadas con la entrega oportuna de proyectos y la optimización de costos y esfuerzos. [41]

METODOLOGÍA

Para llevar a cabo el primer objetivo “Desarrollar una aplicación móvil para Android y iOS con el framework de Flutter” se implementó el Ciclo de Vida de Desarrollo de Software (SDLC) para guiar las diferentes etapas del proyecto, desde el análisis hasta la implementación y pruebas. A continuación, se describe cada etapa de este ciclo aplicada al desarrollo de la aplicación móvil:

Ciclo de Vida de Desarrollo de Software para la Aplicación de Flutter

1. Análisis
 - a. Requisitos funcionales.
 - b. Requisitos no funcionales.
2. Diseño
 - a. Diseño de interfaz de usuario UI.
 - b. Diagrama de componentes.
 - c. Diagrama de objetos.
3. Implementación (Codificación)
 - a. Configuración del entorno de desarrollo.
 - b. Codificación.
 - i. Módulos.
 1. Módulo de consumo de API de TheMovieDB.
 2. Módulo de favoritos con ISAR.
4. Pruebas
 - a. Velocidad de ejecución.
 - b. Consumo de recursos (CPU y RAM).
 - i. Flutter DevTools.
 - ii. Xcode Instruments.

A continuación, se detallará cada una de las etapas realizadas para el desarrollo de la aplicación móvil para Android e iOS con el framework de Flutter

Análisis

Requisitos funcionales

1. Visualización de películas: La aplicación debe mostrar una lista de películas obtenidas desde la API.
2. Detalle de película: Al seleccionar una película, se debe mostrar su descripción y los actores que participaron en ella.
3. Agregar a favoritos: El usuario debe poder marcar una película como favorita.
4. Visualización de favoritos: La aplicación debe contar con un apartado donde el usuario pueda ver todas las películas que ha marcado como favoritas.
5. Sincronización con la API: La aplicación debe poder realizar peticiones a la API para obtener la lista actualizada de películas.
6. Gestión de estado: La aplicación debe recordar las películas marcadas como favoritas, incluso si se cierra y se vuelve a abrir.

Requisitos no funcionales

1. Compatibilidad y soporte multiplataforma: La aplicación debe funcionar en dispositivos Android e iOS, con una experiencia de usuario consistente en ambas plataformas para facilitar la comparación.
2. Facilidad de uso y navegación: La aplicación debe contar con una interfaz intuitiva que permita al usuario entender fácilmente cómo ver películas, agregar a favoritos y navegar entre secciones.
3. Uso de almacenamiento local: Debe optimizar el uso de almacenamiento local para los datos de favoritos, asegurando que el rendimiento no se vea afectado y que los datos se guarden de manera persistente incluso después de cerrar la aplicación.

Requisitos Técnicos

1. Sistema Operativo:
 - a. Android: La aplicación debe ser compatible con Android 10 (API nivel 29) y superior.
 - b. iOS: La aplicación debe ser compatible con iOS 12 y superior.
2. Plataformas de arquitectura:

- a. Flutter: Usar Flutter SDK versión 3.0 o superior para desarrollo multiplataforma. Dispositivos físicos de prueba:
- 3. Dispositivos físicos de prueba:
 - a. Android: Un dispositivo de gama media representativo, como el Samsung Galaxy A52 (Android 11, 6GB RAM, procesador Snapdragon 720G).
 - b. iOS: Un dispositivo de gama media en iOS, como el iPhone 11 con iOS 14 o superior.
- 4. Entorno de desarrollo integrado (IDE):
 - a. Flutter: Visual Studio Code con los plugins necesarios de Flutter y Dart.
- 5. Herramientas de perfilado de rendimiento específicas:
 - a. Flutter: DevTools para análisis de rendimiento y uso de CPU/memoria.
- 6. Requerimientos de red:
 - a. Conectividad constante a internet para las pruebas de sincronización de datos con la API de películas.
- 7. Administración de dependencias:
 - a. Flutter: Manejo de dependencias mediante pubspec.yaml.

Diseño

Diseño de interfaz de usuario UI

A continuación, mostraremos brevemente el diseño pensado para la aplicación a desarrollar con el fin de mantener una interfaz limpia y fluida para la medición del rendimiento, con base a esto el diseño para implementar es el siguiente:

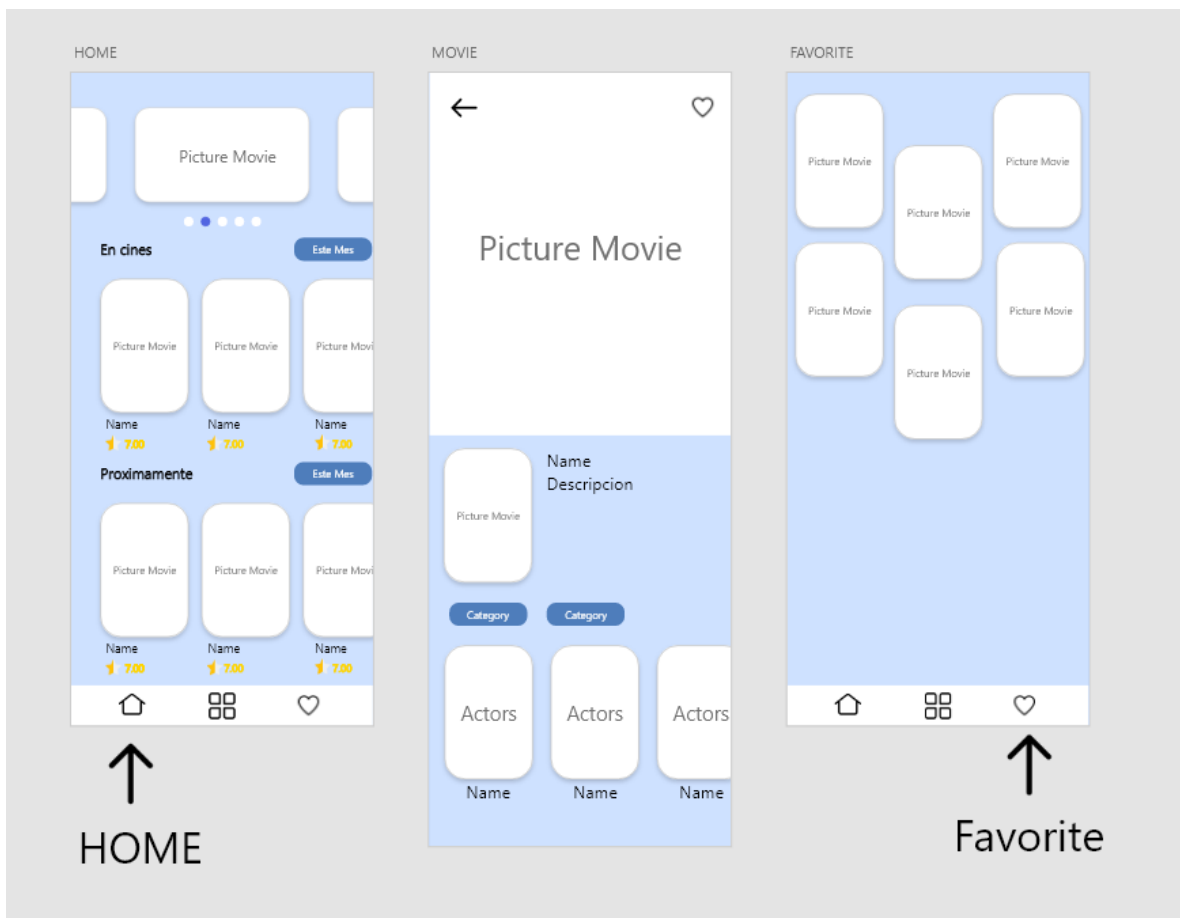


Imagen 15: Diseño de la Aplicación Flutter

Como se observa en la imagen anterior se tiene un diseño de tres pantallas para la aplicación móvil en las cuales destacan:

1. Pantalla HOME:

- En la parte superior, hay un carrusel de imágenes que muestra varias películas.
- Debajo, hay secciones etiquetadas como "En cines" y "Próximamente", que contienen tarjetas de películas con una imagen

y el nombre de la película. Al lado del nombre de la película, se muestra la puntuación de la película.

- En la parte inferior, hay una barra de navegación con tres íconos: uno de Home (seleccionado), el otro icono es representado para las categorías (como un apartado que puede existir en la app) y por último un corazón el cual representa las películas favoritas guardadas.

2. Pantalla MOVIE:

- Esta pantalla muestra detalles de una película en específico.
- En la parte superior izquierda, hay un ícono de flecha para regresar.
- En la parte superior derecha, hay un icono para agregar la película a favoritos.
- En el centro, aparece el título "Picture Movie" que tendrá la imagen de la película y debajo, el nombre y una descripción de la película.
- Más abajo, hay etiquetas de la categoría a la que pertenece la película, y una sección que lista actores junto con sus nombres e imágenes.

3. Pantalla FAVORITE:

- Muestra una lista de películas favoritas con una cuadrícula de tarjetas que tendrán la imagen de la película.
- La barra de navegación en la parte inferior es la misma que la de la pantalla "HOME", pero aquí el ícono del corazón está seleccionado.

Diagrama de componentes

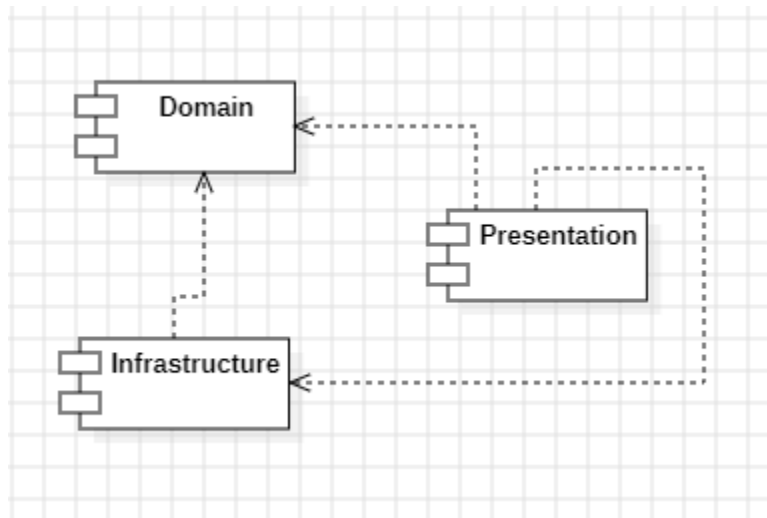


Imagen 16: Diagrama de Componentes de la Arquitectura del Proyecto en Flutter

En la imagen anterior se visualiza como se organiza el proyecto en tres capas principales: Domain, Infrastructure y Presentation. Cada una tiene subcarpetas especializadas que cumplen roles específicos en la obtención, procesamiento y visualización de los datos de películas, actores y favoritos.

1. Domain: Es la capa de lógica de negocio, donde se definen las reglas y estructuras fundamentales de la aplicación.
2. Infrastructure: Proporciona la implementación concreta para la obtención y almacenamiento de datos, además de la comunicación con APIs externas.
3. Presentation: Es la capa responsable de la interfaz de usuario, donde se organizan las vistas y se gestionan las interacciones con el usuario.

Interacción entre Capas

Cada capa interactúa de manera que se mantenga la separación de responsabilidades:

- La capa de Presentation depende de la Domain para recibir y mostrar la información procesada al usuario.
- La Domain se comunica con la Infrastructure para obtener y almacenar datos, sin preocuparse por los detalles de implementación.

- Infrastructure implementa los detalles específicos de cómo obtener y almacenar datos, asegurando que la lógica de negocio en Domain permanezca independiente y reutilizable.

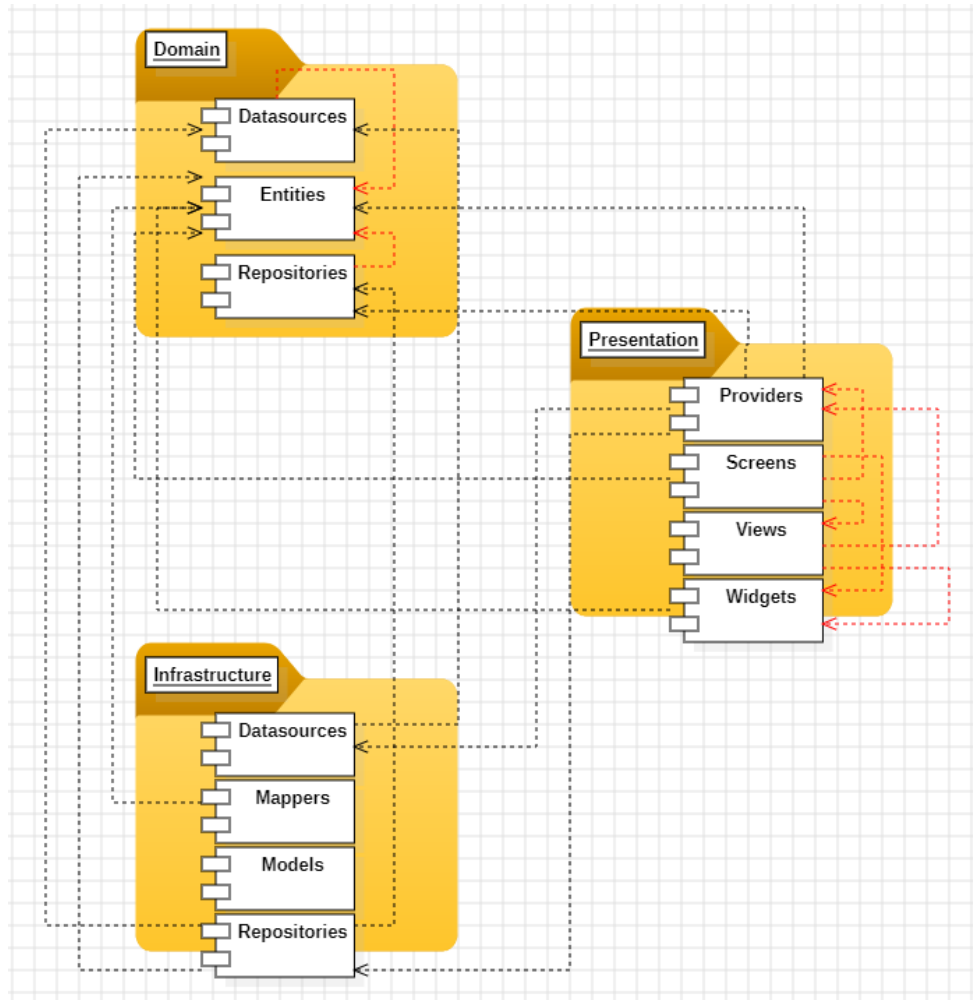


Imagen 17: Estructura Interna de las Capas de la Arquitectura del proyecto en Flutter

En la imagen anterior se visualiza como está conformada cada capa del proyecto y sus relaciones entre sí para el manejo de datos, la cual sigue un enfoque de Clean Architecture.

Relaciones entre las capas:

1. Relación entre Domain e Infrastructure

- Domain/datasources → Infrastructure/datasources: En Domain/datasources, se definen las clases abstractas necesarias para

obtener películas, actores y favoritos, actuando como contratos para la obtención de estos datos. En Infrastructure/datasources, se implementa la lógica específica para realizar llamadas a la API o gestionar la base de datos, cumpliendo así con los contratos de Domain sin que este dependa de los detalles de implementación.

- b. Domain/repositories → Infrastructure/repositories: Los repositorios en Domain definen las operaciones abstractas para obtener y almacenar datos, permitiendo el acceso a películas, actores y favoritos de manera desacoplada. En Infrastructure/repositories, se implementan estas operaciones, conectándose con los datasources específicos de API y base de datos para obtener y guardar datos, de modo que Domain accede a estos repositorios sin conocer sus detalles concretos.
- c. Domain/entities → Infrastructure/models y mappers: Las entidades en Domain definen cómo se estructuran los datos en la aplicación. Dado que los datos de la API llegan en formato JSON, Infrastructure/models representa estos datos tal cual se obtienen de la API. Luego, Infrastructure/mappers transforma estos modelos en las entidades esperadas por Domain, permitiendo un formato uniforme de datos en toda la aplicación.

2. Relación entre Domain y Presentation

- a. Domain/repositories → Presentation/providers: Los repositorios en Domain definen las operaciones necesarias para acceder a los datos de la aplicación. En Presentation/providers, se configuran estos repositorios para que las vistas y pantallas de la aplicación accedan a los datos sin conocer sus detalles específicos, delegando la obtención y gestión de datos a los repositorios de Domain.
- b. Domain/entities → Presentation/widgets: Las entidades en Domain representan datos de forma estructurada y unificada. En Presentation/widgets, estas entidades se utilizan para crear widgets personalizados que visualicen la información (como películas y

actores) en las pantallas, asegurando que la información se muestre de manera consistente en toda la aplicación.

3. Relación entre Infrastructure y Presentation

- a. Infrastructure/repositories → Presentation/providers: Infrastructure implementa los repositorios necesarios para gestionar la obtención y almacenamiento de datos desde fuentes concretas como APIs o bases de datos. Presentation/providers utiliza estos repositorios para obtener los datos y pasarlos a las vistas y pantallas, sin que Presentation dependa de los detalles técnicos de Infrastructure.
- b. Infrastructure/models y mappers → Presentation/views y widgets: Los modelos en Infrastructure contienen los datos en el formato original de la API. Los mappers convierten estos datos en entidades listas para ser usadas en Presentation/views y widgets. Así, Presentation puede acceder a datos en el formato adecuado para mostrarse en las interfaces de usuario, sin preocuparse por las conversiones y adaptaciones de datos.

Diagrama de Objetos

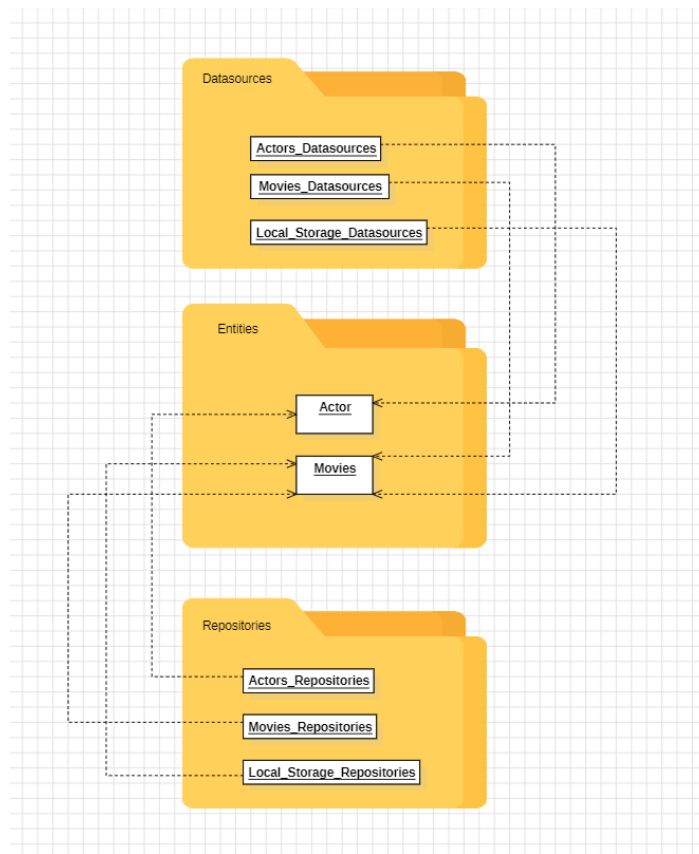


Imagen 18: Diagrama de Objetos de Domain

Domain

1. `/datasources`: Define clases abstractas que representan fuentes de datos para obtener películas, actores y favoritos, pero sin implementar la lógica específica.
2. `/entities`: Contiene las representaciones de entidades, como películas y actores, que definen sus características principales.
3. `/repositories`: Declara interfaces abstractas que describen las operaciones para acceder a películas, actores y favoritos, delegando la implementación a la capa Infrastructure.

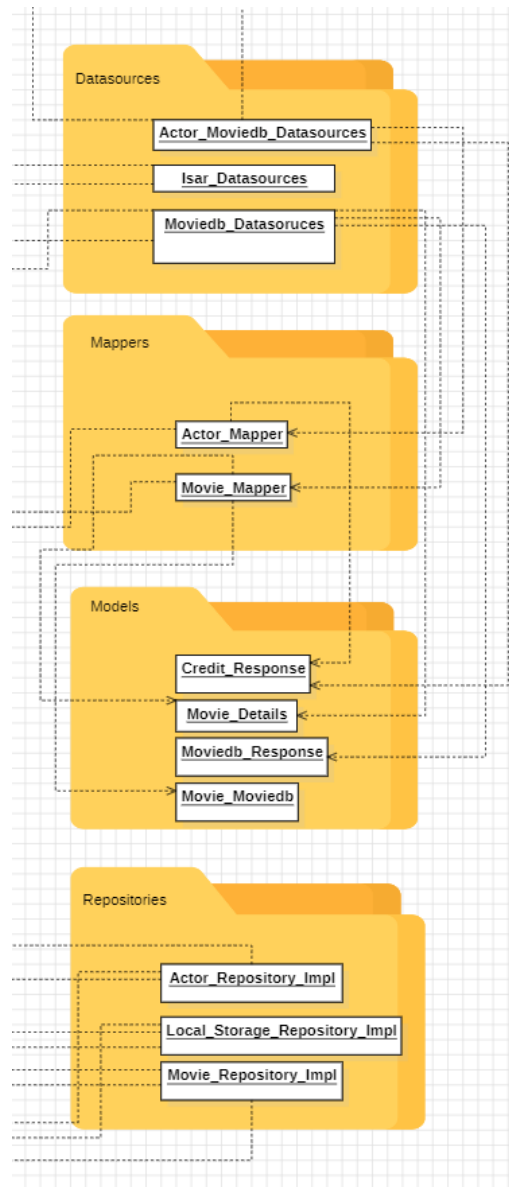


Imagen 19: Diagrama de Objetos Infraestructura

Infrastructure

1. `/datasources`: Implementa la lógica concreta para realizar llamadas al API correspondiente y gestionar la base de datos para almacenar favoritos.
2. `/mappers`: Contiene la lógica que convierte los objetos obtenidos de la API en instancias de clases específicas de la aplicación, como películas y actores, para que puedan mostrarse correctamente.

3. /models: Define los modelos de datos que corresponden al JSON de la API para películas y actores, asegurando que los datos puedan interpretarse y procesarse de manera estructurada.
4. /repositories: Implementa las interfaces declaradas en Domain, permitiendo la obtención de películas y actores desde las fuentes de datos y el almacenamiento de favoritos.

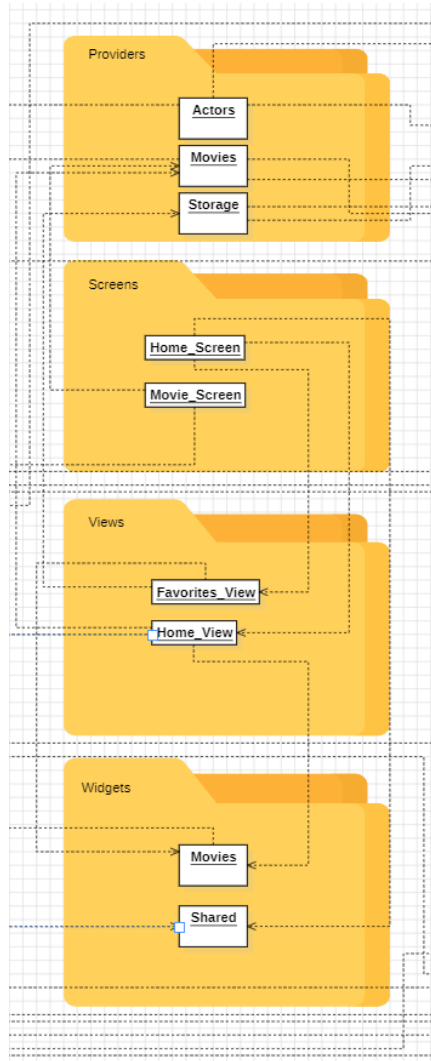


Imagen 20: Diagrama de Objetos de Presentation

Presentation

1. /providers: Gestiona la inyección de dependencias para los servicios de películas, actores y almacenamiento local de favoritos.

2. /screens: Contiene el diseño de las pantallas principales de la aplicación, como la pantalla de inicio y la de favoritos.
3. /views: Aglutina las vistas principales de la aplicación, como las secciones de inicio y favoritos.
4. /widgets: Contiene widgets personalizados, como los que muestran la información de películas y actores en las pantallas.

