Universidad Autónoma del Estado de México
Centro Universitario UAEM Zumpango

Ingeniería en computación

Desarrollo de Aplicaciones para Sistema Operativo Android con Flutter, React Native y Kotlin

ENSAYO

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

Gerardo Rivero Gómez

DIRECTOR DEL ENSAYO:
Dr. Asdrúbal López Chau

Enero, 2023

Resumen

En la actualidad existen varias tecnologías que permiten desarrollar aplicaciones móviles, esto resulta ser un problema para las personas que quieren iniciar a desarrollar aplicaciones móviles; Surgiendo las siguientes preguntas: ¿Cómo puede un programador crear aplicaciones?, ¿Qué tecnologías existen?, ¿Qué plataformas soportan?

En este documento, se realizó un breve análisis de las tecnologías móviles que usan los desarrolladores en el ámbito laboral o personal, interpretando la información de 2 páginas que tienen gran influencia en programadores, estas páginas son: stack Overflow y JetBrains. Posteriormente, se eligen 3 tecnologías más atractivas para desarrollar aplicaciones móviles.

Se elabora un análisis de requerimientos de hardware de las 3 tecnologías utilizando una laptop con: procesador core i5 de 2 núcleo y 4 hilos, 12GB de Memoria RAM, un SSD de 512GB. El análisis se elabora mediante: la medición del tiempo de carga y memoria RAM que ocupa el editor de código al abrirse; se observa el tiempo y almacenamiento que se requiere al elaborar una plantilla para crear la aplicación; se examina: el almacenamiento de la aplicación en el equipo de cómputo, el consumo de memoria RAM de la laptop, el tiempo de ejecución de la aplicación, y el almacenamiento en memoria ROM que ocupa en el dispositivo móvil.

Para el primer proceso de carga requiere mayor trabajo por la creación de archivos. Por este motivo, cambia los valores después de la primera ejecución de la aplicación, y se requiere de otro análisis de ejecución de aplicación. A diferencia del proceso anterior, se elimina el verificar el almacenamiento de la aplicación tanto en el equipo de cómputo como en el teléfono; aumentando dos procesos que brindan las tecnologías multiplataforma que son: hot-reload y hot-restart, estos procesos no sirven para observar los cambios en la aplicación que se hacen al momento que se está creando la aplicación.

Se procede a desarrollar una aplicación que permite evaluar las cualidades que ofrece cada tecnología. La aplicación a desarrollar fue una calculadora, evaluando las siguientes características: la sintaxis del lenguaje, que comodidad tiene el lenguaje para el programador, se observa que tecnología requiere más caracteres con el número de líneas de código, el almacenamiento que requiere la aplicación en el teléfono como en la computadora.

Por último, se crean conclusiones para evaluar y comparar las ventajas o desventajas que brinda cada tecnología, en las revisiones mencionadas con anterioridad.

Abstract

Currently there are several technologies that allow you to develop mobile applications. This turns out to be a problem for people who want to start developing mobile apps. The following questions come up: How can a programmer create applications?, What technologies exist?, What platforms do they support those apps?

A brief analysis of the mobile technologies used by developers in the workplace or personal environment was made. Interpreting the information of 2 pages that have great influence on programmers. These pages are: stack Overflow and JetBrains. Subsequently. Furthermore, three of the most attractive technologies are chosen to develop mobile applications.

Thus, an analysis of hardware requirements of the 3 technologies was elaborated, by using a laptop with the following characteristics: core i5 processor with 2 cores and 4 threads , 12Gb of RAM, 512 GB SSD. The analysis was elaborated by measuring the load time and RAM that the code editor occupies when it opened. It was seen the time and storage required when developing a template to create the application. At the time of running the application was examined: the storage of the application on the computer equipment, the laptop's RAM consumption, application runtime, and ROM storage that is occupied on the mobile device.

For the first loading process is required more work to create files. For this reason, the values are changed after the first execution of the application, and another application execution analysis is required. Unlike the previous process, it is eliminated by checking the storage of the application on both the computer equipment and the mobile phone; increasing two processes that are provided cross-platform technologies that are: hot-reload and hot-restart. These processes do not make any changes to the application at the time of execution, Unless the code that is written in the files or the dependencies of it.

It is proceeded to develop an application that allows to evaluate the qualities offered by each technology. The application developed was a calculator; evaluating the following characteristics: language syntax, the convenience that the language has to be manipulated by the programmer. It is observed that technology requires more characters with the number of lines of code and the storage is required by the app on the phone as well as on the computer.

Finally, conclusions were created to evaluate and compare the advantages or disadvantages offered by each technology in the reviews mentioned previously.

Contenido

Lista de figuras

Lista de tablas

Lista de códigos

Capítulo 1

Introducción

Al inicio del año 2000 las tareas que se ejecutaban en las aplicaciones informáticas eran de alcances mínimos, es decir, eran "tareas muy sencillas". Ejemplos de aplicaciones de software sencillas de esa época son agendas simples, reproductores de sonidos y editores de texto. Conforme fue avanzando la tecnología, las aplicaciones requirieron de realizar tareas más complejas, por lo que la programación ha implementado procedimientos con mayor estructura de códigos organizados. Por ejemplo, anteriormente se buscaba la resolución de un problema usando un estilo de programación estructurado. Actualmente esto ya no es conveniente.

Las aplicaciones para dispositivos móviles nativas, al ser populares para el momento que se lanzó el iPhone, integraba la Appstore que permitía comprar aplicaciones. Otro de los factores que impulsaron a que las aplicaciones fueran populares fue Facebook, ya que era una novedad en esos tiempos; en general, si el teléfono no permitía usar esta aplicación , el dispositivo era poco atractivo para el usuario promedio de esa época.

En la segunda década de los 2000 las tecnologías para el desarrollo de las aplicaciones fueron mejorando, de tal manera que surge el desarrollo híbrido, donde las personas que utilizaban algunas tecnologías para crear páginas Web, ya podía usar esas aplicaciones en el teléfono. Pero tenía que modificar partes de su código para que se mostrara en un Smartphone. El marketing era atractivo, aunque en realidad se evidenciaron ciertos

errores en las interfaces gráficas de las aplicaciones. Cabe mencionar que no todas las tecnologías que se implementaban en web eran soportadas por los teléfonos. [14]

Además surgen otro tipo de tecnologías que permite desarrollar aplicaciones donde no es forzoso que domines un lenguaje de programación ya que permiten desarrollar aplicaciones de una manera más fácil. Esta categoría es desarrollo de terceros .

A finales de la segunda década de los 2000 surge una evolución de las tecnologías híbridas, el desarrollo multiplataforma. La principal característica de esta tecnología busca hacer una compilación nativa, permitiendo agilizar la comunicación entre el hardware y el código que crea el programador en el teléfono. En esta tecnología se puede crear aplicaciones en las 4 plataformas de desarrollo: de smartphone, tabletas, Web y escritorio.

En actualidad, la era de la tecnología ha evolucionado con gran magnitud lo que ha permitido la interacción con aplicaciones móviles, son un cúmulo de aplicaciones que incluso a los programadores suelen confundir al pretender crear nuevas aplicaciones o manipulación de la información tecnológica [14]; derivado de ellos surge la inquietud ¿Cómo puede un programador crear aplicaciones con diferentes tecnologías?

Para dicho análisis es importante conocer características de cada programa y así poder categorizarlas, permitiendo enfocar las necesidades del desarrollo. Existen 3 categorías principales en la parte de desarrollo móvil: la primera es desarrollo nativo donde el desempeño de las aplicaciones tiene un excelente desempeño; la segunda permite desarrollar una aplicación que se ejecute en un sistema operativo, pero no de forma nativa, como por ejemplo app inventor o Swift Playgrounds; la tercera es desarrollo multiplataforma, en el que se permite desarrollar una aplicación para más de una plataforma (sistema operativo) con base en un código. En la tabla 1.1 se mencionan algunos frameworks de esta última categoría.

En la mayoría de los casos es muy conveniente que una aplicación se encuentre en más de una tienda, ya que las personas puedan instalarlas en dispositivos con distinto sistema operativo. Un ejemplo muy claro es la aplicación Netflix, que nos permite acceder desde la mayoría de dispositivos que se conectan a Internet. Esta accesibilidad

TABLA 1.1: Tecnologías de Desarrollo

Nombre	Sistemas Operativos Soportados
Swift, Objective C	iOS de manera nativa
Java,Kotlin	Android de forma nativa
Xamarin	Android e iOS,Desktop(Windows)
Flutter	Android, iOS, Web y Desktop
Ionic	Android, iOS, y Web
App inventor	Android
React Native	Android, iOS y Web
Swift Playgrounds	iOS
NativeScript	Android e iOS
Kotlin Multiplatform Mobile (KMM)	Android y iOS

permite que la aplicación sea más usada, dando mayor comodidad para el usuario final. Otro ejemplo es la aplicación de Facebook, donde permite acceder desde cualquier teléfono, Tablet o computadora. Con los anteriores ejemplos se demuestra que al poder utilizarla en más de una plataforma, ayuda a que la aplicación tenga mayor éxito.