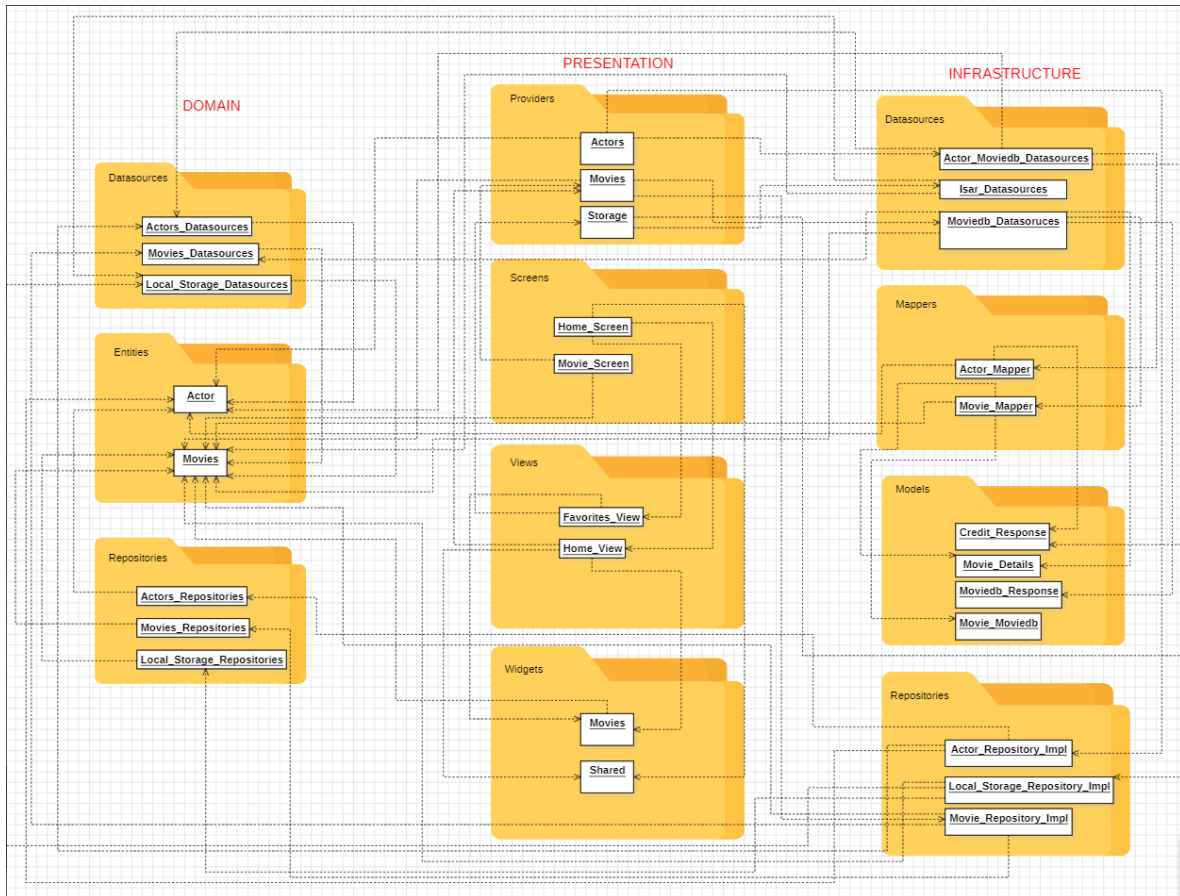


Imagen 21: Diagrama de Objetos General

Relación entre las capas profundizada:

1. Relación entre Domain e Infrastructure
 - a. Domain/datasources → Infrastructure/datasources: Los objetos Actors_Datasources, Movies_Datasources y Local_Storage_Datasources en Domain definen interfaces y contratos abstractos para la obtención de datos de actores, películas y

almacenamiento local. Estas interfaces son implementadas en Infrastructure por Actor_MovieDb_Datasources, Isar_Datasources y MovieDb_Datasources, que contienen la lógica concreta para obtener datos de la API de TheMovieDB y gestionar la base de datos local.

- b. Domain/repositories → Infrastructure/repositories: Los objetos Actors_Repositories, Movies_Repositories y Local_Storage_Repositories en Domain definen las operaciones abstractas para la interacción con fuentes de datos. Infrastructure/repositories implementa estas abstracciones con carpetas como Actor_Repository_Impl, Movie_Repository_Impl y Local_Storage_Repository_Impl, que utilizan los datasources de Infrastructure para interactuar con las APIs y bases de datos de manera concreta.
- c. Domain/entities → Infrastructure/models y mappers: Los objetos Actor y Movies en Domain representan los datos de manera estructurada y estándar. En Infrastructure/models, se representan datos en el formato recibido de la API (por ejemplo, MovieDb_Response, Movie_Details, Credit_Response). Infrastructure/mappers convierte estos modelos en entidades de Domain usando Actor_Mapper y Movie_Mapper, facilitando que los datos pasen de un formato API a un formato manejable por Domain.

2. Relación entre Domain y Presentation

- a. Domain/repositories → Presentation/providers: Los repositorios en Domain (Actors_Repositories, Movies_Repositories, Local_Storage_Repositories) proveen una capa abstracta que permite la obtención de datos desde Presentation/providers (Actors, Movies, Storage). Presentation usa estos providers para que las pantallas y vistas puedan acceder a la lógica de negocio sin depender de detalles específicos de obtención de datos.

- b. Domain/entities → Presentation/widgets y views: Los objetos Actor y Movies se utilizan en Presentation/widgets y views para mostrar los datos en la interfaz de usuario. Por ejemplo, Movies en widgets usa la estructura de Domain para representar y renderizar los datos de las películas en la interfaz de manera coherente, y views como Favorites_View y Home_View utilizan estas entidades para presentar la información al usuario.

3. Relación entre Infrastructure y Presentation

- a. Infrastructure/repositories → Presentation/providers: Los repositorios concretos en Infrastructure (Actor_Repository_Impl, Movie_Repository_Impl, Local_Storage_Repository_Impl) son utilizados por los providers de Presentation (Actors, Movies, Storage) para obtener y manejar los datos necesarios para las pantallas y vistas.
- b. Infrastructure/models y mappers → Presentation/screens y views: Los modelos en Infrastructure (MovieDb_Response, Credit_Response, etc.) son convertidos a entidades de Domain usando los mappers. Presentation/screens y views (como Home_Screen, Movie_Screen, Favorites_View) usan estos datos ya transformados para presentar información coherente al usuario.

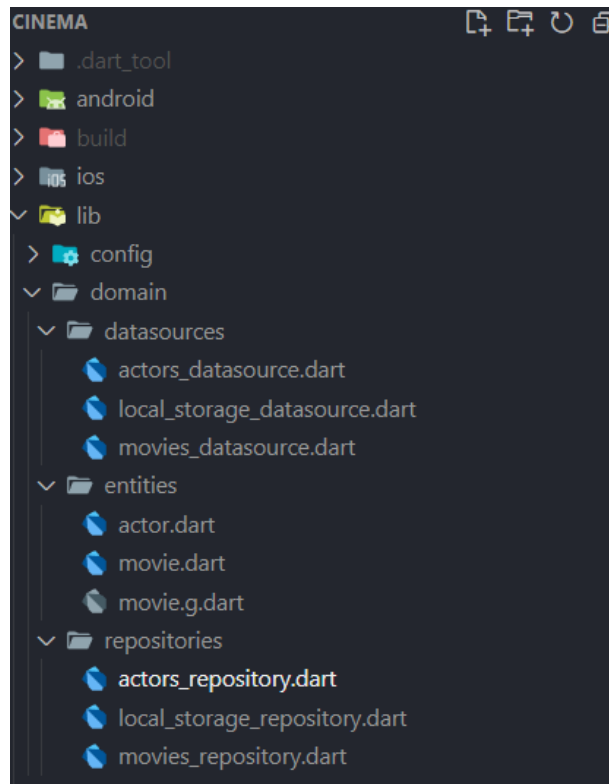


Imagen 22: Imagen visual de la estructura del proyecto domain

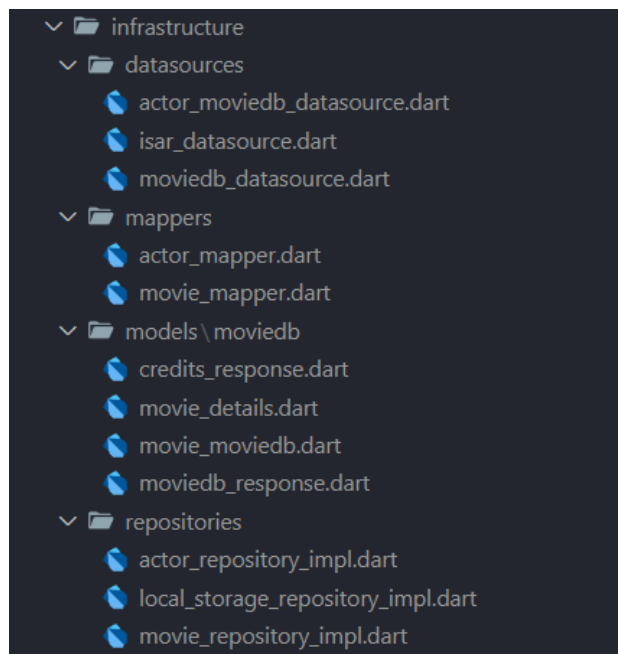


Imagen 23: Imagen visual de la estructura del proyecto infrastructure

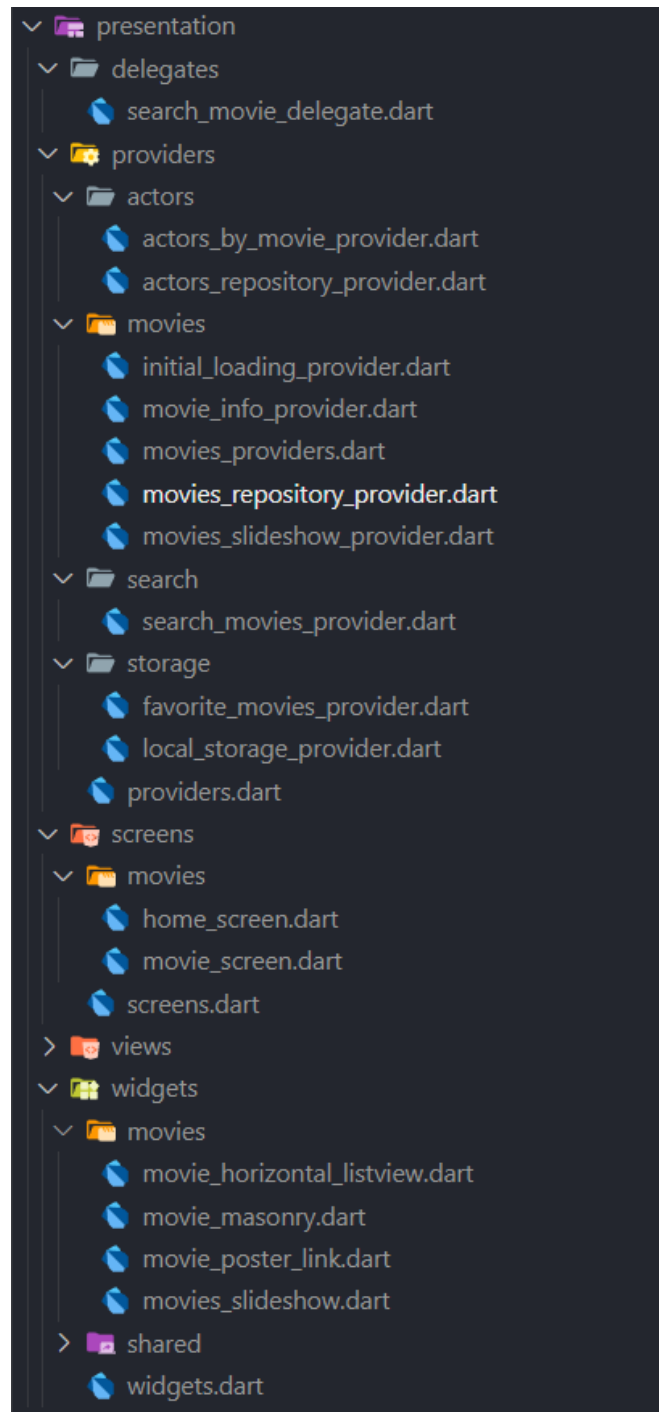


Imagen 24: Imagen visual de la estructura del proyecto presentation

Implementación

Configuración del entorno

Dado que el framework de Flutter permite el desarrollo de aplicaciones multiplataforma, no está restringido a un sistema operativo en particular como lo es el desarrollar aplicaciones móviles para iOS.

Para este proyecto se utilizó un equipo MacBook Pro con macOS Ventura como sistema operativo, por lo que facilita el acceso tanto a las herramientas de desarrollo para dispositivos Android como iOS. La elección de macOS también permite aprovechar el entorno de desarrollo nativo de iOS para realizar pruebas y simulaciones en ese sistema operativo, garantizando una cobertura completa de la aplicación en ambas plataformas.

Flutter permite trabajar en varios entornos de desarrollo como Android Studio y Visual Studio Code, ambos de los cuales ofrecen extensiones que simplifican la escritura, ejecución y depuración de código. En este proyecto, se optará por el uso de Visual Studio Code como IDE, ya que proporciona una interfaz ligera, pero potente, que se ajusta a las necesidades del desarrollo multiplataforma.

Requisitos de Hardware

Los requisitos mínimos para trabajar con el framework de Flutter en macOS son los siguientes:

- Al menos 8 GB de memoria RAM.
- 44 GB de almacenamiento disponible.
- Sistema operativo macOS 10.15.

Sin embargo, en este estudio se trabajará con un equipo con las siguientes características:

- 16 GB de memoria RAM.
- 300 GB de almacenamiento disponible.
- Sistema macOS 14.7
- Procesador Intel Core i7 3.1 GHz.

A continuación, se muestra los pasos para una instalación correcta de las herramientas necesarias para comenzar a desarrollar la aplicación móvil en Flutter:

Instalación de Flutter

Paso 1: Descargar el Flutter SDK

- Visitar la pagina oficial de Flutter: <https://docs.flutter.dev/get-started/install/macos/mobile-ios> y descargar el SDK para macOS.
- Extraer el archivo .zip en una carpeta sin permisos de administrador (en nuestro caso será “~/development/flutter”).

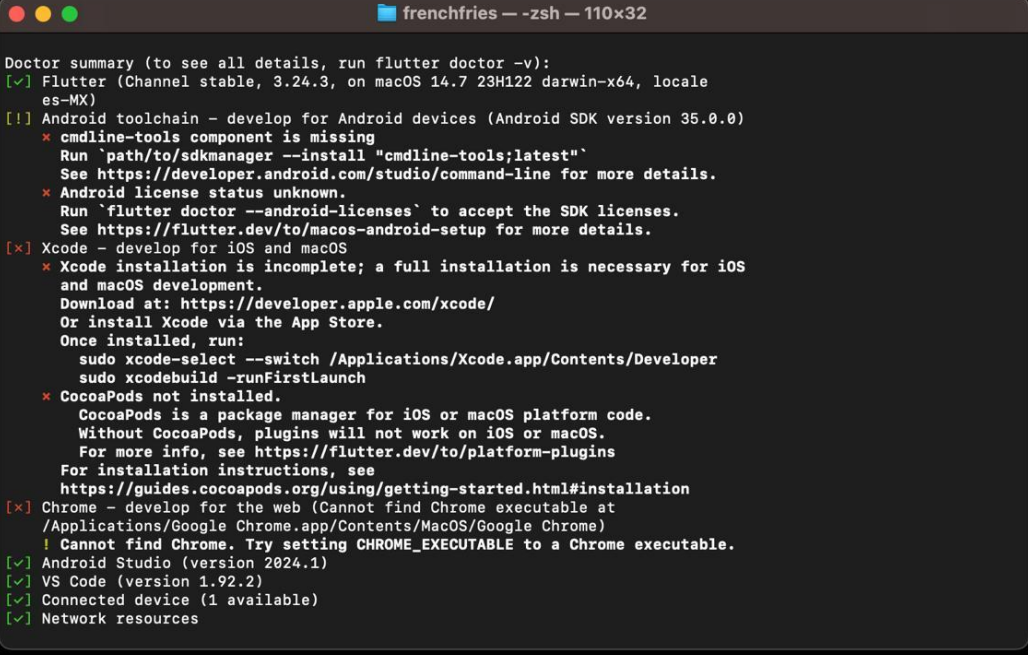
Paso 2: Configurar la variable de entorno PATH

- Abrimos el archivo .zshrc que se encuentra en “/Users/NameUser/.zshrc”.
- Agregáramos el PATH “export PATH=/Users/NameUser/development/flutter/bin:\$PATH”.
- Guardamos los cambios del archivo .zshrc.

Paso 3: Verificar la instalación

Para determinar que la instalación se hizo correctamente lo siguiente será correr el comando **flutter doctor** en la Terminal, ya que este comando proporciona una lista de las herramientas necesarias y su estado.

Por lo que el comando nos debería mostrar algo similar a la siguiente figura:



```
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.24.3, on macOS 14.7 23H122 darwin-x64, locale
es-MX)
[!] Android toolchain - develop for Android devices (Android SDK version 35.0.0)
    × cmdline-tools component is missing
      Run `path/to/sdkmanager --install "cmdline-tools;latest"`
      See https://developer.android.com/studio/command-line for more details.
    × Android license status unknown.
      Run `flutter doctor --android-licenses` to accept the SDK licenses.
      See https://flutter.dev/to/macos-android-setup for more details.
[×] Xcode - develop for iOS and macOS
    × Xcode installation is incomplete; a full installation is necessary for iOS
      and macOS development.
      Download at: https://developer.apple.com/xcode/
      Or install Xcode via the App Store.
      Once installed, run:
        sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
        sudo xcodebuild -runFirstLaunch
    × CocoaPods not installed.
      CocoaPods is a package manager for iOS or macOS platform code.
      Without CocoaPods, plugins will not work on iOS or macOS.
      For more info, see https://flutter.dev/to/platform-plugins
      For installation instructions, see
        https://guides.cocoapods.org/using/getting-started.html#installation
[×] Chrome - develop for the web (Cannot find Chrome executable at
/Applications/Google Chrome.app/Contents/MacOS/Google Chrome)
    ! Cannot find Chrome. Try setting CHROME_EXECUTABLE to a Chrome executable.
[✓] Android Studio (version 2024.1)
[✓] VS Code (version 1.92.2)
[✓] Connected device (1 available)
[✓] Network resources
```

Imagen 25: Salida de Flutter Doctor para Diagnóstico del Entorno de Desarrollo

Instalar herramientas adicionales

1. Xcode (para desarrollo en iOS)

- Si no tienes Xcode instalado, puedes instalarlo desde la App Store.
- Una vez instalado, abre Xcode y acepta el Acuerdo de Licencia.
- Instala las herramientas de línea de comandos:

“sudo xcode-select --install”.

1. Android Studio (para desarrollo en Android)

- Descarga e instalar el IDE Android Studio desde:
https://developer.android.com/studio?_gl=1*10nfeiy*_up*MQ..&gclid=Cj0KCQjwjY64BhCaARIsAlfc7YZ_AliN-sp_8U_RWTuCRz26oXAGm9E9mGhsE6S3B1h5m9hMqD7xTu4aAlc8EALw_wcB&gclsrc=aw.ds.

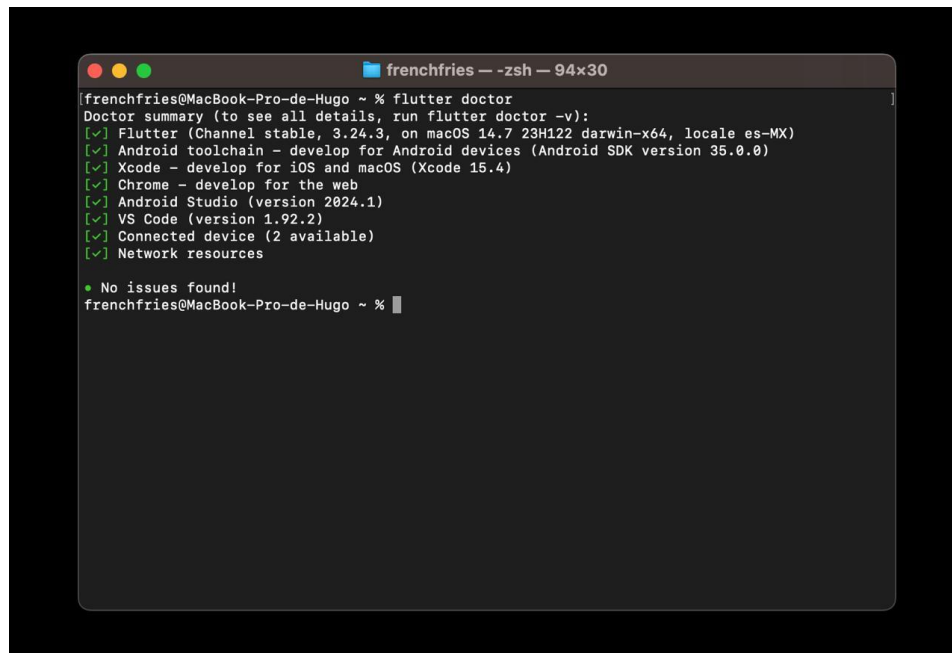
- Durante la instalación, asegúrate de incluir las herramientas de línea de comandos y el SDK de Android además de resolver los problemas que “flutter doctor” hace hincapié.
- Abre Android Studio y sigue el asistente de configuración inicial para instalar los componentes recomendados.

2. Visual Studio

- Descargar e instalar el IDE Visual Studio desde su página oficial en <https://code.visualstudio.com/download>.

3. Comprobaciones finales

Una vez terminado con todas las instalaciones adicionales volvemos a ejecutar el comando “**flutter doctor**” para asegurarnos de que tanto Xcode como Android Studio estan correctamente configurados.

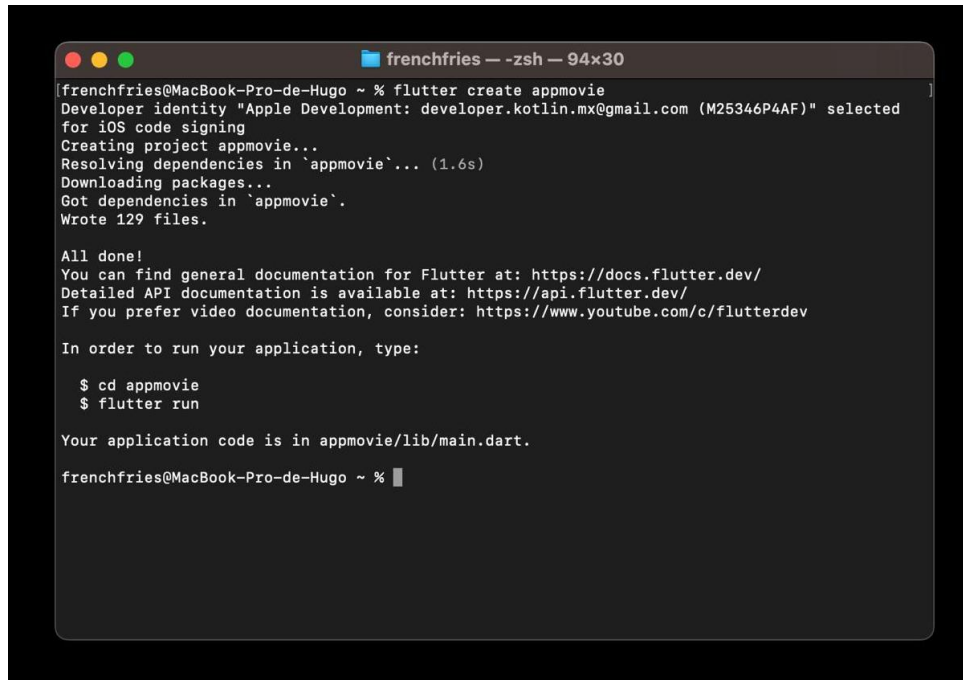


```
frenchfries@MacBook-Pro-de-Hugo ~ % flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.24.3, on macOS 14.7 23H122 darwin-x64, locale es-MX)
[✓] Android toolchain - develop for Android devices (Android SDK version 35.0.0)
[✓] Xcode - develop for iOS and macOS (Xcode 15.4)
[✓] Chrome - develop for the web
[✓] Android Studio (version 2024.1)
[✓] VS Code (version 1.92.2)
[✓] Connected device (2 available)
[✓] Network resources

• No issues found!
frenchfries@MacBook-Pro-de-Hugo ~ %
```

Imagen 26: Resultado de Flutter Doctor: Entorno Configurado Correctamente

Tras terminar con la instalación de todas las herramientas de Flutter, es necesario proceder con la creación del proyecto de la aplicación utilizando el siguiente comando en la terminal **flutter craate [NombredelaApp]**.

A terminal window titled "frenchfries -- -zsh -- 94x30" showing the execution of the command "flutter create appmovie". The output indicates that the developer identity "Apple Development: developer.kotlin.mx@gmail.com (M25346P4AF)" was selected for iOS code signing. The project "appmovie" was created, dependencies were resolved (1.6s), and 129 files were written. The terminal also provides links to Flutter documentation and instructions on how to run the application.

```
frenchfries@MacBook-Pro-de-Hugo ~ % flutter create appmovie
Developer identity "Apple Development: developer.kotlin.mx@gmail.com (M25346P4AF)" selected
for iOS code signing
Creating project appmovie...
Resolving dependencies in `appmovie`... (1.6s)
Downloading packages...
Got dependencies in `appmovie`.
Wrote 129 files.

All done!
You can find general documentation for Flutter at: https://docs.flutter.dev/
Detailed API documentation is available at: https://api.flutter.dev/
If you prefer video documentation, consider: https://www.youtube.com/c/flutterdev

In order to run your application, type:

$ cd appmovie
$ flutter run

Your application code is in appmovie/lib/main.dart.

frenchfries@MacBook-Pro-de-Hugo ~ %
```

Imagen 27: Comando para crear un proyecto en Flutter

TheMovieDB API

La aplicación que desarrollaremos para medir su rendimiento será una app de para ver las películas más recientes, populares, ver sus actores, descripción de la película y agregar a favoritos las películas deseadas y para ello es necesario el consumo de una API de películas, para este caso utilizaremos la API de TheMovieDB ya que es una API publica, lo único que debemos hacer es crear una cuenta, y para hacer esto es tan fácil como ingresar a <https://www.themoviedb.org/signup>

Crea una cuenta

Crear una cuenta es fácil y gratis. Rellena el formulario para empezar. Se necesita JavaScript para continuar.

Usuario

Contraseña (mínimo 4 caracteres)

Repite la contraseña

Correo electrónico

Al hacer clic en el botón "Registrarse", certifico que he leído y acepto los términos de uso y la política de privacidad de TMDB.

Registrarse Cancelar

Imagen 28: Formulario de Registro de Usuario en TMDB

Una vez registrados nos dirigimos a ajustes de nuestra cuenta y seleccionamos la opción API la cual nos mostrara nuestra API a utilizar para el proyecto.

Sesiones	Clave de la API
API	7995405c88d12154d62e5eaa947b9e25

Imagen 29: Visualización de Clave de API en TMDB

NOTA: Esta API es tanto para la aplicación desarrollada con el framework de Flutter como React Native.

Módulos

Módulo de Consumo de API de TheMovieDB

El Módulo de Consumo de API de TheMovieDB es fundamental para la obtención de datos de películas y actores en nuestra aplicación. Este módulo se encarga de realizar las solicitudes HTTP necesarias para interactuar con la API de TheMovieDB, recibir los datos en formato JSON y convertirlos en modelos manejables para la aplicación. A través de este módulo, podremos acceder a información en tiempo real, como listas de películas populares, detalles de cada película y datos sobre actores.

Este módulo no solo realiza las llamadas a la API, sino que también implementa los mappers necesarios para transformar la respuesta cruda de la API en entidades que la aplicación pueda entender y utilizar en la capa de presentación. Desglosaremos su estructura en tres partes principales:

- Los datasources que ejecutan las llamadas a la API.
- Los mappers que transforman los datos de la API en entidades de la aplicación.
- Los models que representan la estructura de las respuestas JSON de la API.

Esto permitirá una comunicación fluida entre la API y nuestra aplicación, garantizando que los datos se gestionen de manera óptima y consistente.

Comenzando con los datasources tenemos el archivo llamado **moviedb_datasources.dart** el cual en su primera parte encontraremos lo siguiente:

```

class MoviedbDatasource extends MoviesDatasource {

  final dio = Dio(BaseOptions(
    baseUrl: 'https://api.themoviedb.org/3',
    queryParameters: {
      'api_key': Environment.theMovieDbKey,
      'language': 'es-MX'
    }
  )); // BaseOptions // Dio

  List<Movie> _jsonToMovies( Map<String,dynamic> json ) {

    final movieDBResponse = MovieDbResponse.fromJson(json);

    final List<Movie> movies = movieDBResponse.results
      .where((moviedb) => moviedb.posterPath != 'no-poster' )
      .map(
        (moviedb) => MovieMapper.movieDBToEntity(moviedb)
      ).toList();

    return movies;
  }
}

```

Imagen 30: Implementación de un MovieDatasource para consumir TheMovieDB API con Dio en Flutter

Como se puede observar en la imagen la primera parte de este archivo consta de una clase la cual extiende de MoviesDatasource la cual está diseñada para facilitar el uso de obtención de la información de las películas.

Dentro de **moviedb_datasources** se define un cliente “**dio**” el cual es una librería para realizar peticiones HTTP. La configuración incluye:

1. **baseUrl:** Especifica la URL base de la API de TheMovieDB, en este caso 'https://api.themoviedb.org/3'.
2. **queryParameters:** Configura los parámetros predeterminados para cada solicitud:
 - a. **api_key:** Se obtiene del entorno mediante Environment.theMovieDbKey para autenticar la solicitud.
 - b. **language:** Define el idioma de respuesta, que está configurado en 'es-MX' para obtener resultados en español de México.

Por último, tenemos el método **_jsonToMovies** que convierte la respuesta JSON de la API en una lista de objetos de tipo “movie”, dentro de esta función tenemos:

1. **Parámetro json:** Recibe el Map<String, dynamic> json, que es la respuesta JSON de la API de TheMovieDB.
2. **Conversión del JSON a MovieDbResponse:** Primero, el JSON se convierte a una instancia de MovieDbResponse mediante el método fromJson. Esto permite manejar la respuesta con una estructura más organizada.
3. **Filtrado de resultados:** Se filtran los resultados para incluir solo aquellas películas que tienen un posterPath válido (distinto de 'no-poster'), asegurando que solo las películas con póster se incluyan en el listado.
4. **Mapeo de resultados:** Cada resultado de movieDbResponse.results se transforma en una entidad Movie mediante el método MovieMapper.movieDBToEntity. Esto es útil si necesitas transformar el modelo de la API a un modelo de dominio que sea más adecuado para tu aplicación.
5. **Retorno de la lista de películas:** Finalmente, la función retorna la lista de películas (List<Movie>), que puede ser utilizada en otras partes de la aplicación para mostrar o manipular los datos de las películas obtenidas.

```

@override
Future<List<Movie>> getNowPlaying({int page = 1}) async {

  final response = await dio.get('/movie/now_playing',
    |   queryParameters: {
    |     'page': page
    |   }
  );

  return _jsonToMovies(response.data);
}

@override
Future<List<Movie>> getPopular({int page = 1}) async {

  final response = await dio.get('/movie/popular',
    |   queryParameters: {
    |     'page': page
    |   }
  );

  return _jsonToMovies(response.data);
}

```

Imagen 31: Métodos de MoviesDbDatasource para Obtener Películas Populares y en Cartelera

Como se puede observar en la imagen, la segunda parte de este mismo archivo llamado **moviedb_datasource** se manejan métodos con la principal función de obtener los datos en una lista desde la API de The MovieDB, el cual está estructurada de la siguiente forma:

1. Parámetro page
 - a. Este parámetro es opcional y tiene un valor predeterminado de 1, lo que permite manejar la paginación de los resultados.
 - b. Se pasa como un parámetro de consulta (queryParameters) en la solicitud HTTP, permitiendo cargar diferentes páginas de resultados de la API.
2. Llamada a la API

- a. Utiliza el cliente dio para realizar una solicitud GET a la ruta '/movie/now_playing' de TheMovieDB.
 - b. Esta solicitud incluye el parámetro de paginación en queryParameters, lo que permite a la API devolver la página específica de resultados solicitada.
3. Transformación de Datos
- a. Una vez que se recibe la respuesta, el método getNowPlaying llama a _jsonToMovies y le pasa los datos (response.data).
 - b. _jsonToMovies convierte la respuesta JSON en una lista de objetos Movie, filtrando y mapeando los datos de la API a las entidades Movie de la aplicación.
4. Retorno de Resultados
- a. El método retorna una Future<List<Movie>>, una lista de películas, que será completada cuando la solicitud a la API termine y se hayan procesado los datos.

El siguiente elemento importante para el funcionamiento del módulo del consumo de la API de TheMovieDB son nuestros mappers, el primer mapper es el archivo de **movie_mapper.dart** el cual consta de lo siguiente:


```

class MovieMapper {
  static Movie movieDBToEntity(MovieMovieDB moviedb) => Movie(
    adult: moviedb.adult,
    backdropPath: (moviedb.backdropPath != '')
      ? 'https://image.tmdb.org/t/p/w500${ moviedb.backdropPath }'
      : 'https://sd.keepcalms.com/i-w600/keep-calm-poster-not-found.jpg',
    genreIds: moviedb.genreIds.map((e) => e.toString()).toList(),
    id: moviedb.id,
    originalLanguage: moviedb.originalLanguage,
    originalTitle: moviedb.originalTitle,
    overview: moviedb.overview,
    popularity: moviedb.popularity,
    posterPath: (moviedb.posterPath != '')
      ? 'https://image.tmdb.org/t/p/w500${ moviedb.posterPath }'
      : 'https://www.movienewz.com/img/films/poster-holder.jpg',
    releaseDate: moviedb.releaseDate != null ? moviedb.releaseDate! : DateTime.now(),
    title: moviedb.title,
    video: moviedb.video,
    voteAverage: moviedb.voteAverage,
    voteCount: moviedb.voteCount
  ); // Movie

  static Movie movieDetailsToEntity( MovieDetails moviedb ) => Movie(
    adult: moviedb.adult,
    backdropPath: (moviedb.backdropPath != '')
      ? 'https://image.tmdb.org/t/p/w500${ moviedb.backdropPath }'
      : 'https://sd.keepcalms.com/i-w600/keep-calm-poster-not-found.jpg',
    genreIds: moviedb.genres.map((e) => e.name ).toList(),
    id: moviedb.id,
    originalLanguage: moviedb.originalLanguage,
    originalTitle: moviedb.originalTitle,
    overview: moviedb.overview,
    popularity: moviedb.popularity,
    posterPath: (moviedb.posterPath != '')
      ? 'https://image.tmdb.org/t/p/w500${ moviedb.posterPath }'
      : 'https://sd.keepcalms.com/i-w600/keep-calm-poster-not-found.jpg',
    releaseDate: moviedb.releaseDate,
    title: moviedb.title,
    video: moviedb.video,
    voteAverage: moviedb.voteAverage,
    voteCount: moviedb.voteCount
  );
}

```

Imagen 32: Mapper para transformar objetos de TheMovieDB a entidades de dominio en Flutter

Este archivo se encarga de mapear los datos obtenidos desde la API de TheMovieDB a la entidad Movie el cual se encuentra dentro del proyecto, además este mapeo es crucial para convertir los datos provenientes de **MovieMovieDB** y

MovieDetails a una estructura de dominio Movie, este archivo está estructurado de la siguiente manera:

Como principal tenemos la clase la cual maneja 2 métodos estáticos que facilitan la transformación de datos desde modelos de infraestructura a entidades de dominio

El método **movieDBToEntity** transforma un objeto de tipo **MovieMovieDB** (un modelo que representa los datos de una película tal como los recibe de la API de TheMovieDB) en una instancia de la entidad Movie.

El método de **movieDetailsToEntity** transforma un objeto MovieDetails (otro modelo que representa detalles específicos de una película desde la API de TheMovieDB) en una instancia de la entidad Movie. La estructura es similar a movieDBToEntity, pero adapta algunos campos específicos a partir de MovieDetails.

El segundo archivo de nuestros mappers es el de **actor_mapper.dart** el cual contiene lo siguiente:

```
class ActorMapper {
  static Actor castToEntity( Cast cast ) =>
    Actor(
      id: cast.id,
      name: cast.name,
      profilePath: cast.profilePath != null
        ? 'https://image.tmdb.org/t/p/w500${ cast.profilePath }'
        : 'https://st3.depositphotos.com/4111759/13425/v/600/depositphotos_134259',
      character: cast.character,
    );
}
```

Imagen 33: Mapper para convertir un objeto Cast a una entidad Actor en Flutter

Este archivo tiene como objetivo mapear la información de un actor proveniente de la API, en una instancia de la entidad "Actor". Al igual que en el caso de MovieMapper, este mapeo permite transformar un modelo de infraestructura (Cast)

en una entidad de dominio (Actor). Esto es importante para tener datos consistentes y listos para ser utilizados en la lógica de la aplicación.

Por último, tenemos los modelos que representan la estructura de las respuestas del cual se cuenta con 2 archivos el primero es **moviedb_response.dart** el cual contiene lo siguiente:

```
class MovieDbResponse {
  MovieDbResponse({
    required this.dates,
    required this.page,
    required this.results,
    required this.totalPages,
    required this.totalResults,
  });

  final Dates? dates;
  final int page;
  final List<MovieMovieDB> results;
  final int totalPages;
  final int totalResults;

  factory MovieDbResponse.fromJson(Map<String, dynamic> json) =>
    MovieDbResponse(
      dates: json["dates"] != null ? Dates.fromJson(json["dates"]) : null,
      page: json["page"],
      results: List<MovieMovieDB>.from(
        json["results"].map((x) => MovieMovieDB.fromJson(x)), // List.from
      totalPages: json["total_pages"],
      totalResults: json["total_results"],
    ); // MovieDbResponse

  Map<String, dynamic> toJson() => {
    "dates": dates?.toJson(),
    "page": page,
    "results": List<dynamic>.from(results.map((x) => x.toJson())),
    "total_pages": totalPages,
    "total_results": totalResults,
  };
}
```

Imagen 34: Modelo MovieDbResponse para manejar datos de la API de TheMovieDB en Flutter

La clase de **MovieDbResponse** representa la respuesta JSON que se recibe de la API de TheMovieDB cuando se hace una solicitud para obtener una lista de películas, esta clase ayuda a mapear y estructurar los datos de la respuesta para facilitar su uso en la aplicación.

El constructor de **MovieDbResponse** requiere que se proporcionen valores para todos los campos, excepto dates, que puede ser nulo. Esto asegura que al crear una instancia de **MovieDbResponse**, todos los datos importantes estén presentes.

El metodo de **fromJson** es un factory constructor que permite crear una instancia de MovieDbResponse a partir de un mapa Map<String, dynamic>, que representa el JSON devuelto por la API. Este método permite crear la estructura MovieDbResponse a partir de un JSON con solo una línea de código, simplificando el procesamiento de la respuesta.

El método **toJson** convierte la instancia MovieDbResponse de vuelta a un mapa Map<String, dynamic>, lo que facilita la serialización de los datos para enviarlos o almacenarlos si es necesario.

El segundo y último archivo del módulo del consumo de la API es el de **movie_details.dart** el cual contiene lo siguiente:

```

factory MovieDetails.fromJson(Map<String, dynamic> json) => MovieDetails({
  adult: json["adult"],
  backdropPath: json["backdrop_path"] ?? '',
  belongsToCollection: json["belongs_to_collection"]
    == null ? null : BelongsToCollection.fromJson(json["belongs_to_collection"]),
  budget: json["budget"],
  genres: List<Genre>.from(json["genres"].map((x) => Genre.fromJson(x))),
  homepage: json["homepage"],
  id: json["id"],
  imdbId: json["imdb_id"],
  originalLanguage: json["original_language"],
  originalTitle: json["original_title"],
  overview: json["overview"],
  popularity: json["popularity"]?.toDouble(),
  posterPath: json["poster_path"],
  productionCompanies: List<ProductionCompany>.
    from(json["production_companies"].map((x) => ProductionCompany.fromJson(x))),
  productionCountries: List<ProductionCountry>.
    from(json["production_countries"].map((x) => ProductionCountry.fromJson(x))),
  releaseDate: DateTime.parse(json["release_date"]),
  revenue: json["revenue"],
  runtime: json["runtime"],
  spokenLanguages: List<SpokenLanguage>.
    from(json["spoken_languages"].map((x) => SpokenLanguage.fromJson(x))),
  status: json["status"],
  tagline: json["tagline"],
  title: json["title"],
  video: json["video"],
  voteAverage: json["vote_average"]?.toDouble(),
  voteCount: json["vote_count"],
}); // MovieDetails

```

Imagen 35: Factory constructor para deserializar MovieDetails desde JSON en Flutter

```

Map<String, dynamic> toJson() => {
  "adult": adult,
  "backdrop_path": backdropPath,
  "belongs_to_collection": belongsToCollection?.toJson(),
  "budget": budget,
  "genres": List<dynamic>.from(genres.map((x) => x.toJson())),
  "homepage": homepage,
  "id": id,
  "imdb_id": imdbId,
  "original_language": originalLanguage,
  "original_title": originalTitle,
  "overview": overview,
  "popularity": popularity,
  "poster_path": posterPath,
  "production_companies": List<dynamic>.from(productionCompanies.map((x) => x.toJson())),
  "production_countries": List<dynamic>.from(productionCountries.map((x) => x.toJson())),
  "release_date": "${releaseDate.year.
toString().padLeft(4, '0')}-${releaseDate.month.toString().
padLeft(2, '0')}-${releaseDate.day.toString().padLeft(2, '0')}",
  "revenue": revenue,
  "runtime": runtime,
  "spoken_languages": List<dynamic>.from(spokenLanguages.map((x) => x.toJson())),
  "status": status,
  "tagline": tagline,
  "title": title,
  "video": video,
  "vote_average": voteAverage,
  "vote_count": voteCount,
};

```

Imagen 36: Método toJson para serializar un objeto MovieDetails a formato JSON en Flutter

La clase **MovieDetails** representa el conjunto completo de detalles de una película y es la clase principal del modelo. Contiene varias propiedades que describen diferentes aspectos de la película, como el título, fecha de lanzamiento, presupuesto, lista de géneros, y mucho más.

Al igual que la clase de **MovieDbResponse** se maneja el mismo método de **fromJson** y **toJson** para la respuesta de la llamada a la API, la única diferencia particularmente son los datos que se piden y que aquí se crea una instancia de **MovieDetails**.

Módulo de Favoritos con ISAR

El Módulo de Favoritos con ISAR permite a los usuarios guardar y gestionar sus películas favoritas localmente en la aplicación. A través de este módulo, los datos de favoritos se almacenan de manera persistente en el dispositivo del usuario, de modo que incluso sin conexión a internet, el usuario puede acceder a su lista de favoritos. ISAR es la base de datos elegida debido a su ligereza y eficacia en el almacenamiento local.

Este módulo cubre todas las operaciones CRUD (crear, leer, actualizar y eliminar) para los favoritos, y consta de:

- Datasources que implementan la lógica para interactuar con la base de datos ISAR, realizando operaciones de almacenamiento y recuperación de favoritos.
- Un repositorio que centraliza estas operaciones y expone métodos fáciles de usar en la capa de dominio.
- Un provider que gestiona el estado de los favoritos en la interfaz de usuario, asegurando que la lista se mantenga actualizada y reactiva ante los cambios realizados por el usuario.

Al desglosar este módulo, se busca crear una funcionalidad de favoritos sólida y accesible, que garantice una experiencia de usuario intuitiva y confiable, permitiendo que los usuarios tengan control sobre las películas que desean guardar y ver posteriormente.

Comenzando con los datasources tenemos el archivo llamado **isar_datasource.dart** el cual en su primera parte encontraremos lo siguiente:

```

class IsarDatasource extends LocalStorageDatasource {

  late Future<Isar> db;

  IsarDatasource() {
    db = openDB();
  }

  Future<Isar> openDB() async {
    if ( Isar.instanceNames.isEmpty ) {
      final dir = await getApplicationDocumentsDirectory();
      return await Isar.
        open([ MovieSchema ], inspector: true,directory: dir.path);
    }

    return Future.value(Isar.getInstance());
  }
}

```

Imagen 37: Datasource para gestionar la base de datos local con Isar en Flutter

Como se puede observar en la imagen anterior tenemos una clase llamada **IsarDatasource** el cual extiende de **LocalStorageDatasource** el cual está diseñado para manejar la persistencia de datos utilizando la base de datos local Isar. Esta clase se encarga de abrir y gestionar la conexión a la base de datos de manera asíncrona para que otros componentes de la aplicación puedan interactuar con los datos almacenados.

Algunos elementos clave son:

1. Propiedad **db**:
 - a. **Db** es una propiedad de tipo **Future<Isar>** que mantiene la instancia de la base de datos Isar.
 - b. Esta propiedad se inicializa en el constructor mediante la función **openDB()**, asegurando que la conexión a la base de datos esté lista cuando sea necesario.
2. Constructor **IsarDatasource()**:

- a. Al instanciar **IsarDataSource**, el constructor asigna **db** llamando a **openDB()**. Esto garantiza que la base de datos esté abierta y lista para operar desde el inicio de la instancia de **IsarDataSource**.
3. Metodo **openDB**:
- a. Este método es asíncrono y se encarga de abrir o obtener la instancia de la base de datos Isar.
 - b. Condición inicial: Si **Isar.instanceNames** está vacío (es decir, no hay instancias de la base de datos activas), **openDB**:
 - i. Obtiene el directorio de documentos de la aplicación mediante **getApplicationDocumentsDirectory()**.
 - ii. Abre la base de datos Isar con el esquema de **MovieSchema**, habilitando el inspector (para facilitar la depuración) y especificando el directorio de almacenamiento.
 - c. Instancia existente: Si ya existe una instancia de la base de datos, el método retorna esa instancia con **Isar.getInstance()**.

```
@override
Future<bool> isMovieFavorite(int movieId) async {
  final isar = await db;

  final Movie? isFavoriteMovie = await isar.movies
    .filter()
    .idEqualTo(movieId)
    .findFirst();

  return isFavoriteMovie != null;
}
```

Imagen 38: Método para verificar si una película es favorita en la base de datos Isar en Flutter

Como se puede observar en la imagen, la segunda parte de este mismo archivo llamado **isar_datasource.dart** tenemos un método para verificar si una película con un **movieId** está marcada como favorita en la base de datos.

De tal manera que este método está formado de la siguiente manera:

1. Obtención de la instancia de la base de datos:
 - a. Primero, obtiene la instancia de la base de datos Isar a través **de await db**, asegurando que la base de datos esté abierta y lista para usarse.
2. Búsqueda de la película:
 - a. Utiliza la colección **movies** de Isar para filtrar las películas por el **movieId** especificado.
 - b. El método **filter().idEqualTo(movieId).findFirst()** realiza una consulta en la base de datos para buscar la primera película que coincida con el **movieId**.
 - c. Esta operación devuelve un objeto **Movie** si se encuentra una coincidencia, o null si no existe una película con ese ID en la base de datos.
3. Retorno del resultado:
 - a. Finalmente, el método retorna un valor booleano: true si **isFavoriteMovie** no es null (es decir, la película existe en la base de datos, indicando que es favorita) y false si es null (la película no está en la base de datos).

El siguiente elemento importante para el funcionamiento del módulo de favoritos con Isar es la implementación del nuestro repositorio de local storage el cual el archivo de **local_storage_repository_impl.dart** se encarga de actuar como un intermediario entre la capa de dominio de la aplicación y al fuentes de datos local por lo que este mismo se ve de la siguiente manera:

```

class LocalStorageRepositoryImpl extends LocalStorageRepository {

  final LocalStorageDatasource datasource;

  LocalStorageRepositoryImpl(this.datasource);

  @override
  Future<bool> isMovieFavorite(int movieId) {
    | return datasource.isMovieFavorite(movieId);
  }

  @override
  Future<List<Movie>> loadMovies({int limit = 10, offset = 0}) {
    | return datasource.loadMovies(limit: limit, offset: offset);
  }

  @override
  Future<void> toggleFavorite(Movie movie) {
    | return datasource.toggleFavorite(movie);
  }
}

```

Imagen 39: Implementación del repositorio de almacenamiento local en Flutter

Como podemos observar este archivo maneja una clase con varios métodos con el propósito de manejar la lógica de almacenamiento local de películas, como verificar si una película es favorita, cargar películas y alternar el estado de favorito de una película.

Por lo que esta clase se conforma de los siguientes elementos:

1. Propiedad **datasource**:

- a. **datasource** es una instancia de **LocalStorageDatasource** que permite acceder a los métodos para interactuar con el almacenamiento local
- b. **LocalStorageRepositoryImpl** delega las operaciones de almacenamiento a **datasource**.

2. Constructor **LocalStorageRepositoryImpl**:

- a. Recibe un **datasource** como parámetro y lo asigna a la propiedad **datasource**.
- b. Este constructor permite inyectar cualquier clase que implemente **LocalStorageDatasource**, lo cual facilita la modularidad y el cambio de la fuente de datos si fuera necesario.

3. Métodos Sobrescritos (**@override**):

- a. Cada método en **LocalStorageRepositoryImpl** está sobrescribiendo (**@override**) un método de **LocalStorageRepository**, lo que significa que debe cumplir con el contrato de la interfaz **LocalStorageRepository**.
- b. **isMovieFavorite**:
 - i. Llama a **datasource.isMovieFavorite(movieId)** para verificar si una película con el **movieId** especificado está marcada como favorita en la base de datos local.
 - ii. Devuelve un **Future<bool>**, que indica si la película es o no favorita.
- c. **loadMovies**:
 - i. Llama a **datasource.loadMovies(limit: limit, offset: offset)** para cargar una lista de películas desde el almacenamiento local.
 - ii. Acepta parámetros opcionales **limit** (para limitar el número de películas) y **offset** (para la paginación).
 - iii. Devuelve un **Future<List<Movie>>** con la lista de películas cargadas.
- d. **toggleFavorite**:
 - i. Llama a **datasource.toggleFavorite(movie)** para alternar el estado de favorito de una película específica.
 - ii. No retorna nada (**Future<void>**), ya que su propósito es actualizar el estado de favorito en el almacenamiento local.

Por último, tenemos nuestro archivo llamado **local_storage_provider.dart** el cual utiliza un proveedor de Riverpod (un gestor de estado en flutter) que nos permitirá el acceso global y centralizado a la lógica de almacenamiento de datos.

```
import 'package:cinemapedia/infrastructure/datasources/isar_datasource.dart';
import 'package:cinemapedia/infrastructure/repositories/local_storage_repository_impl.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

final localStorageRepositoryProvider = Provider(ref) {
  return LocalStorageRepositoryImpl( IsarDatasource() );
}; // Provider
```

Imagen 40: Definición del proveedor para el repositorio de almacenamiento local con Riverpod en Flutter

Como podemos ver en la imagen tenemos un provider llamado **localStorageRepositoryProvider** usando el paquete de **flutter_riverpod**, este proveedor crea y administra una instancia de **LocalStorageRepositoryImpl** con la fuente de datos de **IsarDatasources** para el almacenamiento local.