Existen aplicaciones que no requieren hacer tareas complejas, simplemente es necesario una interfaz muy amigable y simple para que el usuario; Como por ejemplo: analizar un conjunto de datos, almacenamientos de datos o mostrar información. El desarrollo de aplicaciones multiplataforma nos ayuda a reducir costos y tiempo para desarrollar una aplicación, ya que la mayoría de tecnologías que permiten desarrollo multiplataforma comparten el tema de reutilizar código. Permitiendo que unas bases de código se pueda implementar en diferentes plataformas.

La intensión de este ensayo es mostrar el panorama actual de la programación móvil, enfocándose en las características que brindan las tecnologías Andorid, Flutter y React Native, facilitando selección de la tecnología que sea más conveniente.

En particular, en este documento se realiza lo siguiente:

a) Dar a conocer los requisitos de cada framework, y comprobar si el entorno de desarrollo integrado (IDE) ayuda en la creación de una aplicación.

b) Comprobar si existe una diferencia visible del rendimiento de cada una de las tecnologías.

c) Mostrar la complejidad para crear una aplicación: cuál necesita más código, lo amigable que es utilizar las funciones de cada lenguaje para resolver una tarea, que tan accesible es la documentación para resolver los problemas que se presenten.

Capítulo 2

Desarrollo

2.1 React Native y Flutter en Stack Overflow

Jetbrains y Stack Overflow aplican anualmente una encuesta sobre las tecnologías más usadas por desarrolladores a nivel mundial. En este documento, se mostrarán los resultados de las encuestas de los años 2021 y 2020, orientadas a desarrollo de aplicaciones de dispositivos móviles.

Antes de dar a conocer los resultados de las encuestas, se muestra información sobre Jetbrains y Stack Overflow.

La empresa Jetbrains [20] es una desarrolladora de herramientas software a nivel mundial. Estas herramientas brindan la posibilidad de trabajar más rápido, automatizar tareas repetitivas dejando al programador enfocarse en el diseño de código y resolver problemas. Jetbrains fue fundada en el año 2000 por Sergey Dmitriev, Vlentin Kipyatkov y Eugene Belyaev.

Las herramientas creadas en Jetbrains son desarrolladas usando su propio software. Cada herramienta optimiza ciertas tareas, y se han creado varios productos para distintas necesidades.

Algunas de las empresas a las que Jetbrains les provee de software son: Ubisoft, Citibank, Google, P & G, Netflix, Hp, Samsung, y a la NASA. Los productos más

destacados son Intelliji IDEA, kotlin, WebStorm, PhpStorm.

Jetbrains y Stack Overflow aplican anualmente una encuesta sobre las tecnologías más usadas por desarrolladores a nivel mundial. En este documento, se mostrarán los resultados de las encuestas de los años 2021 y 2020, orientadas a desarrollo de aplicaciones de dispositivos móviles.

Antes de dar a conocer los resultados de las encuestas, se muestra información sobre Jetbrains y Stack Overflow.

La empresa Jetbrains [20] es una desarrolladora de herramientas software a nivel mundial. Estas herramientas brindan la posibilidad de trabajar más rápido, automatizar tareas repetitivas dejando al programador enfocarse en el diseño de código y resolver problemas. Jetbrains fue fundada en el año 2000 por Sergey Dmitriev, Vlentin Kipyatkov y Eugene Belyaev. Por otra parte, Stack Overflow es un sitio para programadores, donde se publican preguntas y respuestas para resolver alguna duda de programación en cualquiera lenguaje de programación; el sitio tiene tanta influencia que las organizaciones a nivel mundial la consideran para tomar decisiones en versiones posteriores de sus productos. En la comunidad de Stack Overflow, existen miembros que participan sin fines de lucro, hasta miembros empresariales con sus respectivos beneficios; este sitio cuenta con la comunidad más grande a nivel mundial. Una gran cantidad de programadores a nivel mundial usan este sitio diariamente para resolver problemas de programación diariamente.

2.1.1 Datos de Stack Overflow en 2020

Primero, se observan los datos obtenidos en la encuesta de Stack Overflow en el año 2020 [24], donde participan más de 60,000 desarrolladores. Los resultados obtenidos son los siguientes:

- El 19.2% de las personas entrevistadas realiza desarrollo de aplicaciones móviles.
- Los lenguajes más famoso que se usan en desarrollo móvil son: Kotilin con 7.8%, continua Swift con 5.9% Objective-C con 4.1%, por ultimo dart con 4.0%.

- Se realiza una clasificación de 40,314 desarrolladores para abordar información sobre Frameworks, bibliotecas y herramientas. Los resultandos obtenidos son: el 11.5% utiliza React Native, el 7.2% emplea Flutter, el 6% implementa Cordova y el 5.8% crea en Xamarin.

- De 53,843 desarrolladores respondieron a la pregunta ¿En qué plataforma han trabajado en este año? Android obtiene 26.2% , e iOS consigue 12%.

- Se realizó un apartado de clasificación donde se evalúa, si los desarrolladores tienen interés de continuar trabajando con la tecnología actual. obteniendo los siguientes resultados:

  - El lenguaje con mayor reputación es Kotlin, ya que el 62.9% le agrada.
  - El lenguaje Dart ocupa un 62.1%.
  - Swift obtiene un 59.5%
  - por último, Objective-C con 23.4%.

- Los lenguajes de desarrollo móvil con mayor salario son: Objective-C con sueldos de hasta 64,000 USD, le sigue swift con denominaciones de 58,000 USD, kotlin con 54,000 USD y por último Dart con 37,000 USD.

- El 19.1% de desarrolladores busca trabajo en el área de desarrollo móvil.

- La tecnología que tiene mayor desarrollo en Android es Java y por el lado de iOS es Swift.

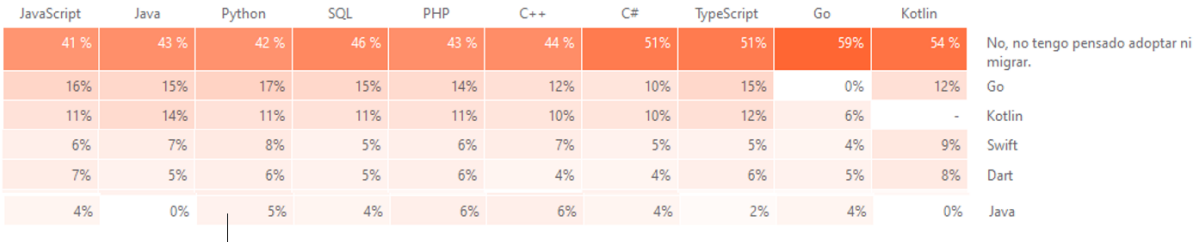
- El salario de un desarrollador móvil es de aproximadamente 43,000 USD a nivel mundial.

2.1.2 Datos de JetBrains en 2020

A continuación se muestran los datos obtenidos de la página de JetBrains en el año 2020 [18] en el que se encuestaron a 19,696 desarrolladores.

- Las personas que han utilizado el lenguaje en los últimos 12 meses: el 17% ha utilizado Kotlin y planean migrarlo 10% , el 9% ha manejado Dart y el 5% quiere migrar, Swift ocupa el 9% y el 3% lo quieren migrar, Objective-C lo ocupa el 4% y 1% quiere migrar.

  En la Figura: 2.1 se muestran datos de las tecnologías Java y Kotlin, señalando los lenguajes a los cuales se desean migrar.



JavaScript	Java	Python	SQL	PHP	C++	C#	TypeScript	Go	Kotlin	
41 %	43 %	42 %	46 %	43 %	44 %	51%	51%	59%	54 %	No, no tengo pensado adoptar ni migrar.
16%	15%	17%	15%	14%	12%	10%	15%	0%	12%	Go
11%	14%	11%	11%	11%	10%	10%	12%	6%	-	Kotlin
6%	7%	8%	5%	6%	7%	5%	5%	4%	9%	Swift
7%	5%	6%	5%	6%	4%	4%	6%	5%	8%	Dart
4%	0%	5%	4%	6%	6%	4%	2%	4%	0%	Java

Figura 2.1: JetBrains tecnologías móviles 2020

- La gente que utiliza Java; el 14% quiere migrar a Kotlin; el 7% quiere pasar a Swift; el 5% quiere pasar a Dart.

- La gente que utiliza Kotlin el 9% quiere pasar a Swift; el 8% quiere pasar a Dart, da la casualidad de que nadie de Kotlin quiere pasarse a Java.

- El 32% de las personas crea desarrollo móvil.

- Los sistemas operativos que tienen mayor desarrollo en aplicaciones es Android e iOS, el 41% crea aplicaciones para Android, el 11% par iOS y el 2% para otro sistema operativo.

- Domina el desarrollo nativo con dos tercios de desarrolladores que lo utilizan. por otro lado, la mitad implementa el desarrollo multiplataforma.