Lo útil del **Provider** de Riverpod es que es un proveedor “perezoso” (lazy-loaded) ya que solamente se crea la instancia cuando un widget o función lo necesita, mientras tanto no.

Explicación de la función de creación de la instancia del provider:

1. La función anónima **(ref) {return LocalStorageRepositoryImpl(IsarDatasource()); }** se utiliza para crear la instancia de **LocalStorageRepositoryImpl**.
2. **IsarDatasource()**: Esta es la implementación específica de la fuente de datos que maneja el almacenamiento local utilizando la base de datos Isar. Se inyecta en **LocalStorageRepositoryImpl** como la dependencia **datasource**.

3. **LocalStorageRepositoryImpl(IsarDatasource())** permite que **LocalStorageRepositoryImpl** use **IsarDatasource** para realizar operaciones de almacenamiento, como verificar favoritos o cargar películas.

Pruebas

En este apartado se evaluó el rendimiento de la aplicación móvil desarrollada con el framework Flutter, considerando factores como la velocidad de ejecución y el consumo de recursos (CPU y RAM). La evaluación se llevó a cabo utilizando herramientas como FlutterDevTools, Android Profile y Xcode Instruments con el objetivo de obtener una visión detallada del comportamiento de la aplicación. A través de estas pruebas, se busca identificar las capacidades y limitaciones de Flutter en el desarrollo de aplicaciones móviles, permitiendo analizar cómo este framework gestiona los recursos del sistema y cuál es su impacto en el rendimiento general de la aplicación.

Velocidad de Ejecución

Como uno de los aspectos principales es medir la velocidad de ejecución desde el momento en que la aplicación es abierta hasta que los datos se cargan por completo, la mejor manera de determinar este tiempo es registrando la duración del proceso. A continuación, se presenta una tabla con los tiempos exactos de carga de la aplicación tanto para Android como para iOS:

Dispositivo	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
Android	3.45 s	2.95 s	2.30 s	3.29 s	2.70 s
iOS	3.01 s	2.83 s	2.96 s	2.72 s	2.81 s

Tabla 3: Tiempos de Velocidad de Ejecución de la App Flutter en Android e iOS

A partir de los resultados obtenidos de las pruebas de velocidad de ejecución en dispositivos Android e iOS, se pueden realizar los siguientes análisis y conclusiones:

1. Promedio de Tiempos de Ejecución:

- Android: Promedio = $(3.45 + 2.95 + 2.30 + 3.29 + 2.70) / 5 = 2.94$ segundos.
- iOS: Promedio = $(3.01 + 2.83 + 2.96 + 2.72 + 2.81) / 5 = 2.87$ segundos.

Los resultados muestran que el rendimiento promedio entre Android e iOS es similar, con una ligera ventaja para iOS, cuyo promedio de tiempo es ligeramente menor en la velocidad de ejecución de la aplicación.

2. Variabilidad:

- En Android, los tiempos de ejecución varían entre 2.30 segundos y 3.45 segundos, mostrando una mayor diferencia entre el mejor y el peor caso, lo cual es característico de dispositivos con especificaciones de gama media, como el Samsung Galaxy A52.
- En iOS, los tiempos se encuentran entre 2.72 segundos y 3.01 segundos, lo que indica una menor variabilidad y un comportamiento más consistente, característico del iPhone 11.

3. Análisis Comparativo:

- iOS tiende a mostrar un rendimiento más consistente, con menores diferencias entre los distintos tiempos de ejecución. Esto puede ser atribuido a la optimización del sistema operativo y la integración hardware-software en los dispositivos Apple.
- Android presenta tanto el mejor tiempo de ejecución general (2.30 segundos) como el peor (3.45 segundos), lo que sugiere una mayor variabilidad en las condiciones de prueba o en la gestión de recursos del sistema. Esto es común en dispositivos Android debido a la diversidad de hardware y la gestión de recursos en entornos con múltiples aplicaciones en segundo plano.

Conclusión

En conclusión, iOS parece ofrecer un rendimiento más estable y consistente en comparación con Android, que muestra una mayor variabilidad en los tiempos de ejecución. Aunque Android cuenta con una prueba que muestra el mejor tiempo general, la variación podría deberse a factores como la gestión de memoria o las diferencias en las condiciones de carga. En promedio, iOS es ligeramente más rápido, con un tiempo de ejecución promedio de 2.87 segundos, frente a los 2.94 segundos de Android. Estos resultados podrían ser indicativos de cómo cada

plataforma maneja la optimización y la gestión de recursos durante la ejecución de la aplicación.

Pruebas de Consumo de CPU y RAM

Además de las pruebas de velocidad de ejecución, se realizaron pruebas para evaluar el consumo de CPU y memoria RAM en dispositivos Android e iOS. Estas métricas son cruciales para entender cómo cada plataforma maneja los recursos del sistema, especialmente en aplicaciones de alto rendimiento.

El consumo de CPU y RAM proporciona una visión sobre la eficiencia en el uso de los recursos, afectando directamente la fluidez de la aplicación y la experiencia del usuario.

Estas pruebas fueron realizadas para comprender mejor cómo los dispositivos de gama media en cada plataforma responden a la carga de trabajo y cómo se comportan en términos de eficiencia en el uso de recursos. A continuación, se presentan los resultados de dichas pruebas, analizando la gestión de la CPU, memoria RAM para ambos dispositivos.

Android

A continuación, se mostrarán las pruebas de rendimiento realizadas a la aplicación hecha con el framework de Flutter en el sistema operativo de Android en términos de consumo de CPU y RAM.

Análisis del Reporte de Uso de CPU

Bottom Up	Call Tree	Method Table	CPU Flame Chart	Filter by tag: AppStartUp	Expand All	Collapse All
Total Time	Self Time	Method				
27.61 s (27.47%)	27.61 s (27.47%)	> FlutterView.__render\$Method\$FfiNative - (dart:ui/window.dart)				
19.99 s (19.89%)	19.99 s (19.89%)	> PlatformDispatcher._scheduleFrame - (dart:ui/platform_dispatcher.dart:793)				
12.02 s (11.96%)	12.02 s (11.96%)	> Stopwatch._now - (dart:core-patch/stopwatch_patch.dart:15)				
9.82 s (9.77%)	9.82 s (9.77%)	> _postEvent - (dart:developer-patch/developer.dart:64)				
2.99 s (2.98%)	2.99 s (2.98%)	> _EventHandler._timerMillisecondClock - (dart:io-patch/eventhandler_patch.dart:1				
2.40 s (2.39%)	2.40 s (2.39%)	> PointerMoveEvent.transformed - (package:flutter/src/gestures/events.dart:1572)				
1.31 s (1.30%)	1.31 s (1.30%)	> _GrowableList._allocateData - (dart:core-patch/growable_array.dart:360)				
890.57 ms (0.89%)	890.57 ms (0.89%)	> Uint32List.Uint32List - (dart:typed_data-patch/typed_data_patch.dart:2574)				
742.14 ms (0.74%)	742.14 ms (0.74%)	> _NativeSocket.nativeAvailable - (dart:io-patch/socket_patch.dart:1730)				
766.88 ms (0.76%)	742.14 ms (0.74%)	> ListIterator.moveNext - (dart:_internal/iterable.dart:344)				
667.92 ms (0.66%)	667.92 ms (0.66%)	> Float64List.Float64List - (dart:typed_data-patch/typed_data_patch.dart:2827)				

Imagen 41: Reporte de análisis de rendimiento de CPU con Flutter DevTools de la aplicación de Flutter en Android

La imagen anterior muestra los resultados de la herramienta Flutter DevTools, específicamente la sección de CPU Profiler. Aquí se presenta un análisis de rendimiento detallado de cada método en términos de consumo de tiempo.

Columnas de "Total Time" y "Self Time"

La columna "Total Time" indica el tiempo total que tomó la ejecución del método, incluyendo el tiempo de las funciones llamadas por dicho método. "Self Time" se refiere al tiempo exclusivamente consumido por el método en sí, sin incluir las funciones anidadas.

- **"FlutterView._render\$Method\$ffiNative" (27.47% del total):** Este método se muestra como el que más tiempo consume. Representa la mayor parte del tiempo de CPU, lo que indica que está gestionando las operaciones visuales (renderizado) de la aplicación. Al ser un método nativo (indicado por ffiNative), se ejecuta fuera del entorno de Dart, lo cual puede implicar un impacto significativo en el rendimiento.
- **"PlatformDispatcher._scheduleFrame" (19.89% del total):** Este método se encarga de programar los frames de la interfaz de usuario. Su consumo relativamente alto sugiere que la gestión de la actualización de la UI es una tarea que demanda considerablemente la CPU.
- **Otros Métodos:** Métodos como Stopwatch._now, _postEvent y _EventHandler._timerMillisecondClock representan otras operaciones internas, como la medición de tiempos y el manejo de eventos en la aplicación. Aunque su porcentaje de uso de CPU es menor que los primeros métodos, su acumulación también contribuye al consumo total de recursos.

En conclusión, el análisis del reporte de uso de CPU realizado con Flutter DevTools revela que las operaciones más demandantes para la CPU están relacionadas con el renderizado visual y la gestión de la interfaz de usuario, como lo reflejan los métodos FlutterView._render\$Method\$ffiNative y PlatformDispatcher._scheduleFrame. Estos resultados destacan la importancia de optimizar el código relacionado con la actualización de la UI y el renderizado para mejorar la eficiencia de la aplicación. Además, aunque los métodos secundarios

tienen un menor impacto individual, su contribución acumulativa al uso de CPU subraya la necesidad de una gestión eficiente de las operaciones internas para reducir el consumo de recursos y mejorar el rendimiento general de la aplicación.

Análisis del Reporte de Uso de Memoria

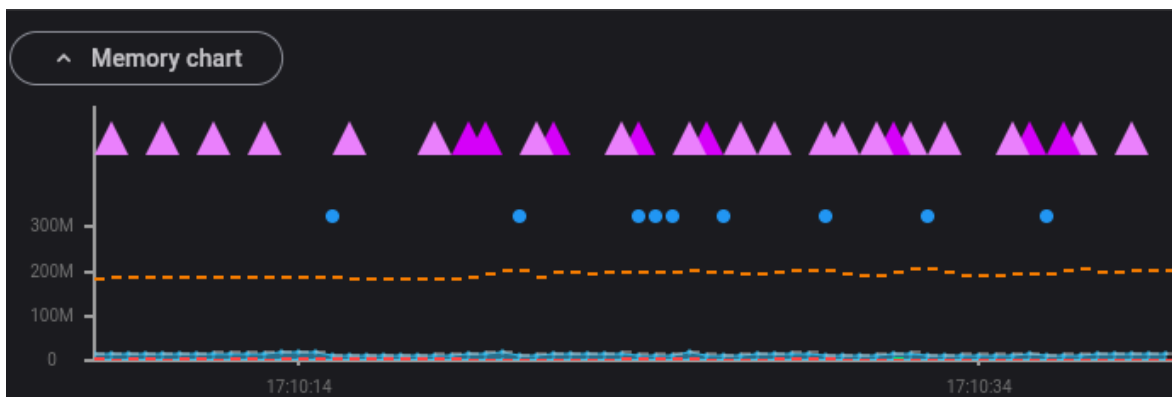


Imagen 42: Gráfica de uso de memoria en tiempo real con Flutter DevTools de la aplicación de Flutter en Android

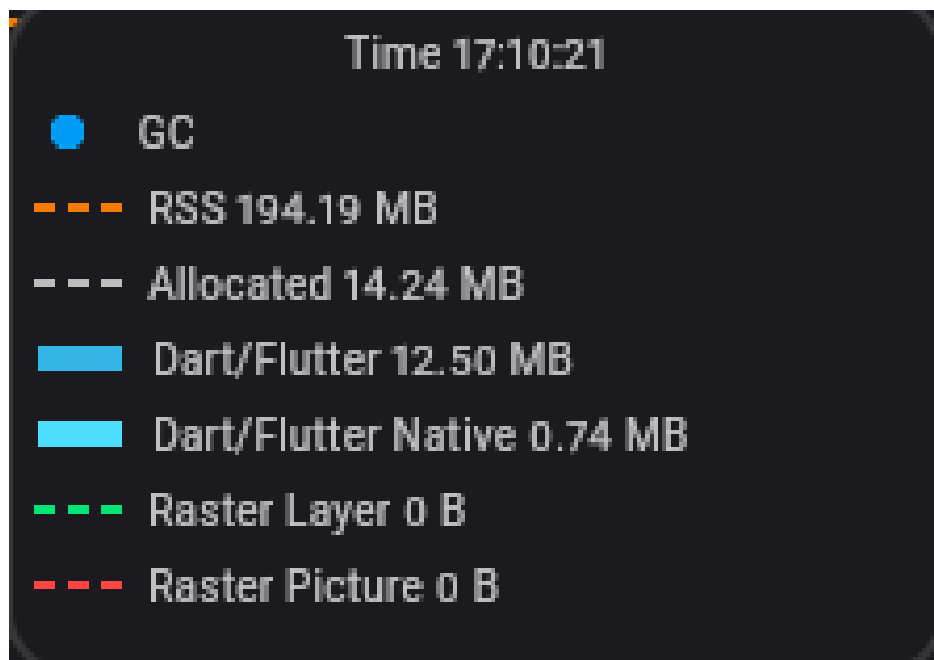


Imagen 43: Detalle de memoria en un evento de recolección de basura (GC)

En las figuras anteriores se muestra el comportamiento de diferentes aspectos del consumo de memoria a lo largo del tiempo. En la figura 1 muestra un eje vertical el cual se observa la cantidad de memoria utilizada, medida en megabytes (MB),

mientras que el eje horizontal representa el tiempo, permitiendo visualizar cómo varía el consumo durante la ejecución de la aplicación.

Mientras que en la figura 2 podemos observar los siguientes datos relevantes:

- **RSS (Resident Set Size):** Representado por una línea naranja punteada, el RSS indica la cantidad total de memoria física ocupada por la aplicación. En el análisis realizado, el RSS se mantiene estable en torno a los 194.19 MB, lo cual sugiere un uso constante y eficiente de la memoria por parte de la aplicación, sin picos ni incrementos bruscos que podrían indicar fugas de memoria.
- **Memoria Asignada (Allocated):** Representada por una línea amarilla punteada, la memoria asignada se mantiene en 14.24 MB durante el período de análisis. Esto indica que la aplicación está gestionando bien la memoria que reserva, sin sobrepasar los límites ni reservar más memoria de la necesaria, lo cual contribuye a un rendimiento eficiente.
- **Dart/Flutter y Dart/Flutter Native:** La línea azul clara muestra la memoria utilizada por los objetos y código Dart/Flutter, mientras que la línea azul más oscura representa la memoria utilizada por componentes nativos. La memoria utilizada por Dart/Flutter es de 12.50 MB, mientras que el uso nativo es mínimo (0.74 MB). Esto refleja que la mayoría del consumo de memoria proviene de la lógica Dart y el framework Flutter, mientras que los componentes nativos tienen un impacto reducido.

En la gráfica también se observan eventos de Garbage Collection (GC), representados por triángulos morados. Estos eventos ocurren de manera regular a lo largo del tiempo, indicando que el sistema de recolección de basura de Flutter está funcionando adecuadamente, liberando memoria no utilizada y evitando la acumulación de objetos innecesarios. Esto contribuye a la estabilidad del consumo de memoria y a la prevención de problemas de rendimiento.

En conclusión, el consumo de memoria de la aplicación Flutter muestra un comportamiento estable y eficiente. La memoria física ocupada (RSS) se mantiene constante sin variaciones significativas, lo cual indica que no hay fugas de memoria

aparentes. La memoria asignada es baja y constante, mientras que el uso de memoria por parte de componentes nativos es mínimo. Los eventos de recolección de basura son frecuentes y regulares, lo cual asegura la liberación de memoria no utilizada.

Conclusión

En conclusión, el análisis del rendimiento de la aplicación Flutter muestra que el mayor consumo de CPU se debe al renderizado y la programación de frames, lo cual impacta significativamente en la eficiencia de la aplicación. Optimizar estos aspectos podría mejorar la experiencia del usuario, especialmente en dispositivos con recursos limitados. Por otro lado, el consumo de memoria de la aplicación Flutter es estable y eficiente. La memoria física ocupada (RSS) se mantiene constante, sin fugas aparentes, y la memoria asignada es baja y constante, con un uso mínimo por parte de componentes nativos. Esto sugiere un manejo adecuado de los recursos de memoria, complementando así el análisis del rendimiento general.

iOS

A continuación, se mostrarán las pruebas de rendimiento realizadas a la aplicación hecha con el framework de Flutter en el sistema operativo de iOS en términos de consumo de CPU y RAM.

Análisis del Reporte de Uso del CPU

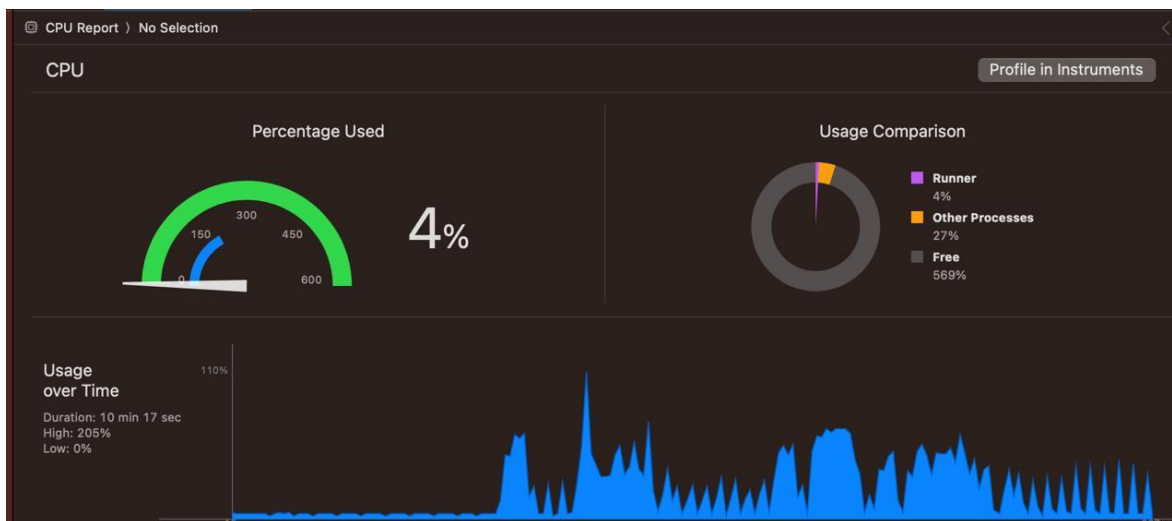


Imagen 44: Reporte de uso de CPU en Xcode Instruments de la aplicación de Flutter en iOS

En la figura anterior se muestra un reporte de uso del CPU obtenido a través de la herramienta de monitoreo que tiene Xcode, llamado Xcode Instruments, que proporciona información detallada sobre el comportamiento del procesador durante la ejecución de una aplicación móvil.

La figura muestra varios elementos clave para analizar el rendimiento del CPU:

1. **Porcentaje Usado:** El uso del CPU en el momento capturado es del 4%, lo que indica una demanda baja sobre los recursos del procesador. Esto sugiere que la aplicación en ejecución no requiere un alto consumo de procesamiento en este punto específico del análisis.
2. **Medidor de Uso del CPU:** El medidor semicircular representa visualmente el porcentaje de uso del CPU. La aguja se encuentra en la zona verde, lo cual refuerza la idea de un uso bajo y eficiente de los recursos disponibles. Esta representación es útil para identificar rápidamente si hay problemas de sobrecarga en el procesador.

3. **Comparación de Uso:** En la parte derecha del reporte, una gráfica de dona compara el uso del CPU entre diferentes procesos:
 - a. El Runner (la aplicación en prueba) está utilizando un 4% del CPU.
 - b. Otros procesos están utilizando un 27%.
 - c. La capacidad del CPU Libre es del 569%, lo cual se refiere a la disponibilidad de múltiples núcleos de procesamiento que no están siendo utilizados.
4. **Uso a lo Largo del Tiempo:** La gráfica en la parte inferior muestra el uso del CPU durante un periodo de 10 minutos y 17 segundos.
 - a. El valor máximo alcanzado durante este tiempo fue del 205%, lo que indica que en algunos momentos se utilizaron varios núcleos simultáneamente para procesar las tareas.
 - b. El valor mínimo fue del 0%, indicando que en ciertos momentos el CPU no tuvo carga alguna.
 - c. La gráfica azul muestra los cambios en la carga del CPU a lo largo del tiempo, con picos que representan momentos de mayor demanda.

Análisis del Reporte de Uso de Memoria



Imagen 45: Reporte de uso de memoria en Xcode Instruments de la aplicación de Flutter en iOS

En la figura anterior se muestra información clave sobre el uso de los recursos de memoria durante la ejecución de la aplicación:

1. **Uso de Memoria:** El reporte indica que la aplicación está utilizando 360 MB de memoria, lo cual representa un 9.4% del total disponible. Esto refleja que la aplicación tiene un impacto moderado sobre los recursos de memoria.
2. **Medidor de Uso de Memoria:** El medidor semicircular muestra el uso de memoria en diferentes niveles. En este caso, la aguja se encuentra en una zona verde, lo cual indica un uso eficiente y manejable de la memoria.
3. **Comparación de Uso de Memoria:** En la gráfica de dona se muestra la distribución del uso de memoria:
 - a. El Runner (la aplicación en prueba) está utilizando 360.1 MB de memoria.
 - b. Otros procesos están utilizando 2.48 GB.
 - c. La memoria Libre es de 944.4 MB, lo cual indica la disponibilidad de recursos de memoria para otras tareas.
4. **Uso de Memoria a lo Largo del Tiempo:** La gráfica en la parte inferior muestra el uso de memoria durante un periodo de 10 minutos y 29 segundos.
 - a. El valor máximo alcanzado fue de 418.4 MB y el valor mínimo registrado fue cero KB, lo cual indica que hubo momentos en los que no se utilizó memoria adicional.
 - b. La gráfica azul representa la estabilidad en el uso de la memoria, con variaciones leves a lo largo del tiempo.

Conclusiones

En conclusión, el uso del CPU y de la memoria revela que la aplicación evaluada tiene un impacto bajo tanto sobre el procesador como sobre la memoria, con gran parte de los recursos del sistema disponibles. Los picos observados en el uso del CPU y de la memoria indican actividad específica que incrementa temporalmente la carga, pero no se perciben problemas de rendimiento general. Este comportamiento es deseable para garantizar una experiencia fluida para el usuario y evitar la sobrecarga del dispositivo durante la ejecución de la aplicación.

Para llevar a cabo el segundo objetivo “Desarrollar una aplicación móvil para Android y iOS con el framework de React Native” se implementó el Ciclo de Vida de Desarrollo de Software (SDLC) para guiar las diferentes etapas del proyecto, desde el análisis hasta la implementación y pruebas. A continuación, se describe cada etapa de este ciclo aplicada al desarrollo de la aplicación móvil:

Ciclo de Vida de Desarrollo de Software para la Aplicación de React Native

1. Análisis
 - a. Requisitos funcionales.
 - b. Requisitos no funcionales.
2. Diseño
 - a. Diseño de interfaz de usuario UI.
 - b. Diagrama de componentes.
 - c. Diagrama de objetos.
3. Implementación (Codificación)
 - a. Configuración del entorno de desarrollo.
 - b. Codificación.
 - i. Módulos.
 1. Módulo de consumo de API de TheMovieDB.
 2. Módulo de favoritos con AsyncStorage.
4. Pruebas
 - a. Velocidad de ejecución.
 - b. Consumo de recursos (CPU y RAM).
 - i. Android Profiler.
 - ii. Xcode Instruments.

A continuación, se detallará cada una de las etapas realizadas para el desarrollo de la aplicación móvil para Android e iOS con el framework de Flutter

Análisis

Requisitos funcionales

1. Visualización de películas: La aplicación debe mostrar una lista de películas obtenidas desde la API.
2. Detalle de película: Al seleccionar una película, se debe mostrar su descripción y los actores que participaron en ella.
3. Agregar a favoritos: El usuario debe poder marcar una película como favorita.
4. Visualización de favoritos: La aplicación debe contar con un apartado donde el usuario pueda ver todas las películas que ha marcado como favoritas.
5. Sincronización con la API: La aplicación debe poder realizar peticiones a la API para obtener la lista actualizada de películas.
6. Gestión de estado: La aplicación debe recordar las películas marcadas como favoritas, incluso si se cierra y se vuelve a abrir.

Requisitos no funcionales

4. Compatibilidad y soporte multiplataforma: La aplicación debe funcionar en dispositivos Android e iOS, con una experiencia de usuario consistente en ambas plataformas para facilitar la comparación.
5. Facilidad de uso y navegación: La aplicación debe contar con una interfaz intuitiva que permita al usuario entender fácilmente cómo ver películas, agregar a favoritos y navegar entre secciones.
6. Uso de almacenamiento local: Debe optimizar el uso de almacenamiento local para los datos de favoritos, asegurando que el rendimiento no se vea afectado y que los datos se guarden de manera persistente incluso después de cerrar la aplicación.