- El 42% utiliza React Native, el 39% ocupa Flutter, el 18% ocupa Cordoba e Ionic

Los resultados de la encuesta del sitio de Stack Overflow en el año 2021 [25] son los siguientes:

- De 66,484 respuestas, el 14.74% realiza desarrollo de aplicaciones móviles.
- De 83,052 personas, la tecnología más utilizada para desarrollo de aplicaciones móviles es; Kotlin con el 8,32%, le sigue Dart con el 6.02%, continua Swift con 5.1%, por último, Objective-C con 2.8%.
- De 59,921 personas que utilizan Frameworks respondieron que; el 14.51% utiliza React Native, el 13.55% utiliza Flutter, el 7.18% ocupa Cordova, y por ultimo, Xamarin es ocupado por el 3.9%
- El entorno de desarrollo integrado (IDE) más utilizado es Visual Studio Code.
- Las tecnologías más apreciadas por 82,914 personas es: Dart con 63.77%, Swift con 63.56%, Kotlin con 61% y Objective-C con con 26%.
- Las tecnologías mejor pagadas es Objective-C con sueldo de 64,859 USD, Swift con pagos de 58,910 USD, kotlin con salarios de 55,071 USD, Dart 32,986.
- El sueldo de un desarrollador móvil es de aproximadamente 41,597 USD.

La encuesta desarrollada de aplicaciones móviles por el sitio de Jetbrains en el año 2021 [19] en la que participaron 31,743 desarrolladores, los resultados fueron:

- El 30% de las personas se enfoca en desarrollo para dispositivos móviles.
- De Kotlin el 14% lo ha utilizado y el 9% quiere probarlo, de Dart el 8% lo ha empleado y el 4% lo quiere ocupar, en Swift el 7% lo ha ocupado y el 5% lo quiere ocupar, por último Objective-C el 3% lo ocupa y el 1% le interesa este lenguaje.
- Los Frameworks más apreciados son Flutter con 68 %, React Native con 58% y Xamarin con 43%.
- El siguiente planteamiento es, de las personas que utilizan un lenguaje, ¿Quien planea adoptar alguna otra tecnología?: De los que usa java; el 13% quiere pasa a Kotlin, el 8% quiere utilizar TypeScript, el 6% quiere usar Swift, el 4% quiere

utilizar Dart. De los que usan Kotlin; el 8% quiere ocupar Swift, el 7% quiere migrar a Dart, el 6% se quiere fijar en Type Script, el 1% opta por Java.

- Las personas que hacen desarrollo móvil, el 94% ocupa Objective c, el 92% ha utilizado Dart, el 90% ocupa Swift, el 63% usa Kotlin,el 35% ocupa JavaScript y el 37% o ocupa TypeScript.

A continuación se realizará un análisis de los datos más importantes de Stack Overflow y JetBrains para poder llegar a unas conclusiones.

TABLA 2.1: Comparación general de Stack OVerflow

Característica	Año 2020	Año 2021
Lenguaje más utilizado	7.8% Kotlin	8.3% Kotlin
	5.9 % Swift	6% Dart
	4.1% Objective-C	5.1% Swift
Framework más utilizado	11.5% React Native	14.5% React Native
	7.2% Flutter	13.51% Flutter
	5.8% Xamarin	7.18% Cordova
Plataformas más utilizadas	26.2% Android	41% Android
	12% iOS	11% iOS
Lenguajes más queridos	62.9% Kotlin	63.7% Dart
	62% Dart	63.5% Swift
	59.5% Swift	61% Kotlin
	23.4% Objective-C	26% Objective-C
Lenguajes mejor pagados	64,000 USD Objective-C	64,859 USD Objective-C
	58,000 USD Swift	58,910 USD Swift
	54,000 USD Kotlin	55,071 USD Kotlin
	37,000 USD Dart	32,486 USD Dart
Personas que buscan trabajo en desarrollo móvil	19%	14%
Salario promedio a nivel mundial	43,000 USD	41,597 USD
Entorno de desarrollo más utilizado		Visual Studio Code

Al analizar los datos de la Tabla 2.1 se obtienen las siguientes conclusiones:

- El lenguaje más utilizado para el desarrollo de aplicaciones móviles es Kotlin, por otra parte, Dart entró fuerte en el último año, ganando a las tecnologías de iOS(Swift y Objective C).

- Del lado de desarrollo multiplataforma, el framework más ocupado es React Native, pasando de 11.5% a 14.5%, en definitiva, ha avanzado; al analizar los datos de Flutter se puede observar que Flutter está creciendo rápido, teniendo la posibilidad de ganar a React Native en un aproximado de 2 años; el framework de Cordova ha crecido más que Xamarin.

- La plataforma que tiene mayor desarrollo es Android.

- Dart es el lenguaje que actualmente es más querido, ganando a los lenguajes más usados de las dos plataformas lideres; otro lenguaje que le agrada a más persona y continúa creciendo es Swift; kotlin se mantiene innovando por algo es la principal tecnología para crear aplicaciones en Android; por último, Objective-C crece en menor medida ya que puede ser requerido para algunas tareas específicas que no se puedan realizar en Swift.

- Los lenguajes que tienen mayor remuneración económica son las tecnologías de Apple (Swift y Objective-C), la siguiente tecnología que deja mayor remuneración es Kotlin, teniendo una diferencia mínima; por último, la tecnología de Dart es remunerada por un poco más de la mitad del salario de las tecnologías anteriores. Es curioso, ya que este lenguaje crea una aplicación para ambas plataformas.

- El porcentaje de las personas que buscan trabajo como desarrollo de aplicaciones móviles ha disminuido en comparación con el año anterior.

- El salario promedio de un desarrollador ha disminuido con el año anterior.

Al estudiar los datos de la Tabla 2.2 obtenemos que:

- Kotlin es el lenguaje que más se utiliza para el desarrollo de aplicaciones móviles, Dart es el segundo lenguaje que más se utiliza en los últimos dos años, Swift es es el tercer lenguaje más ocupado y Objective-C es el 4 lenguaje más ocupado en las tecnologías anteriores.

TABLA 2.2: Comparación general de JetBrains

Característica	Año 2020	Año 2021
Lenguaje más utilizado	17% kotlin	14% kotlin
	9% Dart	8% Dart
	5% Swift	7% Swift
	4% Objective-C	6% Objective-C
Framework más utilizado	42% React Native	68% React Native
	39% Flutter	58% Flutter
	18% cordova	43% Xamarin
Lenguajes más queridos		94 % Objective-C
		92% Dart
		90%Swift
Personas que buscan trabajo		30%

- Los Frameworks más utilizados son React Native, Flutter es el segundo Framework más ocupado en las tecnologías,de igual manera, Xamarin elimina a Cordova obteniendo el último año el 43%.

- Los lenguajes más adeptos son: Objective-C, Dart y Swift, Con poca variabilidad de porcentaje en cada una de las tecnologías.

- El porcentaje de los desarrolladores que busca trabajo para el desarrollo móvil es de 30%.

De manera general se puede concluir que los lenguajes más utilizados para desarrollo móvil son los lenguajes nativos Kotlin y Swift, se igual manera, los Frameworks también forman buena parte del desarrollo móvil y los más utilizados son React Native y Flutter.

El entorno de desarrollo más utilizado por los programadores es Visual Studio Code

2.2 Generalidades

2.2.1 Framework

Es un marco o entorno de trabajo donde establece un conjunto de reglas y herramientas para poder desarrollar otro software en el que realiza tareas específicas. El framework

tiene que estar definido por una estructura en la que permita relacionar cada herramienta que proporciona.

Un framework permite trabajar con más de un lenguaje de programación. Esto nos sirve para poder identificar las tareas que puede realizar respetando las reglas del mismo, seccionando cada parte del desarrollo de software y utilizarlo en el momento que sea requerido.

Existen diferentes tipos de frameworks cada uno se enfoca en una necesidad como por ejemplo: para crear páginas Web, para desarrollo de aplicaciones móviles, para mercadotecnia, bases de datos, entre otros. Como vemos, el desarrollo de tareas es más complejo, por lo que, es necesario el uso de herramientas que facilitan la integración de diferentes programas para poder realizar un mejor trabajo de manera profesional.

El término de Framework se popularizó gracias a las tecnologías Web, ya que existían demasiadas herramientas para realizar alguna tarea, estos procesos de programación en diversos casos resultaban ser muy complejo o tedioso, dejando una gran posibilidad para la optimización de procesos de cada herramienta de desarrollo. Algunos frameworks que surgieron son: Angular, React, Laravel, DJango, .Net y Spring boot.

2.2.2 Android

Es un sistema operativo donde su kernel está basado en Linux, esta plataforma es de código abierto. Sus aplicaciones son desarrollo libre, actualmente, de manera nativa, se puede programar de dos maneras: la primera es con Java y la segunda es con Kotlin; a mediados del 2022, Kotlin se convirtió en el principal lenguaje para desarrollar aplicaciones en Andorid. Su mascota es un robot verde llamado Andy.

Kotlin

Kotlin es un lenguaje de programación de código abierto anunciado en el año 2010, su creador es JetBrains, una empresa famosa que se encarga a desarrollar software a nivel profesional de alta calidad; es popular por hacer un código más simple en comparación

con los lenguajes más famosos de hace algunos años, como con Java, c++, o Scala.

Sus principales características son:

- Maneja un desarrollo fuertemente tipado, evitando el intercambio de tipos de datos en una variable; .
- Fue diseñado para trabajar con código de Java, dejando hacer una buena interoperabilidad.
- Permite generar código para la Maquina Virtual de Java(JVM).
- Tiene que realizar compilación en proceso de desarrollo.
- Es opcional utilizar ";" al final de cada instrucción.
- Introduce la opción que ningún valor obtenga el valor nulo.