Requisitos Técnicos

1. Sistema Operativo:
 - a. Android: La aplicación debe ser compatible con Android 10 (API nivel 29) y superior.
 - b. iOS: La aplicación debe ser compatible con iOS 12 y superior.
2. Plataformas de arquitectura:

- a. React Native: Usar React Native versión 0.70 o superior, compatible con Node.js (versión 14 o superior) para gestión de paquetes y dependencias.
- 3. Dispositivos físicos de prueba:
 - a. Android: Un dispositivo de gama media representativo, como el Samsung Galaxy A52 (Android 11, 6GB RAM, procesador Snapdragon 720G).
 - b. iOS: Un dispositivo de gama media en iOS, como el iPhone SE (2020) o el iPhone 11, ambos con iOS 14 o superior.
- 4. Entorno de desarrollo integrado (IDE):
 - a. React Native: Visual Studio Code y Android Studio para la configuración en Android y Xcode para iOS (versión 12 o superior).
- 5. Herramientas de perfilado de rendimiento específicas:
 - a. React Native: Herramientas de React DevTools y Flipper para perfilado de memoria y rendimiento.
- 6. Requerimientos de red:
 - a. Conectividad constante a internet para las pruebas de sincronización de datos con la API de películas.
- 7. Administración de dependencias:
 - a. React Native: Manejo de dependencias mediante package.json y compatibilidad con npm.

Diseño

Diseño de interfaz de usuario UI

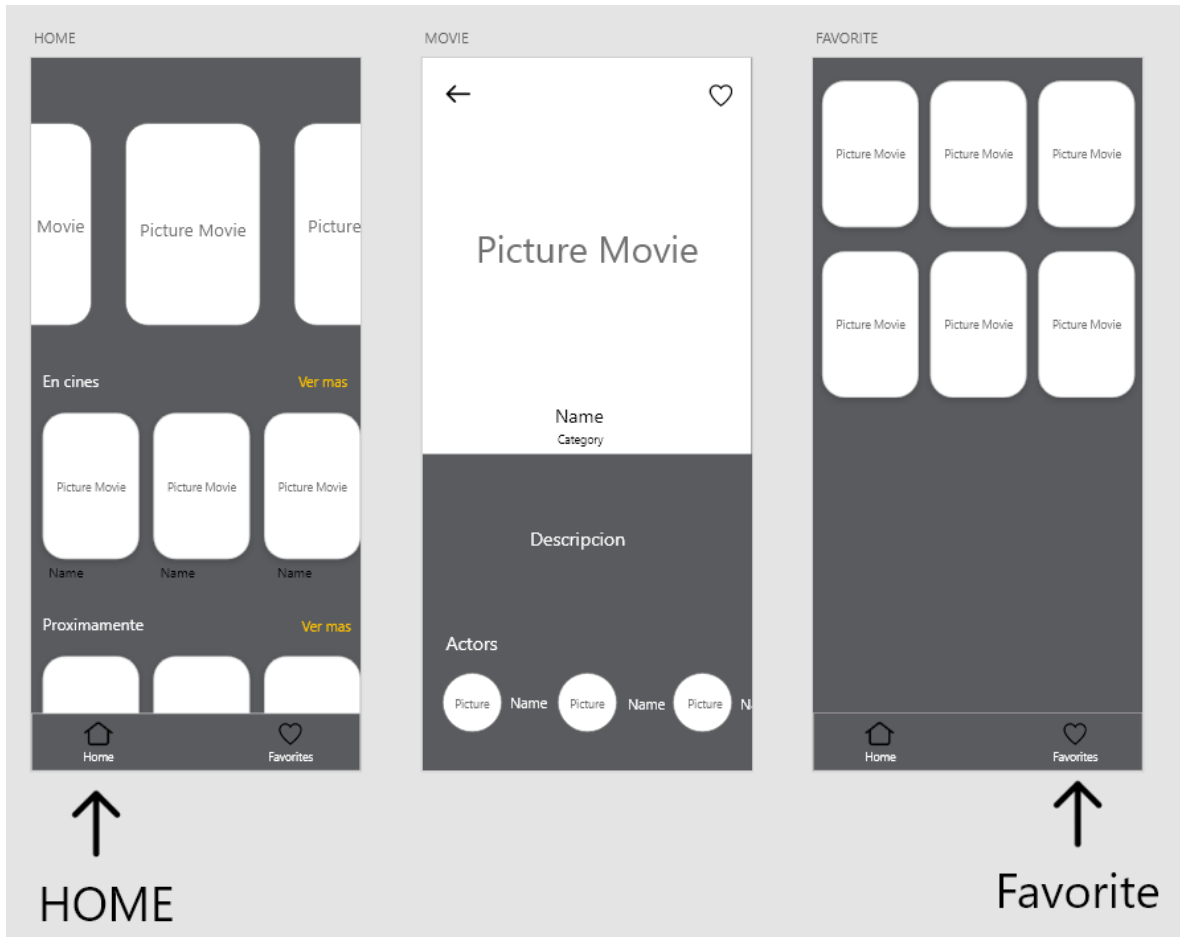


Imagen 46: Diseño de la Aplicación en React Native

Como se observa en la imagen anterior se tiene un diseño de tres pantallas para la aplicación móvil en las cuales destacan:

4. Pantalla HOME:

- En la parte superior, hay un carrusel de imágenes que muestra varias películas.
- Debajo, hay secciones etiquetadas como "En cines" y "Próximamente", que contienen tarjetas de películas con una imagen y el nombre de la película.

- En la parte inferior, hay una barra de navegación con 2 íconos: uno de Home (seleccionado) y el otro icono es un corazón el cual representa las películas favoritas guardadas.

5. Pantalla MOVIE:

- Esta pantalla muestra detalles de una película en específico.
- En la parte superior izquierda, hay un ícono de flecha para regresar.
- En la parte superior derecha, hay un icono para agregar la película a favoritos.
- En el centro, aparece el título "Picture Movie" y debajo, el nombre y una descripción de la película y su categoría.
- Más abajo, hay una sección que lista actores junto con sus nombres e imágenes.

6. Pantalla FAVORITE:

- Muestra una lista de películas favoritas con una cuadrícula de tarjetas que tendrán la imagen de la película".
- La barra de navegación en la parte inferior es la misma que la de la pantalla "HOME", pero aquí el ícono del corazón está seleccionado.

Diagrama de componentes

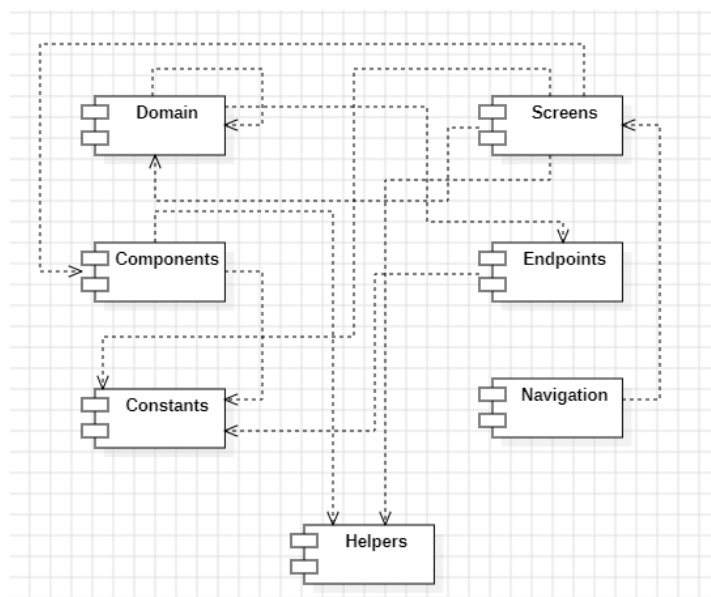


Imagen 47: Diagrama de Componentes de Aplicación Móvil en React Native

En la imagen anterior se visualiza como se organiza el proyecto en 7 carpetas principales: Domain, Components, Constants, Screens, Endpoints, Navigation y Helpers. Cada una tiene archivos que cumplen roles específicos en la obtención, procesamiento y visualización de los datos de películas, actores y favoritos.

1. Domain: Es la capa de lógica de negocio y obtención de datos. Aquí se definen funciones que interactúan con fuentes externas, como la API de TheMovieDB, para obtener información de películas y personas. Esta capa asegura que la lógica central de la aplicación esté encapsulada, facilitando la obtención de datos sin involucrar detalles de la presentación.
2. Components: Representa módulos de la interfaz de usuario reutilizables que permiten una construcción más eficiente de las pantallas. Aquí se encuentran elementos como listas de películas y componentes de carga, los cuales se integran en las pantallas para mantener la consistencia y modularidad de la UI.
3. Constants: Contiene valores fijos y constantes, como URLs predeterminadas o imágenes de respaldo. Esta carpeta centraliza estos valores para que puedan ser usados en toda la aplicación sin repetición, facilitando el mantenimiento.
4. Screens: Es la capa de presentación o interfaz de usuario, donde se organizan las vistas y se gestionan las interacciones con el usuario. Aquí se definen las pantallas principales (Home, Person y Movies), que hacen uso de la lógica de Domain y los componentes de Components.
5. Endpoints: Proporciona la configuración de las URLs o rutas necesarias para conectarse con la API de TheMovieDB. Esta carpeta centraliza la gestión de los endpoints, facilitando su actualización y mantenimiento.
6. Navigation: Contiene la configuración de la navegación de la aplicación, como la definición de rutas y stacks de navegación. Permite organizar y gestionar cómo los usuarios se mueven entre las diferentes pantallas.
7. Helpers: Proporciona funciones de apoyo para manipular y formatear datos, como las URL de las imágenes. Esta carpeta ayuda a simplificar el código

en otras partes de la aplicación, manteniendo la lógica organizada y reutilizable.

Interacción entre Capas

Cada capa interactúa para mantener una separación clara de responsabilidades:

- Screens dependen de Domain para recibir y mostrar la información procesada al usuario. Además, utilizan Components para construir la interfaz de usuario de forma modular y se apoyan en Navigation para gestionar el flujo de la aplicación.
- Domain se comunica con Endpoints para obtener y procesar datos externos, garantizando que la lógica de negocio se mantenga independiente de los detalles de implementación. También utiliza Helpers para simplificar y estandarizar operaciones.
- Endpoints implementan los detalles específicos de las rutas y URLs necesarias para la obtención de datos, permitiendo que Domain acceda a los datos sin preocuparse por las configuraciones exactas.
- Components se integran en Screens y usan Constants para mantener la consistencia en la interfaz. También dependen de Helpers para operaciones comunes, facilitando un desarrollo más modular y reutilizable.
- Constants proporcionan valores constantes y configuraciones reutilizables a Components, Screens, y Helpers, asegurando la consistencia en toda la aplicación.
- Helpers ofrecen funciones auxiliares utilizadas en todas las carpetas, apoyando tanto a Domain, Screens, como a Components, con funciones que simplifican operaciones comunes.
- Navigation organiza la forma en que los usuarios se mueven entre las Screens, y puede utilizar Constants para la configuración de rutas y nombres.

Diagrama de Objetos

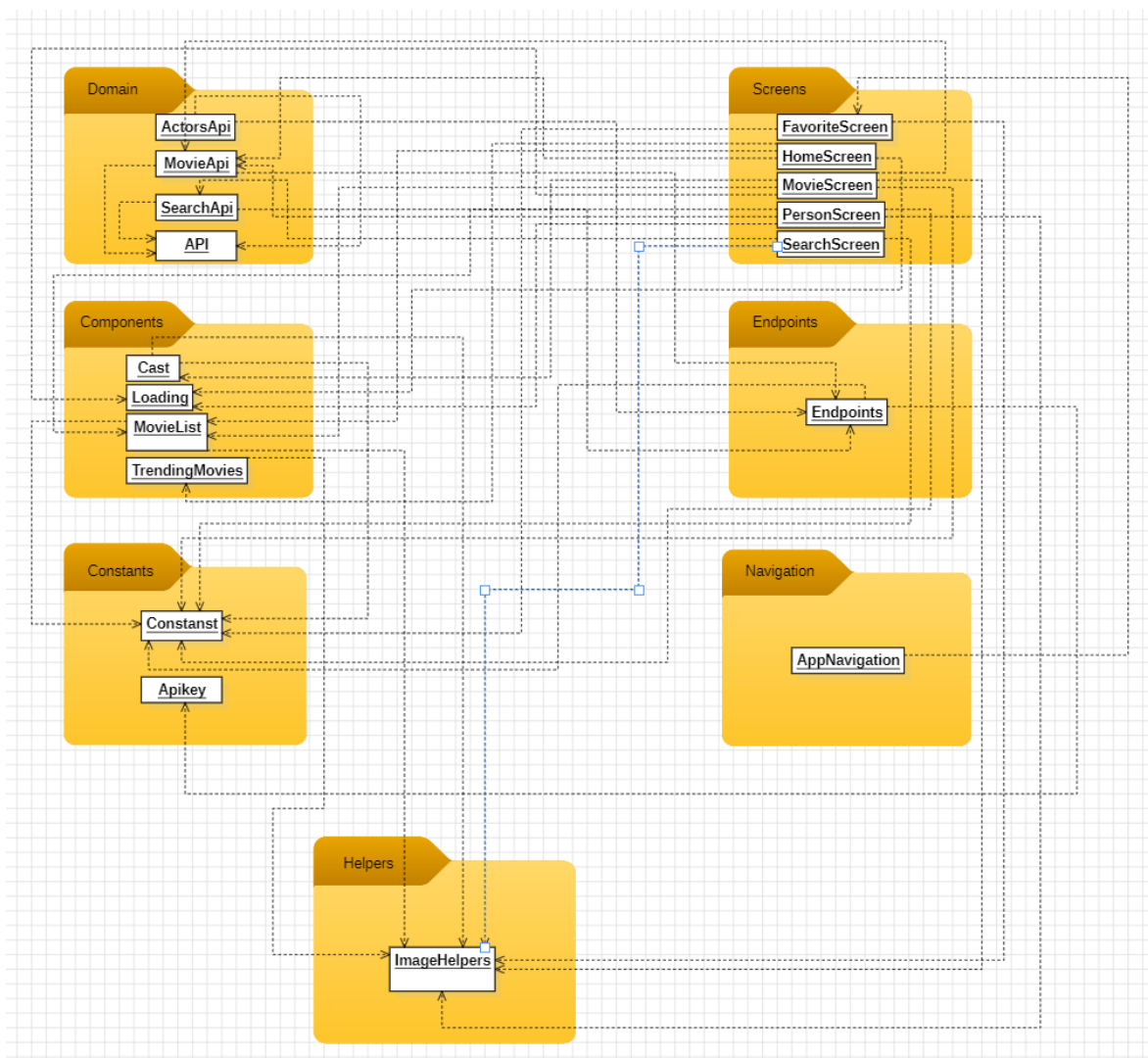


Imagen 48: Diagrama de Objetos de la Estructura de Carpetas en React Native

En la imagen anterior se visualiza cómo se organiza el proyecto en 7 carpetas principales: Domain, Components, Constants, Screens, Endpoints, Navigation, y Helpers. Cada una contiene archivos con roles específicos que contribuyen a la obtención, procesamiento y visualización de datos relacionados con películas, actores y favoritos. A continuación, se detalla el contenido y la interacción entre ellas:

1. Domain

Es la capa de lógica de negocio y obtención de datos. Aquí se encapsulan las interacciones con la API de TheMovieDB y se definen funciones específicas para cada caso de uso:

- a. actorsApi: Maneja la obtención de detalles y películas de un actor mediante funciones como fetchPersonDetails y fetchPersonMovies.
- b. api: Define la configuración base para realizar llamadas HTTP utilizando axios. Este archivo centraliza la lógica de solicitudes a la API.
- c. movieApi: Proporciona funciones para obtener películas populares, próximas, mejor calificadas, detalles de una película, el reparto de una película, y películas similares.
- d. searchApi: Maneja las búsquedas de películas a través de la API.

2. Components

Contiene módulos reutilizables de la interfaz de usuario que permiten construir pantallas de forma modular y consistente. Algunos de los componentes son:

- a. Cast: Muestra el reparto principal de una película, permitiendo la navegación hacia la pantalla de detalles de cada actor.
- b. Loading: Implementa un indicador de progreso para mostrar al usuario cuando los datos están cargándose.
- c. MovieList: Despliega una lista de películas, como "Upcoming" o "Top Rated", con navegación a la pantalla de detalles de cada película.
- d. TrendingMovies: Implementa un carrusel que muestra las películas populares en el momento.

3. Constants

Centraliza valores fijos y constantes, simplificando el mantenimiento y asegurando la consistencia en toda la aplicación:

- a. apiKey: Almacena la clave de acceso a la API de TheMovieDB.
- b. constans: Define valores como:
 - a. apiBaseUrl: URL base para la API.

- b. `fallbackMoviePoster` y `fallbackPersonImage`: Imágenes predeterminadas en caso de que no haya un recurso disponible.

4. Screens

Representa la capa de presentación o interfaz de usuario, donde se organizan las vistas principales y se gestionan las interacciones con el usuario. Incluye:

- a. `FavoriteScreen`: Permite al usuario ver y gestionar su lista de películas favoritas almacenadas localmente con `AsyncStorage`.
- b. `HomeScreen`: Es la pantalla inicial que muestra las películas populares, próximas y mejor calificadas, organizadas en secciones.
- c. `MovieScreen`: Despliega detalles de una película, como título, descripción, géneros, reparto, y películas similares. También permite agregar o eliminar la película de la lista de favoritos.
- d. `PersonScreen`: Muestra detalles de un actor, incluyendo su biografía, lugar de nacimiento, y las películas en las que ha participado.
- e. `SearchScreen`: Proporciona una barra de búsqueda para buscar películas y muestra los resultados en tiempo real.

5. Endpoints

Centraliza las URLs y rutas necesarias para interactuar con la API de `TheMovieDB`:

- a. Define endpoints específicos como `trendingMoviesEndpoint`, `movieDetailsEndpoint`, `personDetailsEndpoint`, entre otros.
- b. Utiliza las constantes de `Constants` para construir las rutas dinámicamente, asegurando que cualquier cambio en la configuración sea centralizado.

7. Navigation

Contiene la configuración de navegación de la aplicación. Define cómo los usuarios se mueven entre las pantallas mediante:

- a. AppNavigation: Configura un Stack Navigator para la navegación principal y un Tab Navigator que organiza las pantallas de inicio y favoritos.

8. Helpers

Proporciona funciones auxiliares para manipular y formatear datos, como URLs de imágenes. Incluye:

- a. imageHelpers: Define funciones como image500, image342, y image185 para generar URLs de imágenes en diferentes resoluciones.

Interacción entre Capas

La interacción entre estas capas asegura una separación clara de responsabilidades:

Screens dependen de Domain para obtener los datos procesados que muestran al usuario. También hacen uso de Components para construir una interfaz de usuario modular y de Navigation para gestionar el flujo de la aplicación.

Domain utiliza Endpoints para acceder a los datos de la API y Helpers para formatear los datos, manteniendo la lógica de negocio independiente.

Endpoints implementan las rutas necesarias para que Domain pueda acceder a los datos externos.

Components integran datos provenientes de Domain y utilizan constantes de Constants para mantener la consistencia visual y funcional.

Constants asegura configuraciones y valores compartidos que se utilizan en Domain, Screens, y Components.

Helpers proporciona utilidades compartidas que simplifican operaciones comunes y son utilizadas por Domain, Components, y Screens.

Navigation organiza cómo los usuarios interactúan con las diferentes Screens, asegurando una experiencia de usuario fluida.

Implementación

Configuración del entorno de desarrollo

Como antes mencionado estamos trabajando con frameworks de desarrollo multiplataforma por lo que no estamos restringidos por un sistema operativo para poder trabajar con el framework de React Native por ende el desarrollo de esta aplicación con dicho framework se trabajara con un equipo MacBook Pro con macOS Ventura como sistema operativo.

Requisitos de Hardware

Los requisitos mínimos para trabajar con el framework de React Native son los siguientes:

- Al menos 4 GB de memoria RAM.
- 10 GB de almacenamiento.
- Sistema macOS 10.10

Sin embargo, en este estudio se trabajará con un equipo con las siguientes características:

- 16 GB de memoria RAM.
- 300 GB de almacenamiento disponible.
- Sistema macOS 14.7
- Procesador Intel Core i7 3.1 GHz.

A continuación, se muestra los pasos para una instalación correcta de las herramientas necesarias para comenzar a desarrollar la aplicación móvil en React Native:

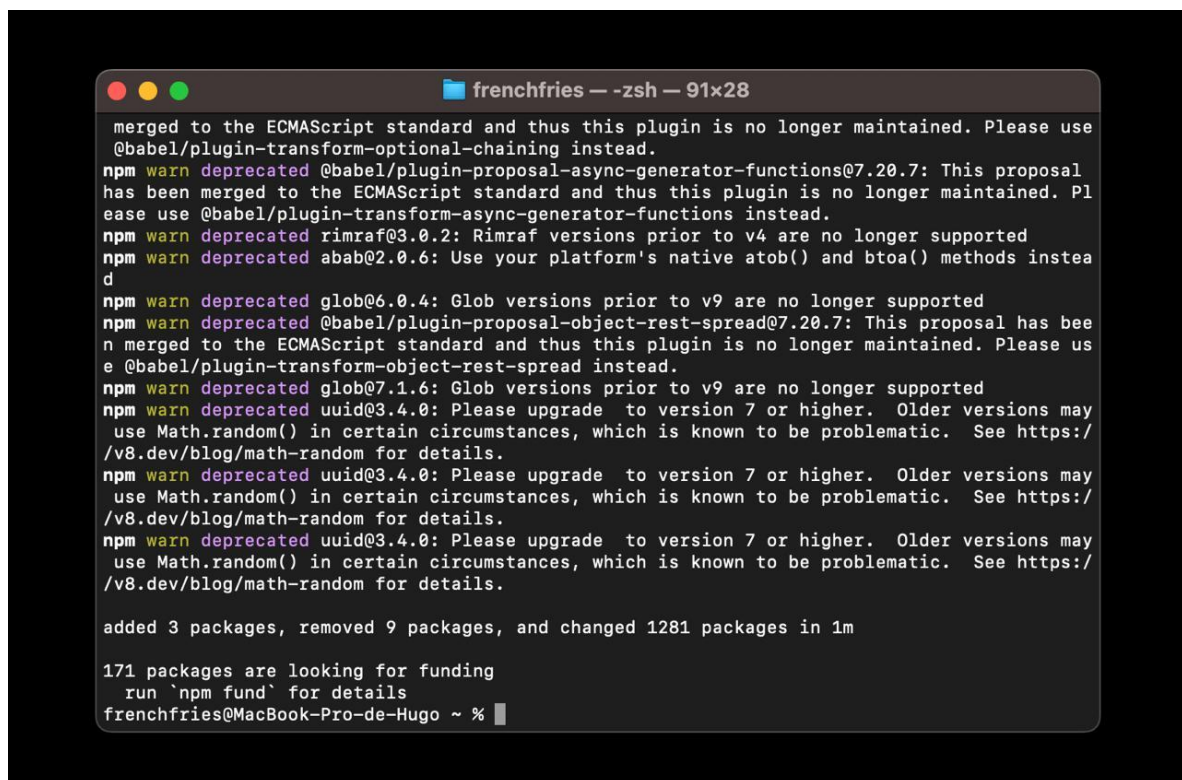
Instalaciones necesarias

Para trabajar con el framework de React Native es necesario contar con Node.js de primera instancia en el equipo para comenzar. Existen 2 formas de tener Node.js en un equipo macOS las cuales son:

- Descargar e instalar Node.js desde su página oficial en <https://nodejs.org/en/download/package-manager/current>.
- Instalar desde la terminal con el comando “**brew install node**” (en el caso de no contar con “brew” es necesario instalarlo desde la terminal con el comando `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`).

Cabe resaltar que hay dos formas de trabajar un proyecto con React Native para el desarrollo móvil, una de ellas es utilizando **React Native CLI** y la otra es utilizando **Expo CLI**, en dicho caso trabajaremos con Expo CLI para gestionar el proyecto en React Native.

Para instalar Expo CLI es necesario ejecutar el siguiente comando de manera global **npm install -g expo-cli**, una vez ejecutado el comando se comenzará la descarga de la herramienta mostrando como resultado final lo siguiente:



```
frenchfries — zsh — 91x28
merged to the ECMAScript standard and thus this plugin is no longer maintained. Please use
@babel/plugin-transform-optional-chaining instead.
npm warn deprecated @babel/plugin-proposal-async-generator-functions@7.20.7: This proposal
has been merged to the ECMAScript standard and thus this plugin is no longer maintained. Pl
ease use @babel/plugin-transform-async-generator-functions instead.
npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm warn deprecated abab@2.0.6: Use your platform's native atob() and btoa() methods instea
d
npm warn deprecated glob@6.0.4: Glob versions prior to v9 are no longer supported
npm warn deprecated @babel/plugin-proposal-object-rest-spread@7.20.7: This proposal has bee
n merged to the ECMAScript standard and thus this plugin is no longer maintained. Please us
e @babel/plugin-transform-object-rest-spread instead.
npm warn deprecated glob@7.1.6: Glob versions prior to v9 are no longer supported
npm warn deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may
use Math.random() in certain circumstances, which is known to be problematic. See https:/
/v8.dev/blog/math-random for details.
npm warn deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may
use Math.random() in certain circumstances, which is known to be problematic. See https:/
/v8.dev/blog/math-random for details.
npm warn deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may
use Math.random() in certain circumstances, which is known to be problematic. See https:/
/v8.dev/blog/math-random for details.

added 3 packages, removed 9 packages, and changed 1281 packages in 1m

171 packages are looking for funding
  run `npm fund` for details
frenchfries@MacBook-Pro-de-Hugo ~ %
```

Imagen 49: Finalización de la instalación de Expo CLI con npm

Si bien, React Native no necesita muchas instalaciones de herramientas para comenzar a trabajar, la creación del proyecto puede ocasionar errores que no permitan seguir con el proceso de desarrollo es por eso que a continuación mostraremos la manera correcta de crear un proyecto para el desarrollo de una aplicación móvil para Android e iOS.