2.2.3 Historia de Android

Android fue desarrollado en el año 2003 por 4 fundadores: Andy Rubin, Nick Sears, Rich Miner y Chris White. Se desarrollo en el estado de California, de los Estado Unidos de América, Posteriormente fue adquirida por Google en el año 2005. Gracias a que en el año 2007 se creó un consorcio llamado "Handset Aliance" en el que participaron diversas empresas como: Motorola, Samsung Ericsson, T-Mobile, entre otros; Donde, se establecieron ciertos estándares para facilitar el funcionamiento entre el hardware y software.

2.2.4 Arquitectura

Android [10] es un conjunto de software abierto, esto permite que la mayoría de fabricantes pueda modificarlo y crear una versión para cada uno. En la Figura 2.2 se presentan los elementos base de la plataforma, posteriormente se explicará de manera detallada cada elemento.

Figura 2.2: Arquitectura de Android

Kernel de Linux

[10] Android se basa en el kernel de Linux permitiendo usar algunas funcionalidades como generar subproceso y administrar memoria. Esto da facilidad para que cada fabricante desarrolle una variedad de controladores de smartphone.

Capa de Abstracción de Hardware (HAL)

[10] Permite el uso de algunas herramientas con características muy específicas para el uso del hardware como puede ser una interfaz de programación de aplicaciones (API). Estos paquetes permiten el uso de módulos como puede ser cámaras, sensores o bluetooth; mediante el sistema Android.

Tiempo de Ejecución de Android

[10] En los dispositivos con Android 5.0 o superior las aplicaciones ejecutan sus propios procesos mediante instancias del tiempo de ejecución de Android (ART). ART permite ejecutar varias máquinas virtuales mediante la ejecución de archivos DEX que se encuentra optimizado, permitiéndole utilizar el consumo mínimo de memoria.

Bibliotecas C/C++ Nativas

[10] La mayoría de componentes y servicios muy específicos como ART o HAL se encuentra hecha en código nativo, por esta razón, requiere de bibliotecas construidas con C y C++. Para el uso de estos paquetes se requiere de NDK de Android.

Framework de la API de Java

[10] Al momento de desarrollar una aplicación en Android se requieren usar las funciones de este sistema operativo que se encuentran alojadas en API, que están escritas en Java. Los componentes más centralizados son los siguientes:

- Sistema de vista nos permite utilizar diferente elemento para mostrar en una interfaz como puede ser; botones, cuadriculas, listas o hasta un navegador web.
- Administrador de recursos nos permite utilizar recursos sin código, como pueden ser gráficos o archivos de diseño.
- Administrador de notificaciones. nos permite que la aplicación presente alguna alerta mediante el icono de la aplicación o en la barra de estado.
- Administrador de Actividad se encarga de gestionar los ciclos de vida de la aplicación permitiendo retroceder a la navegación entre varias activity.
- Proveedores de contenido nos deja que la aplicación acceda a datos de una o varias aplicaciones como puede ser de contactos o llamadas.

Apps del Sistema

[10] Android nos proporciona algunas apps básicas como puede ser correo electrónico, mensajes, calendarios, contactos u otras. Estas aplicaciones ayudan de alguna manera, ya que son usadas de manera común, sí se requiere realizar alguna tarea de estas aplicaciones se puede mandar a llamar a la aplicación requerida, facilitando el desarrollo de alguna aplicación.

2.2.5 Versiones

En la página De Andro4all [12] se recopiló la información de la evolución de las diferentes versiones de Android La primera versión de Android fue lanzada en el año 2008 acompañado por la marca HTC. Algunas características son: la tienda de aplicaciones era Android Market, esa versión era que las aplicaciones eran gratuitas, el acceso a las aplicaciones instaladas era mediante unos iconos que se mostraban con una pequeña animación a un costado del lado derecho; esos iconos son conocidos como Widgets.

La siguiente versión que implemento un avance importante fue Android 1.5, aparece a principios del año 2009.Algunas de sus novedades fueron: se incorporó las pantallas táctiles con integración al teclado virtual. Recordemos que en esta versión los dispositivos que contaban con este sistema eran muy pocos.

Android 2.0 surge a finales del año 2019 y resulta ser un gran cambio para el sistema operativo ya que dejaba usar el GPS y se incluía la aplicación de Facebook preinstalada, implemento botones en la parte inferior de la pantalla. Posteriormente lanza una actualización para emplear la opción de "pich to zoom" que básicamente permite reducir o ampliar el tamaño de cierto contenido como podría ser imágenes, mapas o algún otro contenido.

La siguiente versión fue Android 2.2 con el nombre de Froyo donde añadió el compilador Dalvik JIT que permitía mejorar el rendimiento del sistema operativo, incorporó un icono donde mostraba todas las aplicaciones que contenía el dispositivo en la pantalla principal, se logró adoptar Adobe Flash Player, trajo el control por voz

y se daba la posibilidad para mover aplicaciones a la memoria microSD.

Android 2.3 Gingerbread, llego con un cambio muy importante en la interfaz del panel de notificaciones que resaltaba más, gracias a que era de color negro; impulso el uso multimedia mediante interfaz de programación de aplicaciones (APIs) y la creación de juegos.

Android 3.0 su enfoque principal fue las tabletas. Sus atributos principales fueron: Tenían una interfaz más amplia, a ningún teléfono le llego la actualización para este sistema operativo; surge la idea de implementar botones en la interfaz que permite acceder al menú, home y regresar.

Android 4.0 su principal novedad fue que implemento pantallas más grandes ya que incorporaba los botones en la parte inferior de la pantalla del smartphone, cambio el nombre de la tienda de aplicaciones de Android Market a Google Play Store y otras aplicaciones propias de Google también fueron renombradas.

Android 5.0 Lollipop fue presentada en el año 2014 las principales cualidades fueron: permitir un mayor ahorro de batería mediante el sistema "Project Volta" y el diseño de Material Design; que marco un antes y un después para los desarrolladores.

Material Design es una normativa creada por Google, que se encarga de darle un control a cada interfaz de aplicaciones mediante un conjunto de reglas que puedes ser: colores, botones, imágenes, espacio 3D, animaciones, transiciones, etc. Otra de las grandes novedades fue que incorporaron la máquina virtual ART; que hizo que las aplicaciones fueran rápidas al momento de ejecutarlas, se dio soporte para aplicaciones de 64 bits que para la actualidad es común.

Llega en 2015 la versión 6.0 agregando el sistema de permisos granular, que permitía activar y desactivar diferentes permisos que requería una aplicación sin necesidad de irse a los ajustes del teléfono; gracias a que mostraba un panel con las opciones de aceptar o rechazar. Otra novedad que mencionar fue Adoptable Storage, que permitía transformar la memoria externa adaptándola como ampliación de memoria interna (una función muy útil, por cierto). También empezaba a tener compatibilidad con los lectores de huellas Dactilares.

En el año 2016 llegaría Android 7.0 con la llegada del primer pixel (teléfono creado por Google) en la que permitía ejecutar 2 aplicaciones en la pantalla partida, daba la posibilidad de contestar mensajes por medio de las notificaciones, del lado de los videojuegos se empezó a dar soporte a la API de Vulkan, se incorporaron unas funciones rápidas a las aplicaciones mediante una pulsación desde el icono de la aplicación.

En el año 2017 surge Android 8 incorporaba una gestión de notificaciones clasificándolo por prioridad mostrándolos por un canal de notificaciones, se incorporaron 2 sistemas de notificaciones; Notification Dots y Notification Badges,se implementa el modo Picture in Picture que básicamente permite mostrar contenido Multimedia en algún dispositivo externo o en la misma pantalla del teléfono. Android cambia la estructura del sistema operativo implementando módulos en el mismo, separando los drivers con el hardware de los dispositivos de cada vendedor, dando como resultado la liberación de actualizaciones de manera rápida por cada fabricante. Posteriormente llega una actualización de la implementación de la API para redes neuronales dándole la oportunidad para crear sistemas de inteligencia artificial y Machine Learning en las aplicaciones.

A principios del año 2018 surge Android 9.0 Pie; mejorando la interfaz del usuario dando unos tonos claros creando estándares de desarrollo; se implementan los gestos en el teléfono; se mejora el sistema de posicionamiento a través de Wifi; incorporo restricciones de acceso a: micrófono, cámara, sensores; se empieza a dar soporte para diferentes ventanas con diferentes tipos de notch; se da soporte para incorporar un sistema fotográfico formado por dos cámaras; mejorando los ajustes como capturas de pantalla, control de volumen.

A inicios del año 2019 llega el lanzamiento de Android 10 implementando: burbujas flotantes en notificaciones de aplicaciones compatibles. Se incrementa el uso de multitareas como puede ser responder mensajes mientras se reproduce un vídeo o se usa alguna otra aplicación. En febrero del 2020 llega Android 11 restringiendo los permisos para las aplicaciones incluso para un solo uso, estos dispositivos vienen con soporte para redes móviles 5G, se impulsa las pantallas curvas o plegables

En febrero del 2021 Google anuncia Andorid 12 su principal cambio es adaptar la interfaz del usuario con los colores que este ocupando la pantalla en ese momento, se cambió el diseño de la barra de ajustes, se integran las respuestas inteligentes, el diseño de widgets mejora, mejora en interfaz de pantallas grandes. [12]

2.2.6 Ciclo de Vida de la Aplicación

Actualmente en Android, el programador no puede controlar la duración de los procesos de una aplicación, como, por ejemplo: cuando se deja una aplicación ejecutándose por un periodo muy largo; el proceso de la aplicación es controlado por el sistema operativo, por lo tanto, si el sistema operativo requiere de memoria o procesos, el sistema empieza a eliminar algunos procesos para poder realizar alguna actividad y el programador no puede intervenir.