Pasos para crear un proyecto de React Native con Expo:

1. Crear un proyecto con Expo con el comando **expo init Miproyecto**.
2. Seleccionar el template **blank** a **minimal app as clean as an empty canvas**.
3. Instalar expo en el proyecto utilizando el comando **npx expo install**.
4. Crear las carpetas de Android e iOS con el comando **npx expo prebuild** y nombrar los package como por ejemplo **com.anonymous.Miproyecto**.
5. Ejecutar el comando **npx expo start** para iniciar el servidor de desarrollo.

```
frenchfries@MacBook-Pro-de-Hugo prueba % expo init prueba
WARNING: The legacy expo-cli does not support Node +17. Migrate to the new local Expo CLI: https://blog.expo.dev/the-new-expo-cli-f4250d8e3421.
(node:3531) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)

The global expo-cli package has been deprecated.

The new Expo CLI is now bundled in your project in the expo package.
Learn more: https://blog.expo.dev/the-new-expo-cli-f4250d8e3421.

To use the local CLI instead (recommended in SDK 46 and higher), run:
> npx expo <command>

Migrate to using:
> npx create-expo-app --template

? Choose a template: > - Use arrow-keys. Return to submit.
  Managed workflow
> blank a minimal app as clean as an empty canvas
  blank (TypeScript) same as blank but with TypeScript configuration
  tabs (TypeScript) several example screens and tabs using react-navigation and TypeScript
  Bare workflow
  minimal bare and minimal, just the essentials to get you started
```

Imagen 50: Creación de proyecto en React Native con Expo: punto 1 y 2


```

frenchfries@MacBook-Pro-de-Hugo prueba % sudo npx expo install
Password:
(node:1845) [DEP0040] DeprecationWarning: The `punycode` module
(Use `node --trace-deprecation ...` to show where the warning wa
> Installing using npm
> npm install

added 1 package, and audited 1260 packages in 2s

131 packages are looking for funding
  run `npm fund` for details

3 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
frenchfries@MacBook-Pro-de-Hugo prueba % █

```

Imagen 51: Creación de proyecto en React Native con Expo: punto 3

```

frenchfries@MacBook-Pro-de-Hugo prueba % sudo npx expo prebuild
(node:2018) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Ple
(Use `node --trace-deprecation ...` to show where the warning was created)


📦 Android package Learn more: https://expo.fyi/android-package
✓ What would you like your Android package name to be? ... com.anonymous.prueba

📦 iOS Bundle Identifier Learn more: https://expo.fyi/bundle-identifier
✓ What would you like your iOS bundle identifier to be? ... com.anonymous.prueba

```

Imagen 52: Creación de proyecto en React Native con Expo: punto 4


```
frenchfries@MacBook-Pro-de-Hugo prueba % npx expo start
Starting project at /Users/frenchfries/Desktop/prueba/prueba
(node:3237) [DEP0040] DeprecationWarning: The `punycode` module is deprecated
(Use `node --trace-deprecation ...` to show where the warning was created)
Starting Metro Bundler



> Metro waiting on exp://192.168.0.11:8081
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)
> Using Expo Go
```

Imagen 53: Creación de proyecto en React Native con Expo: punto 5

Módulos

Módulo de Consumo de API de TheMovieDB

El Módulo de Consumo de API de TheMovieDB es esencial para acceder a la información más actualizada sobre películas en nuestra aplicación. Este módulo gestiona las solicitudes HTTP a la API de TheMovieDB para recibir datos en tiempo real, como listas de películas populares, próximas, mejor calificadas y detalles específicos de cada película, así como datos sobre actores. Al recibir las respuestas en formato JSON, el módulo se encarga de extraer los datos relevantes para utilizarlos en la capa de presentación de la aplicación.

En este proyecto de React Native, y dadas las limitaciones de tiempo, la implementación es más simple en comparación con una estructura de capas más robusta como en Flutter. La arquitectura se compone de:

- `api.js`: un archivo centralizado que configura las solicitudes API mediante `axios`, permitiendo manejar cualquier error que pueda surgir durante la comunicación con la API.
- `endpoints.js`: archivo que almacena las URL para cada recurso, formateadas con el `apiBaseUrl` y la `apiKey`.
- `movieApi.js`: archivo donde se definen funciones específicas que llaman a `apiCall` con los distintos endpoints, cada uno correspondiente a un recurso de TheMovieDB (como películas populares, próximas o similares).

Esta implementación simplificada no afecta el análisis de rendimiento y facilidad de aprendizaje que se llevará a cabo en el proyecto. La estructura adoptada permite una comunicación eficiente con la API, proporcionando la información necesaria para la aplicación de manera rápida y estable, mientras se enfoca en el desarrollo ágil requerido para cumplir con los tiempos de entrega.

Por lo tanto, a continuación, se mostrarán los archivos con su contenido y la forma en que estos trabajan entre sí para mantener el consumo de la API de manera eficiente.

```
import axios from "axios";

const apiCall = async (endpoint, params) => {
  const options = {
    method: 'GET',
    url: endpoint,
    params: params ? params : {}
  };

  try {
    const response = await axios.request(options);
    return response.data;
  } catch (error) {
    console.error('Error:', error);
    return {};
  }
};

export default apiCall;
```

Imagen 54: Función genérica para realizar peticiones GET con Axios en JavaScript

Como se puede observar en la imagen anterior tenemos un archivo llamado **api.js** el cual se maneja una función asíncrona llamada **apical** la cual realiza solicitudes HTTP de tipo GET usando el paquete Axios en JavaScript, este paquete facilita el manejo de solicitudes a APIs y la captura de respuestas, permitiendo que la comunicación con servicios externos sea de manera limpia y eficiente.

Dentro de **api.js** tenemos algunos elementos importantes para el buen consumo de la API los cuales son:

1. Parámetros de la función: Como podemos observar la función **apical** tiene dos parámetros que nos servirán para hacer una petición de tipo GET, además, dada que la función está definida como **async** esta nos permite el use de **await** dentro de ella para realizar operaciones asíncronas.
 - a. **Endpoint:** Este es la URL a la cual se enviará la solicitud.

- b. **Params:** Este es un objeto opcional que contiene los parámetros de la solicitud.
- 2. Objeto **options**: dentro de la función de **apiCall** se define un objeto llamado **options** el cual contiene un mapa clave-valor para la llamada.
 - a. **method: 'GET'**: Este elemento especifica el tipo de solicitud HTTP (en este caso GET).
 - b. **url: endpoint**: Este define la URL a la cual se enviará la solicitud.
 - c. **params: params ? params : {}**: Este último sirve por si en la solicitud se proporciona parámetros, este se incluye en la solicitud, de lo contrario se utiliza un objeto vacío.
- 3. Solicitud y manejo de errores: Por último, esta función ejecuta la solicitud HTTP con los datos proporcionados y si llegara a existir un error se maneja un registro en consola.
 - a. Ejecución de la solicitud: La ejecución de la solicitud HTTP se logra usando **axios.request(options)** y con el **await** espera a tener una respuesta.
 - b. Retorno de datos: Si la solicitud es exitosa, **response.data** contiene estos datos recibidos de la API y se devuelve a quien llamo la función (las vistas de renderizado de las películas como el HomeScreen, MovieScreen, etc.).

```
import { apiBaseUrl } from "../constants/constans";
import { apiKey } from "../constants/apiKey";

export const trendingMoviesEndpoint = `${apiBaseUrl}
/trending/movie/day?api_key=${apiKey}`;
export const upcomingMoviesEndpoint = `${apiBaseUrl}
/movie/upcoming?api_key=${apiKey}`;
export const topRatedMoviesEndpoint = `${apiBaseUrl}
/movie/top_rated?api_key=${apiKey}`;
export const searchMoviesEndpoint = `${apiBaseUrl}
/search/movie?api_key=${apiKey}`;

export const movieDetailsEndpoint = id => `${apiBaseUrl}
/movie/${id}?api_key=${apiKey}`;
export const movieCreditsEndpoint = id => `${apiBaseUrl}
/movie/${id}/credits?api_key=${apiKey}`;
export const similarMoviesEndpoint = id => `${apiBaseUrl}
/movie/${id}/similar?api_key=${apiKey}`;

export const personDetailsEndpoint = id => `${apiBaseUrl}
/person/${id}?api_key=${apiKey}`;
export const personMoviesEndpoint = id => `${apiBaseUrl}
/person/${id}/movie_credits?api_key=${apiKey}`;
```

Imagen 55: Definición de endpoints de la API de TheMovieDB en JavaScript

La imagen anterior nos muestra una serie de **endpoints** para hacer peticiones específicas a la API de TheMovieDB. Utiliza valores predefinidos como la URL base de la API (**apiBaseUrl**) y la clave de la API (**apiKey**), almacenados en archivos separados para mantener el código limpio y la información sensible protegida.

Sin embargo, manejamos 2 tipos diferentes de **endpoints**, los primeros 4 permiten acceder a diferentes tipos de datos sobre las películas como:

1. **trendingMoviesEndpoint**: URL para obtener las películas en tendencia del día.

2. **upcomingMoviesEndpoint:** URL para obtener las películas que se estrenarán próximamente.
3. **topRatedMoviesEndpoint:** URL para obtener las películas mejor valoradas.
4. **searchMoviesEndpoint:** URL para buscar películas según un criterio de búsqueda.

Por último, tenemos los **endpoints** que se construyen dinámicamente según el ID de la película y actores:

1. **movieDetailsEndpoint(id):** Genera la URL para obtener los detalles de una película específica usando su ID.
2. **movieCreditsEndpoint(id):** Genera la URL para obtener la información del reparto y equipo de una película usando su ID.
3. **similarMoviesEndpoint(id):** Genera la URL para obtener películas similares a la indicada por el ID.
4. **personDetailsEndpoint(id):** Genera la URL para obtener detalles de una persona (actor, director, etc.) según su ID.
5. **personMoviesEndpoint(id):** Genera la URL para obtener los créditos de películas en las que una persona ha trabajado.

```

import apiCall from "../api";
import {
  trendingMoviesEndpoint,
  upcomingMoviesEndpoint,
  topRatedMoviesEndpoint,
  movieDetailsEndpoint,
  movieCreditsEndpoint,
  similarMoviesEndpoint
} from "../endpoints/endpoints";

// Movie APIs
export const fetchTrendingMovies = () =>
  apiCall(trendingMoviesEndpoint);
export const fetchUpcomingMovies = () =>
  apiCall(upcomingMoviesEndpoint);
export const fetchTopRatedMovies = () =>
  apiCall(topRatedMoviesEndpoint);
export const fetchMovieDetails = (id) =>
  apiCall(movieDetailsEndpoint(id));
export const fetchMovieCredits = (movieId) =>
  apiCall(movieCreditsEndpoint(movieId));
export const fetchSimilarMovies = (movieId) =>
  apiCall(similarMoviesEndpoint(movieId));

```

Imagen 56: Funciones para consumir los endpoints de la API de TheMovieDB utilizando una capa de abstracción en JavaScript

Como se muestra en la imagen anterior tenemos funciones previamente definidas y endpoints para interactuar con una API de películas, facilitando las solicitudes HTTP para obtener diferentes tipos de datos. Las funciones aquí definen cómo obtener datos específicos de la API, cada una llamando a la función `apiCall` y proporcionando un endpoint adecuado.

Cada una de estas funciones se utiliza para hacer las peticiones a la API con respecto a lo que se necesite tales como:

1. **fetchTrendingMovies():** Llama a **apiCall** con el endpoint para obtener las películas en tendencia.
2. **fetchUpcomingMovies():** Llama a **apiCall** con el **endpoint** para obtener las películas que se estrenarán próximamente.
3. **fetchTopRatedMovies():** Llama a **apiCall** con el **endpoint** para obtener las películas mejor valoradas.
4. **fetchMovieDetails(id):** Llama a **apiCall** con el **endpoint** para obtener los detalles de una película específica, utilizando el ID de la película.
5. **fetchMovieCredits(movieId):** Llama a **apiCall** con el **endpoint** para obtener los detalles del reparto de una película específica, utilizando el ID de la película.
6. **fetchSimilarMovies(movieId):** Llama a **apiCall** con el **endpoint** para obtener las películas similares a una película específica, utilizando el ID de la película.

Módulo de Favoritos con AsyncStorage

El Módulo de Favoritos permite a los usuarios guardar y gestionar sus películas favoritas de manera local en la aplicación. Dado el tiempo limitado para implementar este módulo, se ha optado por usar AsyncStorage en lugar de una base de datos local más avanzada como SQLite o Isar, asegurando así una persistencia básica de datos en el dispositivo del usuario. Esto permite que los usuarios puedan acceder a su lista de favoritos sin necesidad de conexión a internet, aunque con algunas limitaciones en comparación con una base de datos más robusta.

La implementación de la funcionalidad de favoritos se integra directamente en las pantallas principales de la aplicación, en lugar de tener una estructura dedicada de carpetas o capas, e incluye:

- **MovieScreen.js:** Permite al usuario agregar o quitar una película de su lista de favoritos. La función `addNewItem` gestiona el almacenamiento o eliminación de favoritos en AsyncStorage.
- **FavoriteScreen.js:** Muestra la lista completa de favoritos y permite eliminar elementos de ella. Al utilizar AsyncStorage, el módulo realiza operaciones de almacenamiento y recuperación de datos de manera eficiente, adecuada para el almacenamiento ligero.

Aunque se ha optado por una solución simplificada debido a las limitaciones de tiempo, esto no interfiere en el análisis de rendimiento y facilidad de aprendizaje del proyecto. Al contrario, el uso de AsyncStorage proporciona una implementación ágil y directa que cumple con los requisitos esenciales de persistencia y permite una experiencia de usuario fluida, manteniendo un enfoque en los objetivos de rendimiento del proyecto.

Teniendo en cuenta lo anterior la lógica del guardado de las películas favoritas se realiza en la pantalla de **MovieScreen** o en su caso en dicho archivo.

La lógica para guardar la película a favoritos en esta pantalla se maneja principalmente a través del estado **isFavourite** y el uso de **AsyncStorage** para almacenar la lista de películas favoritas de manera persistente.

```
export default function MovieScreen() {

  const [isFavourite, setIsFavourite] = useState(false);
  const [existingData, setExistingData] = useState([]);

  useEffect(() => {
    setLoading(true);
    getMovieDetials(item.id);
    getMovieCredits(item.id);
    getSimilarMovies(item.id);
    checkExistingItem();
  }, [item]);
}
```

Imagen 57: Pantalla MovieScreen con gestión de favoritos y verificación de datos existentes

Como se muestra en la imagen anterior tenemos los elementos principales de **isFavorite** y **existingData** los cuales son nuestros useState para guardar los estados iniciales. De la siguiente forma es como trabajan:

1. **isFavourite**: Se utiliza para determinar si la película actual está marcada como favorita.
2. **existingData**: Almacena las películas favoritas existentes que se obtienen de AsyncStorage.

Después de declarar nuestros estados iniciales dentro del useEffect se llama a la función **checkExistingItem** al cargar la pantalla para verificar si la película actual ya está en la lista de favoritos

```
const checkExistingItem = async () => {
  try {
    // get existing data.
    let getExistingData = await AsyncStorage.getItem('favoritelist');
    getExistingData = getExistingData !== null ?
      JSON.parse(getExistingData) : [];
    // check if current item exists or not.
    const checkExistingItem = [...getExistingData].
      some(existingItem => existingItem.id === item.id)
    setIsFavourite(checkExistingItem);
    setExistingData(getExistingData);
  } catch (error) {
    console.log('checkExistingItem error => ', error);
  }
}
```

Imagen 58: Función checkExistingItem para verificar y gestionar favoritos en AsyncStorage

Como se puede observar en la imagen anterior, dicho código nos muestra la verificación inicial de las películas si es que ya existe en favoritos o no, y es así como trabaja de la siguiente manera:

1. **checkExistingItem** intenta recuperar los datos almacenados en **AsyncStorage** bajo la clave **'favoritelist'**.
2. Si la lista de favoritos existe, convierte los datos a formato JSON y verifica si la película actual ya se encuentra en la lista.
3. Dependiendo de esto, actualiza el estado de **isFavourite** para reflejar si la película ya es favorita o no.

Por último, tenemos la función para agregar o eliminar una película a favoritos la cual nos ayudara a agregar a nuestra lista y próximamente tenerla en la vista de favoritos.

```

const addNewItem = async () => {
  try {
    if (isFavourite) {
      let tmp = [...existingData];
      console.log(tmp.length)

      tmp = tmp.filter(favoriteData => {
        return favoriteData?.id !== item?.id
      });
      await AsyncStorage.setItem('favoritelist', JSON.stringify(tmp));
      setIsFavourite(false);
      return;
    }
    const tmp = [...existingData, item];
    await AsyncStorage.setItem('favoritelist', JSON.stringify(tmp));
    setIsFavourite(true);
  } catch (error) {
    console.log('addNewItem error => ', error);
  }
}

```

Imagen 59: Función addNewItem para agregar o eliminar elementos de la lista de favoritos en AsyncStorage

En la imagen anterior podemos observar la función de **addNewItem** el cual trabaja de la siguiente manera:

1. **addNewItem** se ejecuta al hacer clic en el ícono de corazón (**HeartIcon**).
2. Si la película ya es favorita (**isFavourite** es true):
 - a. Se filtra la lista para eliminar la película actual.
 - b. Luego, la nueva lista de favoritos (sin la película) se guarda de nuevo en **AsyncStorage**.
 - c. Se actualiza el estado para indicar que ya no es favorita.
3. Si la película no es favorita (**isFavourite** es false):
 - a. Se agrega la película a la lista de favoritos existente.
 - b. La nueva lista se guarda en **AsyncStorage**.
 - c. Se actualiza el estado para indicar que ahora es favorita.

Por último, tenemos la lógica para mostrar los favoritos en la pantalla de **FavoriteScreen.js** el cual se basa principalmente en el uso de **AsyncStorage** para recuperar la lista de películas guardadas como favoritas anteriormente dentro de **MovieScreen.js**

```
const FavoriteScreen = (props) => {

  const [favoriteData, setFavoriteData] = useState([]);
  const isFocused = useIsFocused();
```

Imagen 60: Pantalla FavoriteScreen con gestión de datos de favoritos

Como se puede observar en la imagen anterior, dentro de **FavoriteScreen.js** tenemos el estado y dependencias iniciales para mostrar las películas añadidas a favoritas.

1. **favoriteData:** Un estado (**useState**) para almacenar la lista de películas favoritas recuperada desde **AsyncStorage**.
2. **isFocused:** Se utiliza **useIsFocused** de la librería de navegación para verificar si la pantalla **FavoriteScreen** está enfocada, de modo que cuando el usuario vuelve a esta pantalla, la lista de favoritos se recarga automáticamente.

```
const getData = async () => {
  let response = await AsyncStorage.getItem('favoritelist');
  if (response) {
    response = JSON.parse(response);
    setFavoriteData(response);
  }
}
```

Imagen 61: Función getData para recuperar y actualizar datos de favoritos desde AsyncStorage

Dentro de **FavoriteScreen.js** tenemos la función **getData** de manera asíncrona para obtener los datos guardados en **AsyncStorage** el cual funciona de la siguiente forma:

1. La función **getData** intenta obtener los favoritos almacenados en **AsyncStorage** bajo la clave **'favoritelist'** (la cual fue usada para guardar las películas inicialmente).

2. Si se encuentran datos, los convierte a formato JSON y los asigna al estado **favoriteData**.

Por último, tenemos la recarga de los datos por si es que existe un cambio este pueda reflejarse de la manera más rápida posible al usuario.

```
useEffect(() => {  
  if (isFocused) {  
    getData();  
  }  
}, [isFocused]);
```

Imagen 62: Hook useEffect para sincronizar datos de favoritos al enfocar la pantalla en React

Como se puede ver en la imagen anterior, para el manejo de recarga de los datos se utiliza un **useEffect** el cual es un hook que nos permite manejar efectos secundarios como es el caso de mostrar los datos y es así como funciona este **useEffect**:

- Cuando la pantalla **FavoriteScreen** está enfocada (**isFocused**), se llama a la función **getData()** para recargar la lista de favoritos.

Esto garantiza que los favoritos se mantengan actualizados cuando el usuario regrese a la pantalla.

Pruebas

En este apartado se realizó una evaluación del rendimiento de la aplicación desarrollada con React Native, enfocándose en medir la velocidad de ejecución y el consumo de recursos. Para ello, se utilizaron herramientas como Android Profiler y Xcode Instruments. A través de estas pruebas, se busca identificar las capacidades y limitaciones de React Native en el desarrollo de aplicaciones móviles, permitiendo analizar cómo este framework gestiona los recursos del sistema y cuál es su impacto en el rendimiento general de la aplicación.

Velocidad de Ejecución

Como uno de los aspectos principales es medir la velocidad de ejecución desde el momento en que la aplicación es abierta hasta que los datos se cargan por completo, la mejor manera de determinar este tiempo es registrando la duración del proceso. A continuación, se presenta una tabla con los tiempos exactos de carga de la aplicación tanto para Android como para iOS:

Dispositivo	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
Android	3.23 s	2.79 s	3.08	2.81 s	2.75 s
iOS	2.10 s	2.01 s	1.98 s	2.96 s	2.18 s

Tabla 4: Tiempos de Velocidad de Ejecución de la App en React Native en Android e iOS

A partir de los resultados obtenidos de las pruebas de velocidad de ejecución en dispositivos Android e iOS, se pueden realizar los siguientes análisis y conclusiones:

1. Promedio de Tiempos de Ejecución:

- Android: Promedio = $(3.23 + 2.79 + 3.08 + 2.81 + 2.75) / 5 = 2.93$ segundos.
- iOS: Promedio = $(2.10 + 2.01 + 1.98 + 2.96 + 2.18) / 5 = 2.25$ segundos.

Los resultados muestran que iOS tiene un rendimiento promedio más rápido en comparación con Android, con una diferencia de 0.68 segundos a favor de iOS.

2. Variabilidad:

- a. **Android:** Los tiempos de ejecución varían entre 2.75 segundos y 3.23 segundos, mostrando una mayor diferencia entre el mejor y el peor caso. Esto puede ser indicativo de una mayor variabilidad en el rendimiento debido a factores como las especificaciones del dispositivo o la gestión de recursos, especialmente en dispositivos Android de gama media.
- b. **iOS:** Los tiempos se encuentran entre 1.98 segundos y 2.96 segundos, lo que muestra una menor variabilidad. Esto sugiere un comportamiento más consistente, lo cual es característico de los dispositivos Apple y su optimización integrada de hardware y software.

3. Análisis Comparativo:

- a. **Rendimiento Consistente en iOS:** iOS presenta un rendimiento más consistente, con una menor variación en los tiempos de ejecución. Esto puede deberse a la estrecha integración entre el hardware y el software, lo cual permite una mejor optimización del rendimiento.
- b. **Variabilidad en Android:** Android muestra tanto el mejor tiempo de ejecución (2.75 segundos) como el peor (3.23 segundos), lo cual refleja la diversidad en la gestión de recursos y el rendimiento, influenciada por la diversidad de hardware disponible en dispositivos Android.

Conclusión

En conclusión, los dispositivos iOS parecen ofrecer un rendimiento más rápido y consistente en comparación con los dispositivos Android, que muestran una mayor variabilidad en los tiempos de ejecución. Aunque Android tiene un rendimiento competitivo con un tiempo mínimo de 2.75 segundos, la variabilidad podría deberse a factores como la gestión de memoria o la carga de trabajo del dispositivo durante las pruebas. En promedio, iOS es más rápido con 2.25 segundos frente a 2.93 segundos de Android, lo cual sugiere una mejor optimización de la plataforma para la ejecución de la aplicación.

Pruebas de consumo de CPU y RAM

Además de las pruebas de velocidad de ejecución, se realizaron pruebas para evaluar el consumo de CPU y memoria RAM en dispositivos Android e iOS. Estas métricas son cruciales para entender cómo cada plataforma maneja los recursos del sistema, especialmente en aplicaciones de alto rendimiento.

El consumo de CPU y RAM proporciona una visión sobre la eficiencia en el uso de los recursos, afectando directamente la fluidez de la aplicación y la experiencia del usuario.

Estas pruebas fueron realizadas para comprender mejor cómo los dispositivos de gama media en cada plataforma responden a la carga de trabajo y cómo se comportan en términos de eficiencia en el uso de recursos. A continuación, se presentan los resultados de dichas pruebas, analizando la gestión de la CPU y memoria RAM para ambos dispositivos.

Android

A continuación, se mostrarán las pruebas de rendimiento realizadas a la aplicación hecha con el framework de Flutter en el sistema operativo de Android en términos de consumo de CPU y RAM.