Sí el programador no entiende cómo se comportan los componentes principales como Activity, Service y BroadcastReceiver o no se usan correctamente, el sistema operativo puede eliminar procesos de la aplicación o en el peor de los casos eliminada la aplicación.

Android maneja los procesos por un sistema de jerarquía. A continuación, se aplicará los procesos por orden de importancia que tienen:

1. El proceso de primer plano realiza tareas que se están ejecutando actualmente. Los procesos que se crean en primer plano deben por lo menos contener uno de los siguientes elementos:

    - Debe ejecutar una Activity que este interactuando con el usuario y llamo a su método onResume().
    - Puede contener un método BroacastReceiver en ejecución con la función BroadcastReceiver.onReceive().
    - Se está ejecutando un service y está esperando una devoucion de los metodos Service.onCreate(), Serivce.onStart() o Service.on Destroy().

En el caso que la aplicación no cumpla con los recursos de memoria, eliminará procesos en el caso que sea posible, por el contrario, se cerrara y no se ejecutará.

2. El proceso sea visible. para que el proceso sea considerado visible debe de cumplir con los siguientes procesos:

   - Deberá ejecutar una actvity que se deberá encontrar en primer plano y por alguna razón llamar al metodo onPause().
   - Se Tendrá que ejecutar un Service en primer plano el método Service.startForeground() (El usuario debe de conocer que el servicio se está ejecutando). Un ejemplo para entender este proceso es, cuando queremos hacer una llamada y queremos activar el micrófono, la bocina, el servicio sería el acceso a la red.
   - Se aloje un servicio que el sistema use para su funcionamiento, como puede ser el teclado, protector de pantalla, el launcher de aplicaciones, entre otras más.

3. El proceso de servicio que contenga un Serivce que inicialice al método startService(). Estos procesos no pueden estar directamente visibles. Un ejemplo puede ser cuando ejecutamos una aplicación que implique el uso de mapas y la ubicación; el servicio que se está ejecutando será el sistema GPS (Sistema de posicionamiento Global).

4. Un proceso almacenado en cache ya que su acceso de memoria es rápido para acelerar proceso; Sí requiere de uso de memoria, el sistema ocupara memoria de otra aplicación. Este proceso requiere de varias instanciáis del tipo activity ocupando el método onStop(). Este proceso nos ayuda al momento de que reclame memoria el teléfono, no afecte a la aplicación anterior. [11]

2.3 Algunos Frameworks para desarrollo de aplicaciones para dispositivos móviles Android

2.4 Framewok Flutter

Flutter es un framework para desarrollo móvil que fue desarrollado por Google, su primera versión fue lanzada en el año 2019. Este framework es capaz de desarrollar interfaces muy atractivas en las cuales puede ser compleja para el desarrollo en otras plataformas las aplicaciones más sobresalientes hechas en este framework son: Alibaba Google ads y Hamilton música. Este framework cuenta con el soporte de Google [15]. Flutter utiliza el lenguaje dart, que permite desarrollar aplicaciones parecidas a Java. Para poder desarrollar una interfaz gráfica debemos de utilizar la biblioteca de material, el cual, esta biblioteca, nos va a permitir utilizar componentes que en Flutter son llamados widgets, estos elementos serán primordiales para desarrollar las interfaces gráficas si no se importa esta biblioteca no se podrá desarrollar interfaces gráficas.

A diferencia de otras tecnologías Flutter nos permite desarrollar interfaces en aplicaciones de forma nativa para las múltiples plataformas que en la actualidad son dos: Android e iOS en la actualidad no existe otro framework capaz demostrar interfaces de forma nativa cómo lo hace Flutter ya que este cuenta con una biblioteca llamada Cupertino que permita permite dibujar los widgets a un estilo de iOS. Una de las principales tecnologías que facilitan al desarrollo de las aplicaciones en este framework es el hot- reload el cual nos permite visualizar cambios instantáneos al momento de guardar el archivo que se está editando ya sea en el teléfono o en un emulador sin necesidad de esperar varios minutos. La otra es el hot restart el cual nos permite volver a subir la aplicación como si fuera la primera vez que se suba al dispositivo el único defecto que tiene este método es que si se edita el archivo “pubspect.yaml” se tiene que reiniciar la aplicación sin importar si se tenga en emulador o en dispositivo físico. Flutter está basado principalmente en widgets ya que todo lo que se muestra en la pantalla son forzosamente widgets. Para poder crear una aplicación se necesita crear

árboles de widgets. ¿Qué es un widget? En Flutter se basaron de React: así como existen componentes en react native en Flutter le llaman widgets. Básicamente un widget es una función que contiene un elemento que permite construir una interfaz. [16]

2.4.1 Dart

Dart es un lenguaje de código abierto desarrollado por Google, su lanzamiento se realizó el año 2011. Se comenta que este lenguaje fue diseñado para reemplazar a JavaScript, lamentablemente no fue exitoso, se empezó a utilizar este lenguaje hasta que llego Flutter. Actualmente se puede utilizar para web, servidores, aplicaciones de consola y aplicaciones de teléfonos.

Características Principales

- Permite interpolación de cadenas de texto.
- Es opcionalmente tipado.
- Es un lenguaje que requiere de compilación o ser interpretado en su propio Dart VM
- El lenguaje está orientado a objetos.
- Permite utilizar programación asíncrona.

2.4.2 Ciclo de Vida de la Aplicación

El ciclo de vida de vida de la aplicación nos permite comprender como la aplicación va cambiando de estados al momento que se ocupa en el dispositivo. Como se mencionó anteriormente, absolutamente todo en Flutter son Widgets, los widgets que manejan uno o varios estados son Stateless Widget y Stateful Widget.

En Stateless Widget básicamente únicamente crean un estado al momento de la creación del Widget, esto evita que todos los widgetes que estén dentro de él no cambien

de estado en el proceso de ejecución, a menos que se cierre la aplicación .

Los Stateful Widget mantiene el estado del widget, permitiendo modificarlo frecuentemente en proceso de ejecución de la aplicación, de esta manera el usuario puede ver cambios en tiempo real de la aplicación. Un ejemplo de este widget podría ser un reloj digital, ya que constantemente están cambiando los dígitos y se debe de mostrar al usuario.

Los estados por lo que se pueden pasar son: CreateState(),initState(), didChangeDependencies(), buid(), didUpdateWidget(), setState(), deactivate(), dispose().

A continuación, se dará una breve explicación de los estados que puede tener un Widget en la etapa del ciclo de vida:

- createState() Este método se ejecuta cuando se crea un Stateful Widget.
- initState() este método inicializa el estado del widget, este método se ejecuta antes de que se construyan los Widgets.
- didChangeDependenceies() este método se ejecuta enseguida de que ocupa el método initState() nos va a permitir modificar dependencias en tiempo de desarrollo.
- build() este método principal nos permite representar todos los widgets que se ocupen en la interfaz del usuario.
- didUpdateWidget() se utiliza cuando se ajusta una configuración por el widget principal, permitiendo retirar los cambios del widget anterior y mostrar los del widget actual.
- setState() se utiliza para notificar a la estructura interna de la aplicación ha cambiado y tiene que volver a compilar el estado del objeto.
- deactivate() este método se utiliza cuando se requiere remover el estado del widget pero posteriormente puede volver a ser incrustado en otro momento. un ejemplo

es cuando queremos que la aplicación guarde un contacto y abre la aplicación de contactos para realizar esa tarea. Al momento de que guardamos el contacto te vuelve a abrir la aplicación para continuar con alguna otra tarea.

- dispose() se utiliza cuando el objeto y estado se quieren eliminar permanentemente del árbol de Widgets, generalmente se utiliza para dejar de eliminar recursos de la aplicación como puede ser documentos, temporizadores, o algunas transmisiones de datos. [22]

2.5 Framewok React Native

React Native es un framework de código abierto que permite el desarrollo de aplicaciones móviles, está desarrollado por el equipo de Facebook. React Native utiliza las bibliotecas de react, permitiendo crear componentes visuales implementadas de manera nativa en cada uno de los sistemas operativos que en este caso son Android e IOS. este framework se basa en bloques para la interacción de interfaces gráficas de javascript con las plataformas nativas.

React Native fue lanzado en el 2015, año en el que Facebook anunció que su aplicación para dispositivos móviles ya estaba desarrollada con React Native. Este anuncio provocó que muchos desarrolladores web Estudiaran este framework para implementarlo en sus aplicaciones móviles. posteriormente iOS no permitiría lanzar sus aplicaciones en la App Store sí no contaban con páginas web por lo que impulsó más el uso de esta tecnología. [13]

Uno de los factores más importantes para que React Netive tuviera éxito, fue que las aplicaciones más utilizadas como Facebook, Instagram, Skipe fueron desarrolladas con este framework.

React native utiliza una extensión dejava script llamada jsx donde se recomienda utilizarlo para describir la interfaz de usuario mediante elementos de react. JSX permite unir Tecnologías como java script y componentes css creando unidades ligeramente

acopladas llamadas ¨componentes¨ que es en lo que se basa React Native:la gran mayoría de elementos de este Framework utiliza componentes. Un componente puede referirse a otro componente: debido a que cada componente tiene una estructura ya definida y estandarizada nos permite utilizar un componente en el nivel que se encuentre.

DOM (Document Object Model) es un modelo o representación gráfica de un documento a la aplicación permitiendo realizar cambios necesarios en cada actualización.

El virtual DOM es una representación del DOM guardada en memoria, que actúa de intermediario entre los estados de la aplicación y los estados del DOM (vistos por el usuario).

Características

Sus características principales son:

- Requiere ser compilado en proceso de desarrollo.
- Puede visualizar cambios al instante.
- Es la primera opción para ser adaptado por tecnologías web por tener la implementación de JavaScript o TypeScript.
- Cuenta con una gran comunidad. [21]

2.5.1 Ciclo de vida de los componentes

Todos los componentes de React tienen etapas cuando se utilizan en aplicaciones, al momento que se instancia un componente y se inserta en el DOM obtiene sus propiedades. El componente no está obligado a pasar por todas las fases, únicamente implementa las que son requeridas en la aplicación. Estas etapas son conocidas como ciclo de vida y se divide en 4 fases:

- Mounting crea la instancia de un componente y la inserta al DOM.Los métodos que se utilizan son: constructor(), render() y ComponentDidMount().

- Updating se realiza cuando el componente requiere mostrar actualizaciones. algunos métodos que se emplean son: shouldComponentUpdate(), ComponentDidUpdate(), y en este caso se puede implementar el método render() también.

- unmounting este proceso remueve el componente del DOM.El único método que usa es ComponentWillUnmount().

- Error Handling se produce cuando se ocurre un error al momento de renderizar el componente. Los métodos que utilizan es getDerivedStateFromError() y componentDidGatch(). [23]

2.6 Comparación de tecnologías

2.6.1 Características de equipo de cómputo

Las características del equipo de cómputo donde se realizaron las pruebas está compuesto por: un sistema operativo Windows 10 de 64 bits, un adaptador de red ralink RT5390 802.11 b/g/n, sus características principales se muestran en la Tabla 2.3.

TABLA 2.3: Características de computadora

Procesador	Memoria RAM	Almacenamiento
Intel i5 3210m 2 núcleos 4 Hilos Frecuencia 2.5GHz a 3.10GHz 3 MB de Cahe	12GB DDR3 8GB + 4GB Frecuencia 1600MHz	SSD 480GB ADATA SU650

Para crear cualquier aplicación es necesario tener una conexión a Internet que permita descargar los paquetes que requiere. Al momento de realizar las pruebas se usó una conexión a Internet con 40Mbps de descarga.

2.6.2 Especificaciones de tecnologías

Las tecnologías previamente instaladas en el equipo de computo se describen en la Tabla 2.4.

TABLA 2.4: Características tecnológicas instaladas

Tecnología	Versión	Lenguaje	Almacenamiento
Flutter	2.0.0	Dart V2.12.0	1.83GB + sdk de Android Studio
React Native	0.67.3	TypeScript V4.5.5	1.2GB + sdk de Android Studio
Android Studio	2021.2.1	Kotlin	1.4GB + sdk que se requieran

Los frameworks (entorno de trabajo) React Native y Flutter no tienen entorno de desarrollo, por lo que se utilizará el editor de código" Visual Studio Code". Para crear aplicaciones nativas de Android se utilizará Android Studio.

En la Figura 2.3 se puede observar que el entorno de desarrollo Android Studio, está desarrollado para crear aplicaciones con la tecnología de Android nativo, permitiendo encontrar de manera rápida las herramientas; optimizando la interfaz para la navegación del proceso de desarrollo como puede ser: el panel que muestra la estructura de archivos del proyecto, una ventana que muestra los archivos seleccionados en la estructura del proyecto, un panel donde identifica si es una interfaz gráfica, crea otra sección para poder mostrar, manipular y configurar la interfaz de la aplicación; de la misma manera, muestra iconos que se están haciendo procesos de compilación o ejecución de ciertas herramientas.

En la Figura 2.4 se muestra la interfaz de Visual Sudio Code. Se puede observar que este software está diseñado para el desarrollo de código, por la simpleza que tiene, no satura de información la pantalla y organiza de manera óptima los procesos que necesita un programador para realizar aplicaciones.

Una característica extra por parte de React Native es que se utilizará React Native CLI (Interfaz de línea de comandos) ya que nos va a permitir interactuar con código nativo y no presenta restricciones por compatibilidad, pero requiere instalar los sdk de Android o Xcode para iOS, por lo que será pesado para la computadora. De esta manera, será similar a la creación de aplicaciones en Flutter. De lo contrario se ocuparía

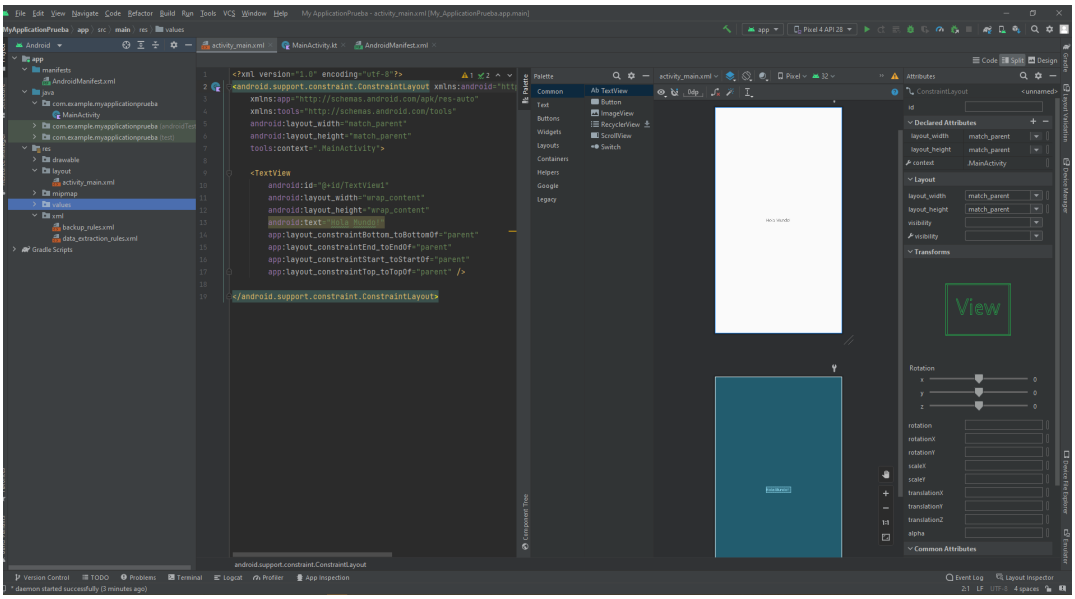


Figura 2.3: Entorno de desarrollo Andorid Studio

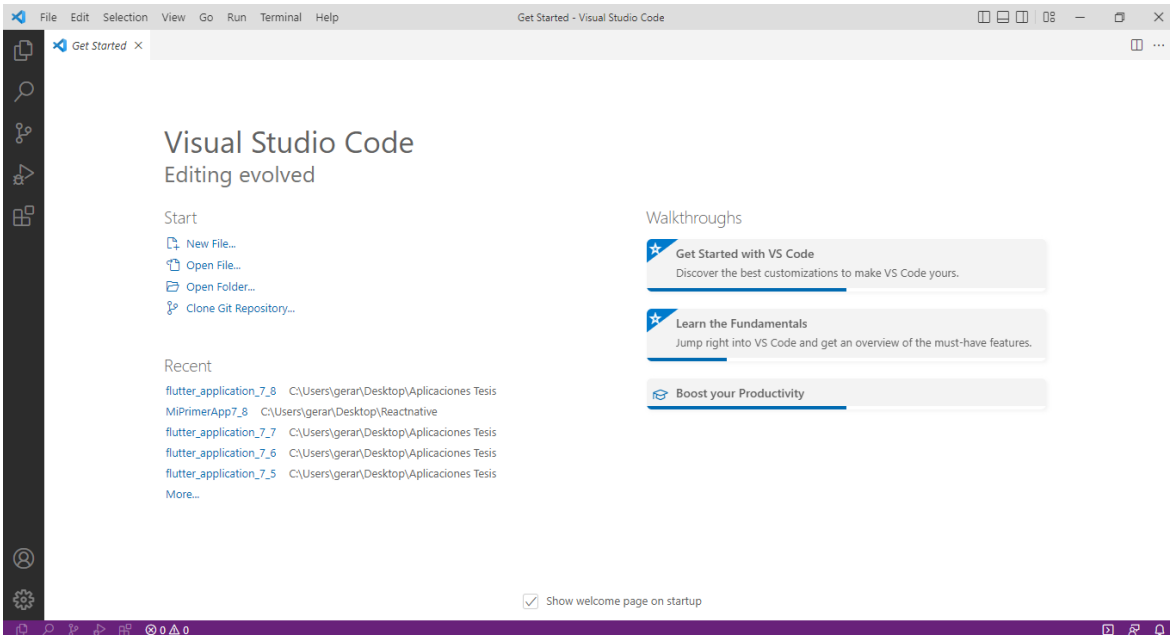


Figura 2.4: Entorno de desarrollo Visual Studio Code

Expo CLI facilitando el desarrollo, sin manipular ninguna configuración por parte de lo nativo convirtiéndose en una caja negra y ese no es el objetivo.