Análisis de Reporte de Uso de CPU

El siguiente análisis presenta el uso del CPU durante la ejecución de una aplicación en un dispositivo Android. Este análisis incluye dos aspectos importantes: el consumo regular del CPU y el consumo más alto de la aplicación, además del consumo más alto de otros procesos del sistema. Estos datos permiten evaluar la eficiencia de la aplicación en términos de uso de los recursos del dispositivo. Debido a limitaciones técnicas con la aplicación desarrollada en React Native mediante Expo, las pruebas de rendimiento no pudieron realizarse directamente en esa aplicación. Por este motivo, se desarrolló una aplicación nativa para Android que sirvió como base para llevar a cabo estas pruebas y generar el reporte de

rendimiento.



Imagen 63: Gráfico de uso de CPU en Android Profiler para la aplicación Android

Consumo Regular del CPU

Como se puede observar en la imagen anterior, se muestra el consumo general del CPU durante el uso regular de la aplicación. En el gráfico se observa una utilización relativamente baja y estable del procesador, con un promedio cercano al 12%. Este resultado es un indicador positivo del rendimiento de la aplicación, ya que no genera una sobrecarga considerable en el sistema y permite que el dispositivo mantenga una buena experiencia de usuario. El bajo uso del CPU asegura que la aplicación pueda funcionar sin afectar el rendimiento de otras tareas concurrentes en el dispositivo, lo cual es fundamental para garantizar una experiencia de usuario fluida.

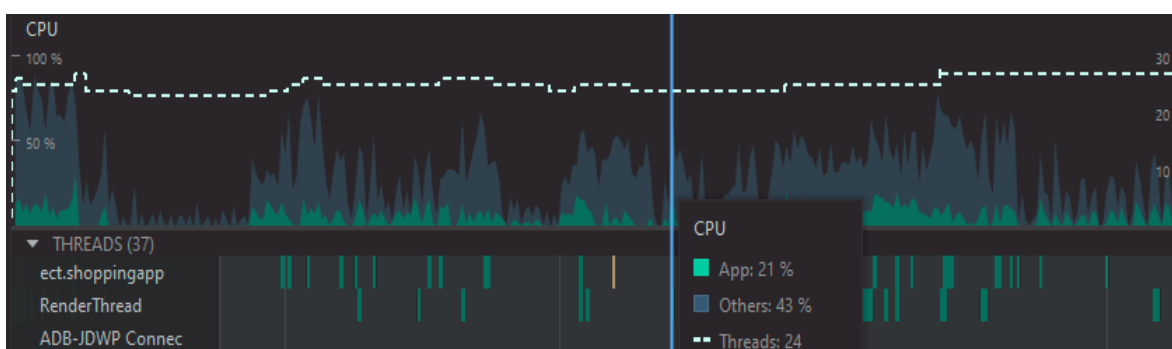


Imagen 64: Gráfico de Android Profiler mostrando el uso máximo del CPU y distribución de threads

Consumo Máximo del CPU

En la imagen anterior se presenta el punto de mayor consumo del CPU durante el uso de la aplicación. En esta parte del uso de la app, el consumo del CPU por parte

de la aplicación alcanza el 21%, mientras que otros procesos del sistema ocupan un 43%. Además, se observa un número de hilos activos de 24. Este incremento en el uso del CPU se debe al proceso de ejecutar peticiones a la API de películas y mostrarlas rápidamente en la interfaz. A pesar de estos picos, el consumo se mantiene en niveles aceptables, lo cual refleja una buena administración de recursos y optimización del código.

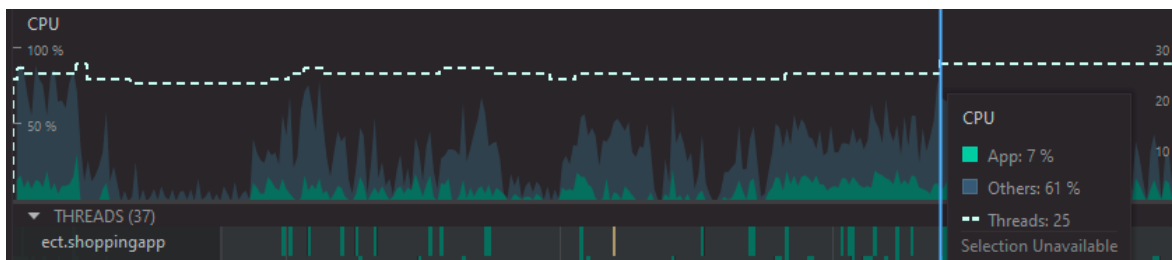


Imagen 65: Gráfico de Android Profiler destacando el uso máximo otros procesos

La última imagen muestra un caso en el que el consumo del CPU es menor por el consumo de la app y mayor al consumo de otros procesos que no pertenecen a la aplicación. En este escenario, la aplicación solo consume un 7% del CPU, mientras que otros procesos del sistema están utilizando un 61% del procesador, con un número de hilos activos de 25. Esto indica que, durante esta fase, la aplicación tiene un bajo impacto sobre el uso del procesador, mientras que otros servicios o procesos del sistema operativo acaparan los recursos. Este comportamiento es relevante para demostrar que la aplicación tiene un uso eficiente del CPU, permitiendo que otros elementos del sistema puedan ejecutarse sin causar una sobrecarga significativa.

Análisis de Reporte de Uso de Memoria.



Imagen 66: Gráfico de uso de memoria en Android Profiler mostrando un pico de 61.9 MB

En la imagen anterior se muestra la cantidad de memoria utilizada por la aplicación durante su ejecución. Se puede observar que el consumo de memoria se mantiene estable en alrededor de 61.9 MB. Este valor es indicativo de un uso eficiente de la memoria, ya que no se presentan picos significativos que puedan generar problemas de rendimiento o inestabilidad en el dispositivo. La estabilidad del consumo de memoria sugiere que la aplicación está gestionando correctamente sus recursos, evitando fugas de memoria o sobrecargas innecesarias.



Imagen 67: Gráfico de uso de memoria segmentada en Android Profiler

La imagen tal muestra un desglose detallado del consumo de memoria en la aplicación desarrollada con React Native, donde se pueden observar diferentes barras de colores que representan las distintas categorías de uso de la memoria. A continuación, se explica cada una de estas categorías:

1. **Barras Azules (Memoria Java):** Representan la memoria utilizada por los objetos Java. Dado que la aplicación está realizada con el framework de React Native, esto hace referencia a las estructuras de datos y los objetos creados tanto por el código JavaScript como por los componentes del framework. Esta es la parte principal del uso de memoria de la aplicación, que incluye las estructuras de datos y objetos creados durante la ejecución de la misma.
2. **Barras Verdes (Memoria Nativa):** Indican la memoria utilizada por el código nativo, incluyendo bibliotecas y componentes que no están escritos en Java. En React Native, esto incluye las bibliotecas nativas que se integran con el código JavaScript para acceder a funcionalidades específicas del dispositivo.

Este tipo de memoria es importante para operaciones que requieren un acceso directo a los recursos del hardware o para la integración de bibliotecas de terceros.

3. **Barras Grises (Memoria del Sistema):** Representan la memoria utilizada por el sistema operativo y otros procesos del sistema que están en ejecución en el dispositivo. Esta memoria no está directamente relacionada con la aplicación, pero es importante considerarla, ya que influye en la cantidad de recursos disponibles para la aplicación.

Conclusiones

En conclusión, el rendimiento de la aplicación muestra un uso eficiente tanto del CPU como de la memoria. La aplicación tiene un consumo moderado de CPU, alcanzando picos de 21%, lo que indica un uso eficaz sin sobrecargar el sistema. En cuanto a la memoria, mantiene un consumo estable de alrededor de 61.9 MB, sin grandes fluctuaciones ni fugas de memoria. Estos resultados sugieren que la aplicación realizada en React Native gestiona bien los recursos, garantizando una experiencia fluida y optimizada.

iOS

A continuación, se mostrarán las pruebas de rendimiento realizadas a la aplicación hecha con el framework de Flutter en el sistema operativo de iOS en términos de consumo de CPU y RAM

Análisis de Reporte de Uso de CPU

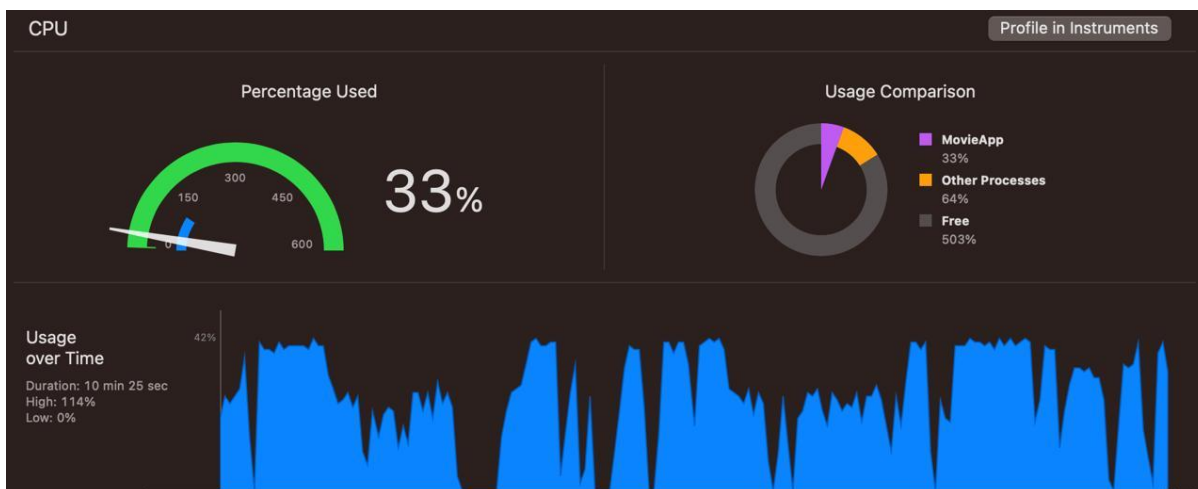


Imagen 68: Reporte de uso de CPU en Xcode Instruments para la aplicación de React Native en iOS

En la figura anterior se muestra un reporte de uso del CPU obtenido a través de una herramienta de monitoreo de rendimiento, como Xcode Instruments, que proporciona información detallada sobre el comportamiento del procesador durante la ejecución de una aplicación móvil.

La figura presenta varios elementos clave para analizar el rendimiento del CPU:

1. **Porcentaje Usado:** El uso del CPU en el momento capturado es del 33%, lo cual indica que aproximadamente un tercio de los recursos del procesador están siendo utilizados. Esto sugiere que la aplicación en ejecución, llamada "MovieApp", está realizando un uso moderado del CPU.
2. **Medidor de Uso del CPU:** El medidor semicircular representa visualmente el porcentaje de uso del CPU. La aguja se encuentra en la parte verde del medidor, lo cual indica que el uso del CPU está en un nivel manejable y no se está sobrecargando el procesador. Este tipo de representación gráfica es útil para identificar rápidamente la eficiencia en el uso de los recursos del dispositivo.
3. Comparación de Uso: En la parte derecha del reporte, una gráfica de dona compara el uso del CPU entre diferentes procesos:
 - a. MovieApp (la aplicación en prueba) está utilizando un 33% del CPU.

- b. Otros procesos en el sistema están utilizando un 64% del CPU.
 - c. Capacidad Libre del CPU es del 503%, lo cual se refiere a la disponibilidad de varios núcleos de procesamiento que no están siendo utilizados activamente.
4. Uso a lo Largo del Tiempo: La gráfica en la parte inferior muestra el uso del CPU durante un periodo de 10 minutos y 25 segundos.
- a. El valor máximo alcanzado durante este periodo fue del 114%, lo cual sugiere que en ciertos momentos el sistema estuvo utilizando más de un núcleo de procesamiento simultáneamente para manejar la carga.
 - b. El valor mínimo fue del 0%, lo cual indica que en algunos momentos específicos el procesador no tuvo carga alguna.
 - c. La gráfica azul representa los cambios en la carga del CPU a lo largo del tiempo, con picos que indican momentos de mayor demanda de procesamiento. Estos picos pueden estar relacionados con operaciones de la aplicación que requieren un mayor poder de cómputo, como la carga de datos desde la API o el renderizado de la interfaz gráfica.

Análisis del Reporte de Uso de RAM

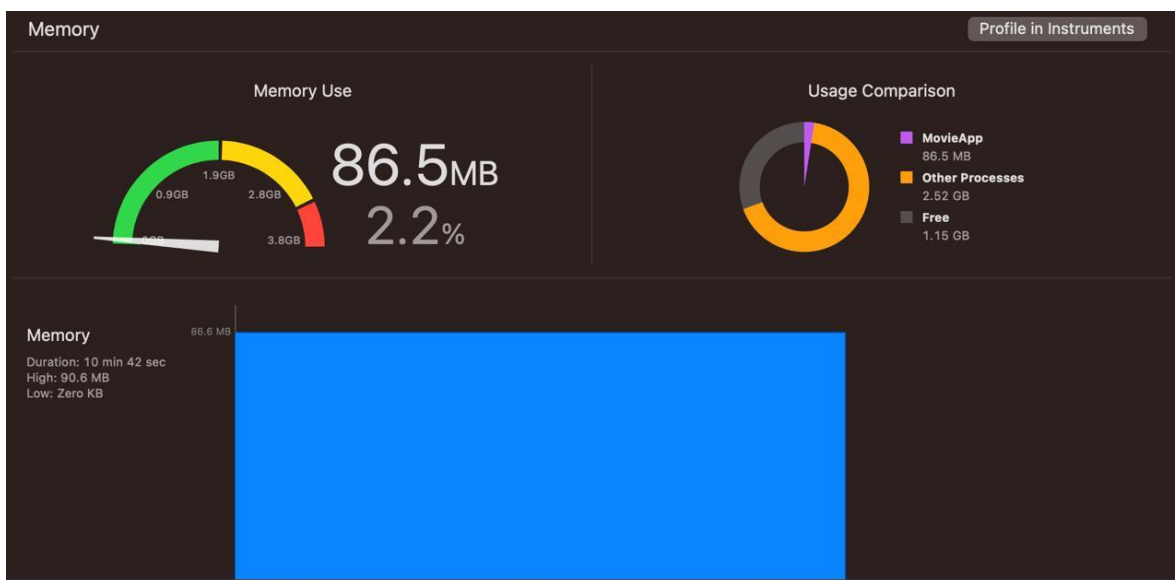


Imagen 69: Reporte de uso de memoria en Xcode Instruments para la aplicación de React Native en iOS

En la figura anterior se muestra un reporte de uso de memoria obtenido a través de una herramienta de monitoreo de rendimiento, como Xcode Instruments, que proporciona información detallada sobre el consumo de memoria durante la ejecución de una aplicación móvil.

La figura presenta varios elementos clave para analizar el uso de memoria:

1. **Uso de Memoria:** La cantidad de memoria utilizada por la aplicación en el momento capturado es de 86.5 MB, lo cual representa un 2.2% del total disponible. Este valor indica que la aplicación "MovieApp" tiene un consumo de memoria moderado, y no está ejerciendo una presión significativa sobre la memoria del sistema.
2. **Medidor de Uso de Memoria:** El medidor semicircular representa visualmente el uso de la memoria RAM. La aguja se encuentra en la zona verde, lo cual indica que el uso de memoria es bajo en comparación con la capacidad total de 3.8 GB. Este tipo de visualización es útil para determinar de manera rápida si hay problemas de consumo excesivo de memoria que pudieran afectar el rendimiento del dispositivo.
3. **Comparación del Uso de Memoria:** En la parte derecha del reporte, una gráfica de dona muestra la comparación del uso de memoria entre diferentes procesos:
 - a. MovieApp (la aplicación en prueba) está utilizando 86.5 MB de memoria.
 - b. Otros Procesos del sistema están utilizando 2.52 GB de memoria, lo cual representa la mayor parte del uso de la RAM en el dispositivo.
 - c. Memoria Libre es de 1.15 GB, lo que indica la cantidad disponible para uso adicional por parte de otros procesos o aplicaciones.
4. **Uso de Memoria a lo Largo del Tiempo:** La gráfica inferior muestra cómo ha variado el uso de memoria durante un periodo de 10 minutos y 42 segundos.
 - a. El valor máximo alcanzado por la aplicación fue de 90.6 MB, lo cual sigue siendo un valor moderado en términos de consumo de memoria.

- b. El valor mínimo fue de 0 KB, indicando que en algunos momentos la aplicación no estaba consumiendo memoria activa, probablemente debido a períodos de inactividad.
- c. La gráfica azul muestra que el uso de memoria se ha mantenido estable, sin fluctuaciones abruptas, lo cual sugiere que la aplicación "MovieApp" tiene un consumo de memoria controlado y no presenta problemas de fugas de memoria (memory leaks) que podrían afectar el rendimiento a lo largo del tiempo.

Conclusiones

En conclusión, el rendimiento de la app muestra un uso eficiente tanto del CPU como de la memoria. La aplicación tiene un consumo moderado de CPU (promedio de 33% con picos de 114%), lo que indica un uso eficaz de múltiples núcleos sin sobrecargar el sistema. En cuanto a la memoria, mantiene un consumo estable de 86.5 MB (alrededor del 2.2%), sin grandes fluctuaciones ni fugas de memoria.

Estos resultados sugieren que la app realizada en React Native gestiona bien los recursos, garantizando una experiencia fluida y optimizada, lo cual es esencial al comparar frameworks como Flutter y React Native en términos de eficiencia y rendimiento.

RESULTADOS

Análisis Comparativo de los Frameworks de Flutter y React Native

Este análisis comparativo evaluó tres aspectos clave de los frameworks Flutter y React Native: velocidad de ejecución, consumo de CPU y RAM, y facilidad de aprendizaje. Estos puntos resultaron fundamentales para determinar cuál de los frameworks ofrecía un rendimiento más óptimo y una experiencia de desarrollo más eficiente, permitiendo a Laptops Master elegir la opción más adecuada según los requisitos de sus proyectos.

Velocidad de Ejecución: Flutter vs React Native

Uno de los aspectos más relevantes en la evaluación del rendimiento de las aplicaciones móviles desarrolladas en Flutter y React Native fue la velocidad de ejecución, medida desde el momento en que la aplicación se abría hasta que todos los datos se cargaban por completo. Este análisis incluyó una comparación detallada de los tiempos de ejecución en dispositivos Android e iOS, utilizando datos obtenidos durante pruebas consecutivas en condiciones controladas.

Resultados de Flutter

En el caso de Flutter, los resultados de las pruebas en dispositivos Android e iOS arrojaron los siguientes promedios:

- **Android:** Promedio de 2.94 segundos
- **iOS:** Promedio de 2.87 segundos

Los datos mostraron que iOS presentó un rendimiento promedio ligeramente mejor que Android, con una diferencia de 0.07 segundos. Aunque ambos sistemas operativos ofrecieron tiempos similares, iOS destacó por su menor variabilidad, con tiempos de ejecución que oscilaron entre 2.72 y 3.01 segundos. En contraste, Android mostró una mayor discrepancia, registrando tiempos entre 2.30 segundos y 3.45 segundos.

Variabilidad en Android: Esta variabilidad fue atribuida a la diversidad del ecosistema Android, donde las especificaciones de hardware y la gestión de

recursos varían significativamente. Las pruebas utilizaron un dispositivo de gama media (Samsung Galaxy A52), lo cual pudo influir en la disparidad de los resultados.

Estabilidad en iOS: En iOS, los tiempos de ejecución fueron más consistentes debido a la integración del hardware y el software en los dispositivos de Apple. Esto se evidenció en los resultados obtenidos con un iPhone 11, que mostró una menor diferencia entre los mejores y peores tiempos registrados.

En conclusión, aunque Android registró un tiempo óptimo de 2.30 segundos, los resultados generales indicaron que la plataforma iOS ofreció una experiencia más estable y predecible, con menor variabilidad en el tiempo de carga de la aplicación.

Resultados de React Native

Para React Native, el rendimiento fue evaluado en condiciones similares y los resultados arrojaron los siguientes promedios:

- **Android:** Promedio de 2.93 segundos
- **iOS:** Promedio de 2.25 segundos

En este caso, la diferencia de rendimiento entre Android e iOS fue más significativa, siendo iOS 0.68 segundos más rápido en promedio que Android. Estos resultados reforzaron la tendencia observada en Flutter, donde iOS ofreció una mejor optimización en términos de velocidad de ejecución.

Rendimiento en Android: Los tiempos de ejecución en Android varían entre 2.75 segundos y 3.23 segundos, lo cual muestra una mayor variabilidad. Esto sugiere que, al igual que con Flutter, la gestión de recursos y las especificaciones del hardware juegan un papel importante en la ejecución de las aplicaciones React Native en Android.

Consistencia en iOS: En iOS, los tiempos fueron más consistentes, oscilando entre 1.98 y 2.96 segundos. La optimización del ecosistema cerrado de Apple permitió una mayor estabilidad en los tiempos de carga, resultando en una experiencia de usuario uniforme.

En conclusión, React Native demostró una ventaja significativa en dispositivos iOS, con un promedio de 2.25 segundos frente a los 2.93 segundos en Android. Esto reforzó la hipótesis de que la optimización del ecosistema cerrado de Apple permite un mejor rendimiento en comparación con la diversidad de hardware en Android.

Análisis Comparativo General

Comparando los resultados de ambos frameworks, se identificaron las siguientes tendencias comunes:

Mejor Rendimiento en iOS: Tanto Flutter como React Native mostraron tiempos de ejecución más rápidos y consistentes en dispositivos iOS, atribuibles a la integración eficiente del hardware y software en los dispositivos Apple.

Mayor Variabilidad en Android: En ambas tecnologías, los dispositivos Android presentaron una mayor variabilidad en los tiempos de ejecución debido a las diferencias en especificaciones de hardware y la carga de trabajo generada por otras aplicaciones en segundo plano.

Velocidad Promedio: React Native en iOS ofreció el mejor rendimiento promedio con 2.25 segundos, mientras que Flutter en Android alcanzó un promedio de 2.94 segundos, muy cercano al rendimiento de React Native en Android (2.93 segundos).

Conclusión

En conclusión, para aplicaciones que requirieron tiempos de carga rápidos y un rendimiento consistente, iOS demostró ser una mejor opción tanto para Flutter como para React Native. No obstante, la experiencia en Android, aunque presentó cierta variabilidad, resultó competitiva y podría mejorarse dependiendo de las especificaciones del hardware utilizado. Estos resultados subrayaron la importancia de considerar las plataformas objetivo al seleccionar un framework para el desarrollo de aplicaciones móviles, especialmente en términos de velocidad de ejecución y experiencia del usuario.

El análisis comparativo de Flutter y React Native evidenció que la plataforma iOS ofreció un rendimiento más consistente y estable en ambas tecnologías, lo cual constituyó una ventaja significativa para aplicaciones en las que la experiencia del

usuario depende de manera crítica de la velocidad de ejecución. En iOS, la estrecha integración entre hardware y software permitió aprovechar al máximo los recursos del sistema, traducándose en tiempos de carga más rápidos y uniformes. Esto resultó particularmente relevante para aplicaciones orientadas al consumidor final, que requieren respuestas inmediatas y una interfaz fluida.

En el caso de Android, aunque los tiempos de carga también fueron competitivos, la mayor variabilidad en los resultados reflejó las diferencias inherentes al ecosistema. Android abarca una amplia gama de dispositivos con diferentes capacidades de hardware, lo cual afectó directamente el rendimiento de las aplicaciones. Esta diversidad representó un desafío adicional para los desarrolladores que buscaron ofrecer una experiencia consistente en distintos modelos y marcas. Sin embargo, tanto Flutter como React Native demostraron ser opciones viables para el desarrollo en Android, especialmente cuando el código fue optimizado y se consideraron las especificaciones del dispositivo objetivo.

React Native en iOS obtuvo los mejores resultados promedio, posicionándose como una excelente opción para Laptops Master si se buscaba maximizar la eficiencia en dispositivos Apple. Sin embargo, Flutter también destacó, presentando tiempos de carga competitivos en ambas plataformas y sobresaliendo en aspectos como la personalización de la interfaz y la capacidad de reutilizar componentes. Esta capacidad de reutilización y el respaldo de Google hicieron de Flutter una opción atractiva para proyectos a largo plazo, donde la consistencia en el diseño y la escalabilidad de la aplicación fueron factores importantes.

La diferencia en la variabilidad de los tiempos de ejecución entre ambas plataformas también se interpretó desde la perspectiva de la experiencia del usuario. Los usuarios de dispositivos iOS tendieron a experimentar mayor uniformidad en el rendimiento, lo cual contribuyó a una percepción de calidad superior. En contraste, los usuarios de Android pudieron experimentar fluctuaciones en el rendimiento dependiendo de su dispositivo. Esto subrayó la importancia de realizar pruebas exhaustivas en una variedad de dispositivos Android antes del lanzamiento para garantizar un comportamiento aceptable de la aplicación en diferentes entornos.

El análisis mostró diferencias claras entre ambos frameworks en cada plataforma. React Native presentó el mejor rendimiento en dispositivos iOS, con un promedio de 2.25 segundos, mientras que Flutter tuvo un rendimiento competitivo con 2.87 segundos. En Android, Flutter y React Native mostraron promedios similares de 2.94 y 2.93 segundos respectivamente, pero con una mayor variabilidad en comparación con iOS.

Para Laptops Master, si el objetivo era desarrollar una aplicación con un rendimiento consistente y predecible, especialmente en dispositivos Apple, React Native fue la mejor opción, ya que ofreció tiempos de carga más rápidos y menor variabilidad. Por otro lado, si se buscó alcanzar un público más amplio en la plataforma Android, Flutter fue una excelente alternativa, aprovechando su flexibilidad en el soporte de componentes reutilizables, el respaldo de Google y tiempos de carga competitivos.