2.6.3 Pruebas de rendimiento en equipos de cómputo

Para poder tener una noción clara de los procesos básicos que se utilizan al momento de crear una aplicación, se realizaron 30 pruebas en las que se evaluaron: Cargar Editor, Crear Aplicación , Ejecutar Aplicación por Primera Vez, Ejecutar aplicación en proceso de desarrollo. Posteriormente, se obtuvo el promedio y desviación estándar, estos datos nos permitirán obtener un intervalo de los valores más frecuentes en las pruebas que se realizaron. Esos valores se muestran en tablas de esta sección. Con estos datos el programador entenderá como va evolucionando el análisis de la aplicación en proceso de desarrollo.

En las Figuras 2.5 y 2.6 se detallan datos de los entornos de desarrollo como son el tiempo de carga y el consumo de memoria RAM en el equipo de cómputo respectivamente.

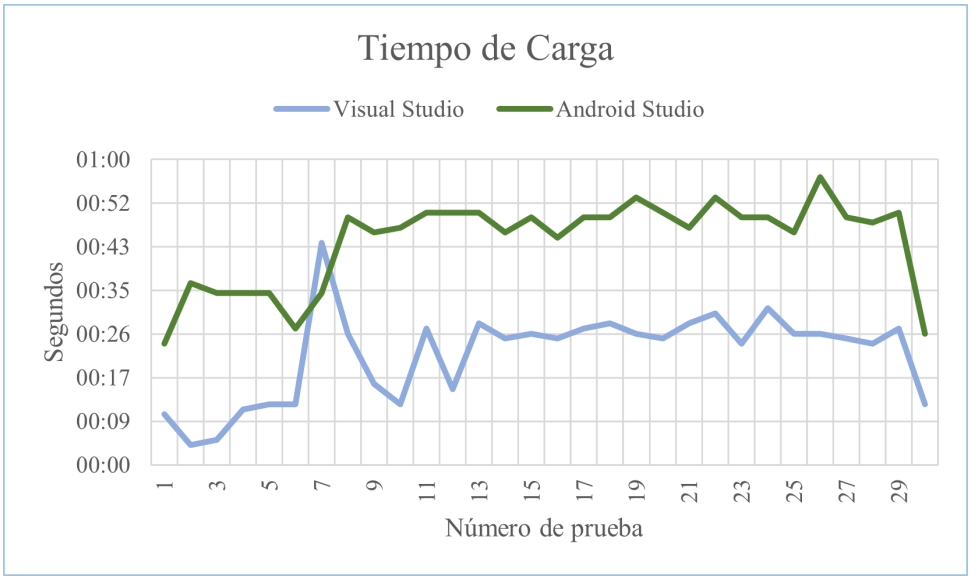


Figura 2.5: Tiempo de carga Android Studio y Visual Studio

TABLA 2.5: Requisitos para Abrir Entornos de Desarrollo en Laptop

Entorno de desarrollo	Tiempo de apertura	Memoria RAM
Android Studio	35seg. a 53seg.	1034MB a 1446MB
Visual Studio Code	13seg a 31seg	628MB a 1242 de RAM

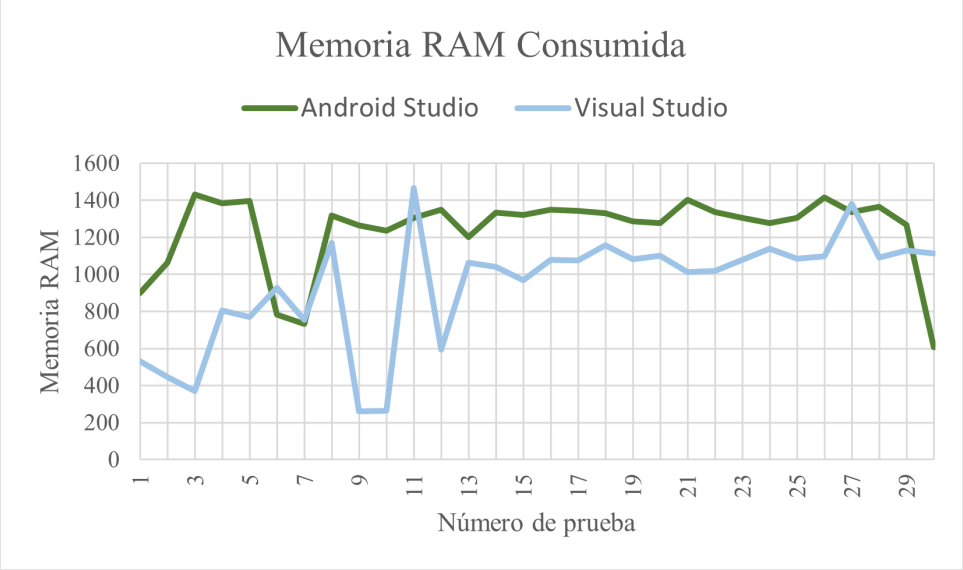


Figura 2.6: Consumo de memoria RAM Android Studio y Visual Studio

Dando continuidad al proceso, se crea la aplicación en Android Studio con base en el lenguaje de programación Kotlin, propuesto por Google para la creación de aplicaciones nativas.

En las Figuras 2.7, 2.8 y 2.9 se vislumbra los resultados de mediciones al momento de crear las aplicaciones en las tecnologías Android, Flutter y React Native.

En la Figura 2.7 se evaluó el tiempo que tarda en crear la aplicación, en la Figura 2.8 se muestra el almacenamiento en el equipo de cómputo al momento de crear la aplicación en las Tecnologías de Android y Fluter, por último, en la Figura 2.9 se observa el almacenamiento que requiere el framework de React Native durante la elaboración del presente trabajo.

TABLA 2.6: Mediciones para crear aplicaciones por cada tecnología.

Característica	Android Studio	Flutter	React Native
Tiempo de creación	69seg a 79seg	21seg. a 39 seg.	166seg. a 226seg.
Tamaño de la app	540KB a 697KB	300KB	328704 MB.

Como se observa en la Tabla 2.6 se determina que el entorno de desarrollo óptimo es Flutter al tener un menor tiempo de creación y optimizar el almacenamiento de la aplicación. La siguiente alternativa es Android Studio consumiendo un poco más de