En resumen, la elección entre Flutter y React Native dependió de la prioridad de Laptops Master: estabilidad del rendimiento (iOS con React Native) o expansión a una base de usuarios más amplia (Android con Flutter). La estrategia de desarrollo consideró tanto las capacidades técnicas como las expectativas del usuario final y la visión a largo plazo del producto que se deseó desarrollar.

Consumo de CPU y RAM: Flutter vs React Native

En ese apartado se abordó el análisis del rendimiento de aplicaciones móviles, destacando que el rendimiento fue un factor clave que determinó la experiencia del usuario. Dicho análisis se centró en la comparación de dos de los frameworks más populares para el desarrollo móvil multiplataforma: Flutter y React Native, específicamente en términos de consumo de CPU y memoria (RAM). La comparación se realizó tanto en dispositivos Android como en iOS, con el propósito de proporcionar una evaluación detallada que ayudara a determinar cuál de estos frameworks resultó ser más adecuado para Laptops Master, una empresa que comenzaba en el ámbito del desarrollo móvil.

Consumo de CPU

Flutter en Android

El análisis del rendimiento en Android mostro que el método "FlutterView._render\$Method\$ffiNative" presento el mayor consumo de CPU (27.47%), ya que estuvo encargado del renderizado y otras operaciones visuales, mientras que "PlatformDispatcher._scheduleFrame" represento un 19.89%, mostrando la demanda de gestionar los frames de la interfaz de usuario. En total, la aplicación de Flutter en Android utilizó un 27% del CPU, con un promedio de 34 hilos activos. La gestión de recursos fue consistente, aunque se destaco el impacto significativo del renderizado en el rendimiento.

Flutter en iOS

En iOS, Flutter mostró un uso de CPU bastante eficiente, con un promedio de uso del 4% y un máximo del 205% durante los picos, lo que sugirió el uso de varios núcleos para tareas específicas. La mayor parte del tiempo, el CPU se mantuvo en un nivel bajo (representado en verde en el medidor), lo cual fue positivo, ya que indicó que la aplicación no ejerció una sobrecarga considerable en el sistema, garantizando así una experiencia fluida para el usuario.

React Native en Android

React Native en Android presentó un consumo de CPU regular del 12%, con picos que alcanzaron hasta el 21% cuando la aplicación realizó peticiones a la API y desplegó datos en la interfaz. Este comportamiento reflejó una buena optimización de los recursos, permitiendo que el sistema operara de manera eficiente incluso durante momentos de mayor carga. En situaciones de baja actividad, el consumo se redujo a un 7%, con la mayor parte del procesamiento siendo acaparado por otros procesos del sistema.

React Native en iOS

En iOS, React Native presentó un consumo de CPU del 33%, considerado moderado, con picos de hasta 114% que indicaron el uso de múltiples núcleos durante la ejecución de operaciones más intensivas, como la carga de datos desde la API. Durante la mayor parte del tiempo, el uso del CPU se mantuvo estable y en niveles bajos, lo que reflejó una buena gestión del rendimiento y la ausencia de sobrecargas.

Consumo de Memoria

Flutter en Android

En Android, el consumo de memoria por parte de Flutter fue de aproximadamente 194.19 MB (RSS), con una asignación estable de 14.24 MB. La mayoría del consumo provino del código Dart/Flutter (12.5 MB), mientras que los componentes nativos tuvieron un impacto reducido (0.74 MB). La memoria se gestionó de manera eficiente, sin picos ni fugas que comprometieran el rendimiento, y con una recolección de basura frecuente que evitó acumulaciones innecesarias.

Flutter en iOS

El uso de memoria en iOS por parte de Flutter alcanzó los 360 MB, representando un 9.4% del total disponible. Este consumo fue considerado moderado y eficiente, sin sobrecargar la memoria del dispositivo. Durante un periodo de 10 minutos, el uso de memoria se mantuvo estable, con ligeras fluctuaciones que alcanzaron un máximo de 418.4 MB. La gestión de la memoria en Flutter resultó efectiva, con una buena estabilidad que garantizó una experiencia de usuario satisfactoria.

React Native en Android

React Native en Android mostró un consumo de memoria estable de 61.9 MB, sin grandes fluctuaciones ni fugas. La memoria se dividió en tres categorías principales: Java (objetos JavaScript y estructuras de datos), memoria nativa (bibliotecas nativas y código no Java) y memoria del sistema. El uso de la memoria Java fue predominante, con una gestión eficiente que evitó la sobrecarga y aseguró una adecuada administración de los recursos.

React Native en iOS

En iOS, React Native presentó un consumo de memoria de 86.5 MB, lo que representó un 2.2% del total disponible. Durante la ejecución, el consumo de memoria se mantuvo estable, alcanzando un máximo de 90.6 MB y un mínimo de 0 KB durante los períodos de inactividad. La gestión de memoria de React Native resultó efectiva, sin indicios de fugas de memoria ni picos significativos que comprometieran la estabilidad del dispositivo.

Conclusión

En términos de consumo de CPU y memoria, tanto Flutter como React Native mostraron un rendimiento eficiente, cada uno con puntos fuertes y áreas de mejora dependiendo del sistema operativo y del perfil de uso.

Flutter destacó por su eficiencia en Android, con un consumo moderado de CPU que se concentró principalmente en el renderizado y la gestión de frames. Sin embargo, esto también implicó que las operaciones de la interfaz de usuario requirieron una cantidad significativa de recursos. En iOS, Flutter presentó un bajo consumo general de CPU, lo que aseguró que el dispositivo no se sobrecargara, además de mostrar una gestión de memoria eficiente tanto en Android como en iOS.

React Native, por su parte, presentó un menor consumo de CPU en Android y un rendimiento estable en iOS, con un promedio de 33% de uso del CPU y la capacidad de utilizar múltiples núcleos cuando fue necesario. El uso de memoria también resultó eficiente, alcanzando un consumo máximo de 86.5 MB en iOS, lo que garantizó la estabilidad del sistema durante la ejecución de la aplicación.

Debido a problemas al realizar las pruebas de rendimiento con la aplicación de React Native, se optó por construir una aplicación móvil nativa en Android para llevar a cabo las comparaciones correspondientes. Esta aplicación nativa proporcionó una referencia clara sobre el rendimiento en términos de CPU y memoria.

Para Laptops Master, que buscaba iniciarse en el desarrollo de aplicaciones móviles, la elección entre Flutter y React Native dependió del tipo de experiencia que se deseó ofrecer y de los recursos del dispositivo objetivo. Si se priorizó la eficiencia en la gestión de la interfaz gráfica y la estabilidad del consumo de memoria, Flutter fue una opción destacada, especialmente en dispositivos Android. Por otro lado, si se buscó un enfoque más equilibrado entre Android e iOS y una gestión eficiente del CPU con menor dependencia del renderizado, React Native ofreció ventajas significativas.

Facilidad de Aprendizaje de los Frameworks de Flutter y React Native

En este apartado, se analizó la facilidad de aprendizaje de los frameworks Flutter y React Native, considerando la experiencia obtenida durante el desarrollo de las aplicaciones móviles correspondientes al proyecto titulado "Análisis Comparativo del Rendimiento y Facilidad de Aprendizaje de los Frameworks de Flutter y React Native para el Desarrollo de Aplicaciones Móviles en Laptops Master". Se trabajó con ambos frameworks desde cero, lo que permitió experimentar los retos y beneficios que cada uno ofreció en cuanto a su facilidad de aprendizaje.

Facilidad de aprendizaje de Flutter

Desde el inicio del desarrollo con Flutter, la forma de construir las vistas con componentes resultó bastante intuitiva. La estructura del código y la reutilización de widgets hicieron que el proceso de creación fuera más eficiente. Una de las características más destacadas fue la facilidad para aplicar estilos y diseños a los widgets. La capacidad de crear widgets personalizados y reutilizables permitió que el desarrollo fuera más modular y organizado, facilitando la comprensión del código y mejorando la experiencia del usuario.

Uno de los retos iniciales fue el manejo de las llaves ('{}'). En algunas ocasiones, resultaba complicado identificar dónde terminaban ciertos widgets, especialmente en estructuras más complejas, lo que generaba errores difíciles de identificar. Para abordar este problema, se realizaron modificaciones en el entorno de desarrollo, específicamente en Visual Studio Code, mejorando la visibilidad y organización del código. Estas mejoras contribuyeron a que la experiencia de codificación fuera mucho más clara y eficiente.

Otro aspecto sumamente útil fue la diferencia entre los widgets estáticos y dinámicos. Comprender esta distinción ayudó a organizar mejor la aplicación y a aprovechar al máximo la reactividad de los widgets dinámicos. Además, trabajar con el lenguaje Dart fue accesible, ya que su sintaxis es fácil de aprender y comprender. Una vez asimilada la sintaxis, fue posible enfocarse completamente en el desarrollo de la aplicación en lugar de en la adaptación al lenguaje.

El proceso de instalación de Flutter y la configuración de las herramientas de desarrollo fue sencillo y estuvo bien documentado. La documentación oficial de Flutter resultó ser una herramienta invaluable, proporcionando ejemplos claros sobre cómo usar los diferentes widgets y sus propiedades. Estos ejemplos facilitaron la implementación de nuevas funcionalidades en la aplicación.

No obstante, uno de los mayores desafíos fue comprender el uso de los gestores de estado, fundamentales para escuchar cambios en la aplicación y reflejarlos en pantalla de manera ordenada y eficiente. Aunque inicialmente esto presentó dificultades, con tiempo y práctica se lograron implementar soluciones eficientes para la gestión del estado.

En cuanto a las pruebas de rendimiento, herramientas como FlutterDevTools y Xcode Instruments permitieron realizar análisis sin mayores complicaciones. Ambas herramientas se destacaron por ser fáciles de utilizar y por proporcionar información clara sobre los aspectos que podían ser mejorados en la aplicación.

Facilidad de aprendizaje de React Native

En cuanto a React Native, la experiencia fue diferente. Aunque la instalación inicial del framework resultó bastante sencilla, la configuración del entorno de desarrollo y la creación del proyecto presentaron mayores complicaciones en comparación con Flutter. Al iniciar un proyecto con React Native, se encontró que la configuración predeterminada utilizaba TypeScript, lo que obligó a buscar cómo iniciar un proyecto en JavaScript, el lenguaje inicialmente más familiar. Este proceso fue tedioso y consumió más tiempo del esperado, además de presentar problemas para ejecutar correctamente el proyecto.

Otro aspecto desafiante fue la gestión de estados y el uso del hook `useEffect`. A diferencia de Flutter, donde la reactividad es más intuitiva, React Native requirió mayor tiempo para entender el flujo de trabajo de los hooks y el manejo de efectos secundarios. La falta de experiencia previa con React complicó la comprensión de los hooks y el ciclo de vida de los componentes. Sin embargo, una vez superados estos desafíos, la experiencia mejoró considerablemente.

En cuanto a la construcción de la interfaz de usuario, React Native resultó accesible gracias a su similitud con HTML y CSS. Los conocimientos previos en estas tecnologías facilitaron la comprensión de la estructura de los componentes y la aplicación de estilos. No obstante, la necesidad de importar cada componente utilizado en la aplicación se percibió como un aspecto engorroso, especialmente en comparación con Flutter, donde este proceso es más directo y simplificado.

A pesar de las dificultades iniciales, React Native presentó aspectos positivos. El uso de JavaScript como lenguaje de desarrollo facilitó la implementación de lógica de negocio y la manipulación de datos de manera sencilla y eficiente. Sin embargo, se consideró que es necesario contar con conocimientos previos de React antes de trabajar con React Native, ya que conceptos como hooks, contexto y gestión de estados pueden resultar complejos para desarrolladores nuevos. Esto podría limitar el acceso y hacer que la curva de aprendizaje sea más pronunciada en comparación con Flutter.

Conclusión

En base a la experiencia obtenida, se concluyó que Flutter presentó una curva de aprendizaje más suave y accesible en comparación con React Native, especialmente para desarrolladores que se están iniciando en el desarrollo de aplicaciones móviles. La estructura clara de los widgets, la documentación completa y la facilidad para configurar el entorno de desarrollo hicieron que Flutter fuera una opción muy atractiva para quienes desean comenzar a construir aplicaciones sin mayores complicaciones. Además, Dart, como lenguaje de programación, resultó intuitivo y facilitó un desarrollo rápido.

Por otro lado, React Native requirió un conocimiento previo de React para aprovechar al máximo sus funcionalidades, lo que pudo representar una barrera inicial para algunos desarrolladores. Aunque React Native tiene la ventaja de utilizar JavaScript y resulta más familiar para quienes tienen experiencia en desarrollo web, la configuración y el manejo de estados hicieron que la curva de aprendizaje fuera más empinada. Sin embargo, una vez superados los obstáculos iniciales, React

Native ofreció gran flexibilidad y potencia, siendo beneficioso para desarrolladores con experiencia previa.

En resumen, si se buscaba una experiencia de aprendizaje más sencilla y un desarrollo rápido de aplicaciones, Flutter se presentó como la mejor opción. No obstante, para aquellos con una base sólida en desarrollo web y conocimientos de React, React Native pudo ser una herramienta igualmente poderosa, aunque con una curva de aprendizaje inicial más pronunciada. Dado que este análisis se realizó particularmente para la empresa Laptops Master, se determinó que la mejor opción es el framework Flutter, ya que su curva de aprendizaje más suave y rápida permitirá comenzar con el desarrollo móvil de manera más eficiente.

Conclusión General

El análisis comparativo entre los frameworks Flutter y React Native ha permitido determinar cuál de estas herramientas es más adecuada para el desarrollo de aplicaciones móviles multiplataforma, considerando los criterios de rendimiento y facilidad de aprendizaje establecidos. Con base en los datos obtenidos, se concluye que **Flutter representa la mejor opción para el inicio del desarrollo móvil en Laptops Master**, debido a su rendimiento competitivo, facilidad de aprendizaje y ecosistema robusto.

Rendimiento General

En términos de velocidad de ejecución, ambos frameworks ofrecen tiempos competitivos en dispositivos Android e iOS, con ligeras ventajas para React Native en iOS. Sin embargo, Flutter muestra una gestión más eficiente de los recursos, especialmente en Android, donde la diversidad del hardware y las especificaciones representan un desafío significativo. En pruebas de consumo de CPU y RAM, Flutter demostró una mayor consistencia en la gestión de recursos, garantizando una experiencia fluida y estable para los usuarios.

Facilidad de Aprendizaje

Flutter destaca por ofrecer una curva de aprendizaje más accesible, gracias a su arquitectura basada en widgets, documentación completa y herramientas de desarrollo robustas como FlutterDevTools. A diferencia de React Native, que requiere conocimientos previos de React y manejo avanzado de hooks, Flutter permite a los desarrolladores adaptarse rápidamente al entorno y concentrarse en la implementación de funcionalidades sin enfrentar barreras técnicas significativas. La facilidad en la configuración del entorno de desarrollo también contribuye a su preferencia, reduciendo los tiempos iniciales de adopción.

Alineación con los Objetivos de Laptops Master

De acuerdo con los criterios de éxito definidos, Flutter cumple con la hipótesis planteada al demostrar un rendimiento eficiente y una mayor facilidad de aprendizaje en comparación con React Native. Esto lo convierte en la herramienta

ideal para iniciar el desarrollo móvil, ya que permite maximizar la productividad del equipo y garantizar la calidad de las aplicaciones. Además, la capacidad de Flutter para ofrecer consistencia en diseño y la reutilización de componentes es especialmente valiosa para proyectos a largo plazo, donde la escalabilidad y el mantenimiento son factores clave.

Conclusión Final

Flutter se posiciona como la opción más equilibrada y eficiente para el desarrollo de aplicaciones móviles multiplataforma en Laptops Master. Su rendimiento competitivo, facilidad de uso y ecosistema robusto permiten cumplir con los objetivos planteados, ofreciendo una solución que garantiza un desarrollo ágil y eficiente. React Native, si bien es una herramienta poderosa y flexible, presenta una curva de aprendizaje más pronunciada y mayores complicaciones en la configuración inicial, lo que lo convierte en una opción más adecuada para equipos con experiencia previa en desarrollo web. En resumen, Flutter se establece como la mejor elección para los objetivos actuales de Laptops Master, asegurando un proceso de desarrollo eficiente y una experiencia de usuario óptima en las aplicaciones móviles.

Referencias

- [1] A. B. Alonso, I. F. Artime, M. Á. Rodríguez y R. B. García, «Dispositivos móviles,» *E.P.S.I.G : Ingeniería de Telecomunicación*, vol. 12, 2011.
- [2] M. L. S. G. Esteban Vázquez-Cano, *Dispositivos digitales móviles en Educación: El aprendizaje ubicuo*, Narcea Ediciones, 2015.
- [3] Noelia, «Google Play vs App Store, ¿Cuáles son sus diferencias?,» *Actualizatec*.
- [4] M. L. R. Quisaguano Collaguazo, M. S. Pallasco Venegas, A. A. Andaluz Guerrero, A. M. N. Martínez Freire y M. S. H. Corrales Beltrán, «Desarrollo Híbrido con Flutter,» *Ciencia Latina Revista Científica Multidisciplinar*, vol. 6, nº 4, pp. 4594-4609, 2022.
- [5] W. Z. Kamil Wasilewski, «A Comparison of Java, Flutter and Kotlin/Native Technologies for Sensor Data-Driven Applications,» *Sensors*, vol. 21, nº 10, p. 3324, 2021.
- [6] J. T. Gironés, *El gran libro de Android*, Alpha Editorial, 2019.
- [7] Everyys, «Everyys,» 10 Junio 2019. [En línea]. Available: <https://www.everyys.com/articles/news/how-huawei-ark-compiler-wants-revolutionize-android-be-more-ios>. [Último acceso: 13 Diciembre 2024].
- [8] Y. Zhang, «Design on Japanese Listening Mobile Learning System based on iOS Platform,» *Atlantis Press*, Vols. %1 de %2In 2017 2nd International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2017, pp. 1026-1030, 2017.
- [9] P. d. J. Morales Guevara, C. Gabriel y Á. Eduardo, *Diseño de un modelo de aplicación móvil en entorno Android*, Editorial Academica Espanola, 2014.
- [10] K. S. R. A. L. A. V. J. Z. J. T. S. J. M. S. A. L. S. & R. J. Chad R. Gordon, «Digital mobile technology facilitates HIPAA-sensitive perioperative messaging, improves physician-patient communication, and streamlines patient care,» *Patient Safety in Surgery*, vol. 9, nº 21, 2015.
- [11] N. G. P. T. P. P. Lisandro Delía, «Un Análisis Experimental de Tipo de Aplicaciones para Dispositivos Móviles,» de *En XVIII Congreso Argentino de Ciencias de la Computación*, 2013.
- [12] C. B. Reynoso, «Introducción a la Arquitectura de Software,» *Universidad de Buenos Aires*, vol. 33, 2004.
- [13] J. A. D. F. A. A. P. C. Andy Hernández Paez, «Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D,» *Revista de I+ D tecnológico*, vol. 14, nº 1, pp. 54-56.
- [14] Y. D. G. Yenisleidy Fernández Romero, «Patrón Modelo-Vista-Controlador,» *Telem@tica*, vol. 11, nº 1, pp. 47-57, 2012.

- [15] N. L. V. R. Oscar Danilo Gavilánez Alvarez, «Análisis comparativo de Patrones de Diseño de Software,» *Polo del Conocimiento: Revista científico-profesional*, vol. 7, nº 7, pp. 2146-2165, 2022.
- [16] P. L. N. Ruiz, «Desarrollo de una aplicacion android para el control de un dispositivo movil usando arquitectura cliente-servidor,» 2017.
- [17] C. Á. Caules, «ArquitecturaJava,» 22 Marzo 2018. [En línea]. Available: <https://www.arquitecturajava.com/arquitecturas-rest-y-sus-niveles/>. [Último acceso: 7 Diciembre 2024].
- [18] C. Á. Caules, «ArquitecturaJava,» 07 Marzo 2023. [En línea]. Available: <https://www.arquitecturajava.com/que-es-rest/>. [Último acceso: 07 Diciembre 2024].
- [19] O. I. T. Buriticá, «Relaciones de aprendizaje significativo entre dos paradigmas de programacion a partir de dos lenguajes de programacion,» *Tecnura*, vol. 18, nº 41, pp. 91-102, 2014.
- [20] A. R. Galván, «ANÁLISIS DE LOS LENGUAJES DE PROGRAMACIÓN».
- [21] E. S. S. Á. P. R. Roberto Rodríguez Echeverría, «Programación orientada a objetos,» 2004.
- [22] G. R. Rivadera, «La Programación Funcional: Un Poderoso Paradigma,» *Cuadernos de la Facultad*, nº 3, pp. 63-77, 2008.
- [23] J. M. S. H. Ana Pradera Gómez, «Programacion Declarativa: guía de la asignatura,» 2022.
- [24] A. M. G. Fernando López Ostenero, Teoría de los lenguajes de programación, Editorial Universitaria Ramon Areces, 2014.
- [25] J. J. T. A. C. T. Ricardo Timarán Pereira, «Programación multiparadigma como estrategia de aprendizaje de los lenguajes de programación en ingeniería de sistemas [Multiparadigm programming as a strategy for learning programming languages in Systems Engineering],» *Ventana Informática*, nº 27, 2012.
- [26] A. L. S. d. P. Rafael Henrique Borges, «EXPLORANDO DESTINOS TURÍSTICOS COM TECNOLOGIA BEACON: UM APLICATIVO DE GUIA TURÍSTICO,» 2023.
- [27] R. P. d. M. F. Filipe Del Nero Grillo, «Aprendendo JavaScript,» 2008.
- [28] R. I. B. RIVERA, «ANÁLISIS COMPARATIVO ENTRE FRAMEWORKS, PARA EL DESARROLLO DE APLICACIONES MÓVILES MULTIPLATAFORMAS,» 2021.
- [29] A. B. U. ., N. S. Ekrem Gülcüoğlu, «Comparison of Flutter and React Native Platforms,» vol. 12, nº 2, pp. 129-143, 2021.

- [30] B. C. d. Freitas, «Flutter e react native: uma análise comparativa entre dois frameworks de desenvolvimento mobile multiplataforma,» 2022.
- [31] M. Á. S. Maza, Javascript, Innovación Y Cualificación, 2012.
- [32] W. Wenhao, React Native vs Flutter, cross-platform mobile application frameworks, 2018.
- [33] L. S. L. G. Victor Hugo Guachimbosa Villalba, «Sistema multiplataforma de código abierto para resolución de problemas de programación lineal utilizando el framework flutter,» Universidad Técnica de Ambato. Facultad de Ingeniería en Sistemas, Electrónica e Industrial. Carrera de Software, 2024.
- [34] E. Windmill, Flutter in Action, Simon and Schuste, 2020.
- [35] M. T. B. L. G. P. M. V. E. Elizabeth Suescún Monsalve, «Hacia un Modelo de Gobierno de APIs, Mapeo Sistemático de la Literatura,» *Perspectivas*, vol. 6, nº 2, 2024.
- [36] E. J. S. C. Uriel Paredes, «Métricas de APIs: Catálogo y Herramienta OMA,» *Informes Científicos Técnicos UNPA*, vol. 15, nº 1, pp. 123-143, 2023.
- [37] Y. P.-D. L. T.-P. M. D. D.-D. C. Y.-M. Sandra Verona-Marcos, «Pruebas de rendimiento a componentes de software utilizando programación orientada a aspectos,» vol. 37, nº 3, pp. 278-285, 2016.
- [38] Google, «Androir Developers,» 29 Ocutbre 2024. [En línea]. Available: <https://developer.android.com/studio/profile>. [Último acceso: 3 Diciembre 2024].
- [39] Apple, «Apple Developer,» [En línea]. Available: <https://developer.apple.com/documentation/xcode/improving-your-app-s-performance>. [Último acceso: 3 Diciembre 2024].
- [40] Google, «Flutter,» [En línea]. Available: <https://docs.flutter.dev/tools/devtools>. [Último acceso: 3 Diciembre 2024].
- [41] C. P. Libardo Pantoja, «Evaluando la Facilidad de Aprendizaje de Frameworks mvc en el Desarrollo de Aplicaciones Web,» *Publicaciones E Investigación*, vol. 10, pp. 129-142, 2016.
- [42] E. Velasteguí López y G. Barona López, «El avance en la tecnología móvil y su impacto en la sociedad,» *ed*, vol. 2, nº 4, pp. 5-19, 2019.
- [43] H. M. Mcluhan, El papel de los celulares en la sociedad:¿ alejamiento o acercamiento?.
- [44] W. Wu, «React Native vs Flutter, cross-platform mobile application,» 2018.
- [45] X. G. Perea, «Diseño e implementación de un extractor,» Universitat Politècnica de València, 2021.

[46] D. Deutsch, «A quick introduction to clean architecture,» *freeCodeCamp*, 2018.

[47] A. Casero, «La arquitectura del cliente y servidor,» 16 Mayo 2024. [En línea]. Available: <https://keepcoding.io/blog/arquitectura-del-cliente-y-servidor/>. [Último acceso: 2 Diciembre 2024].

Anexos

Proyecto de Flutter:

<https://github.com/Hugo-DevM/MobileCinemaFlutter.git>

Proyecto de React Native:

<https://github.com/Hugo-DevM/MobileCinemaReactNative.git>

Diagramas en StarUML:

<https://github.com/Hugo-DevM/DiagramasStarUMLResidenciaTec.git>