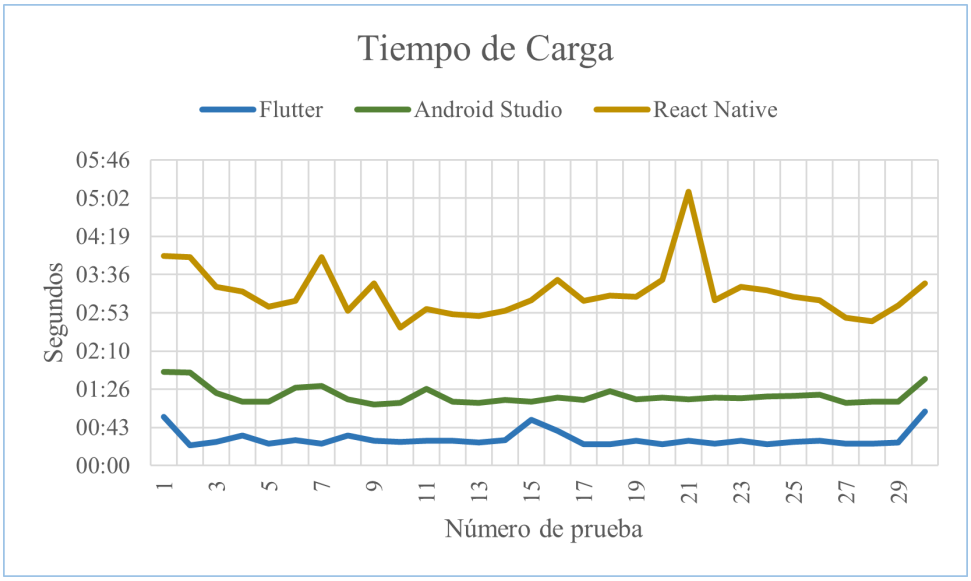


Figura 2.7: Tiempo de carga al crear cada tecnología

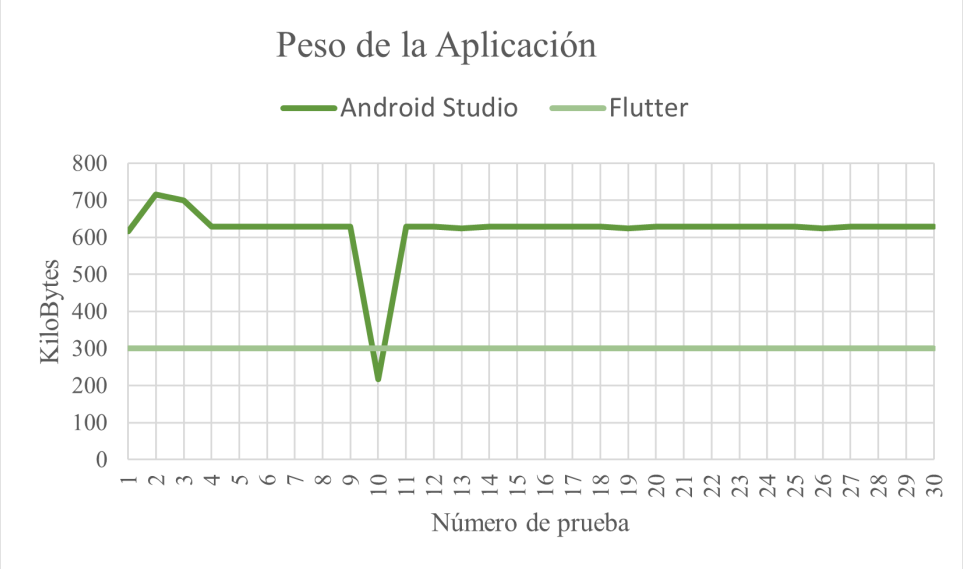


Figura 2.8: Consumo de memoria al crear app en Visual Studio y Flutter

tiempo que Flutter. Por último, React Native con un tiempo de diferencia considerable para crear la aplicación y el peso de la aplicación si es bastante superior.

Posteriormente el proceso que se evaluó ejecutar las aplicaciones. El teléfono que se usa para realizar el proceso de ejecución de las apps es un Nokia 6.1. Sus características son: Procesador Snapdragon 630 (frecuencia máxima de 4 núcleos a 2.2GHz o frecuencia

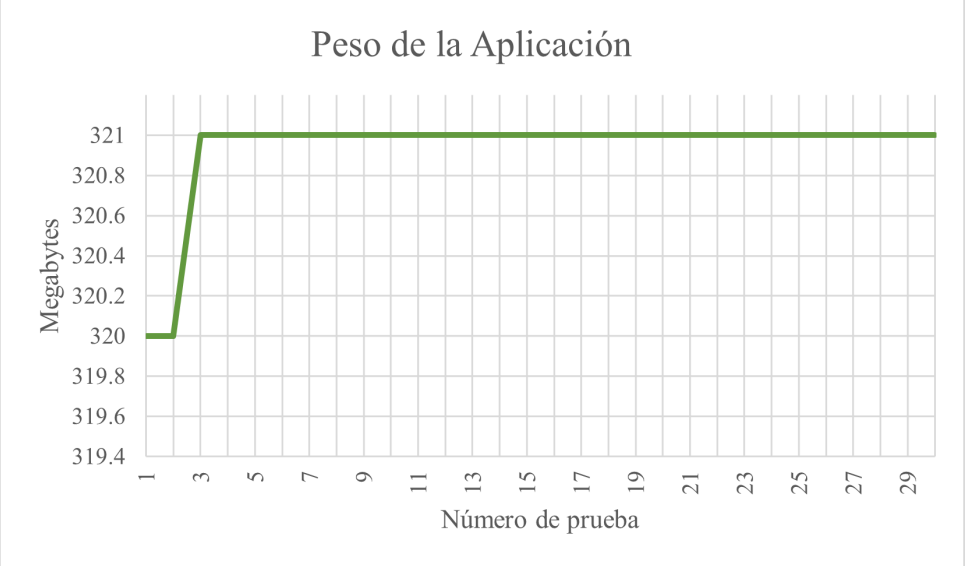


Figura 2.9: Consumo de memoria al crear app en React Native

mínima de 4 núcleos a 1.9GHz), Memoria RAM 3GB, GPU Adreno 508, Sistema operativo Android 10, 32GB de almacenamiento.

Los parámetros en tiempo para ejecutar la aplicación se muestra en la Figura 2.10, el almacenamiento que requiere el equipo de cómputo se puede observar en la Figura 2.11, el consumo de almacenamiento que requiere el teléfono se muestra en la Figura 2.12 ,la cantidad de memoria RAM que requiere se muestra en la Figura 2.13.

TABLA 2.7: Mediciones de proceso para ejecución de la app en teléfono por encendido.

Característica	Android Studio	Flutter	React Native
Tiempo de creación	58seg. a 96seg.	122seg a 154seg	205seg a 227seg
Uso memoria RAM en PC	2243MB. a 3149MB	2062MB a 2703MB	1621MB a 2566MB
Peso de la aplicación	42.1MB	543MB	568MB
Peso de la app en teléfono	10.19MB a 11.83MB	119MB	71.67 MB

Los datos de la tabla 2.5 son hallazgos mediante los cuales se demuestra que: 1.- android nativo es una tecnología que requiere de menor tiempo, menor consumo de almacenamiento en equipos de cómputo y menor consumo de memoria ROM en teléfonos cuando se ejecuta por primera vez. Cabe señalar que esta tecnología únicamente compila los paquetes necesarios para el sistema operativo Android, mientras

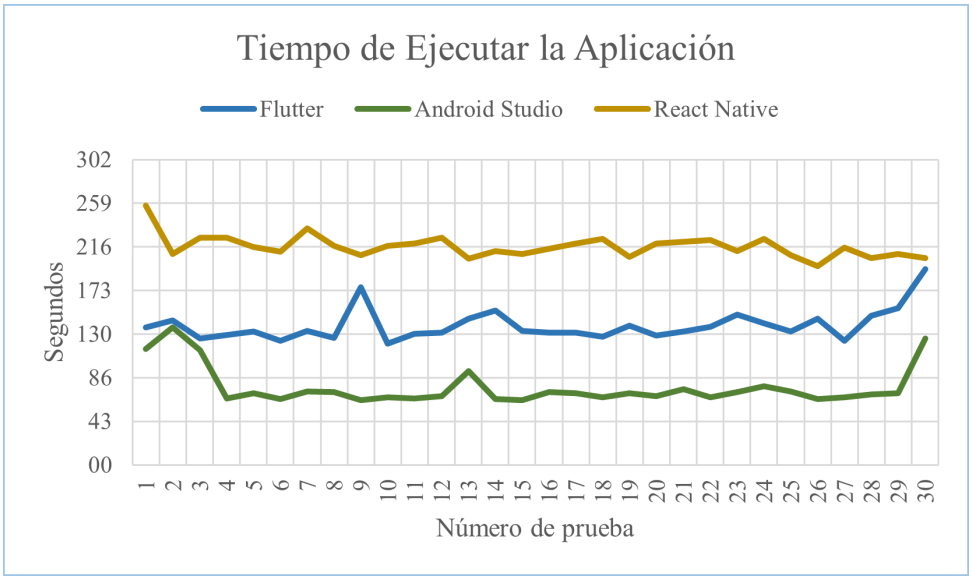


Figura 2.10: Tiempo de ejecución por primera vez de aplicaciones móviles

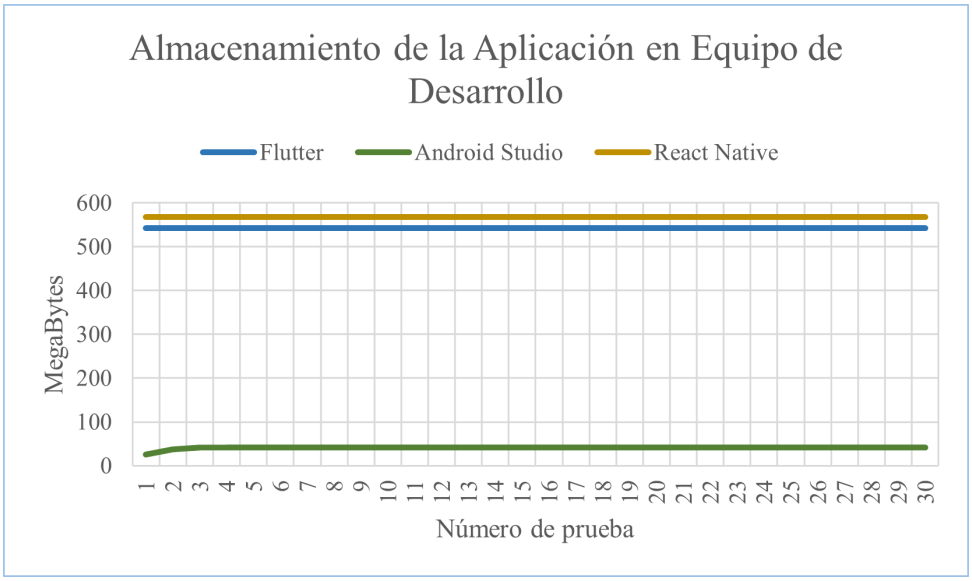


Figura 2.11: Consumo de almacenamiento de la app en equipo de cómputo en desarrollo

tanto, Flutter y React Native crean los paquetes para más de dos plataformas "multiplataforma". Al comparar tiempos entre Andorid y Flutter se perciben similitudes durante la ejecución.

2.-Fluter al consumir peso de la aplicación a la computadora como en el teléfono es más elevado en comparación con con Android nativo.

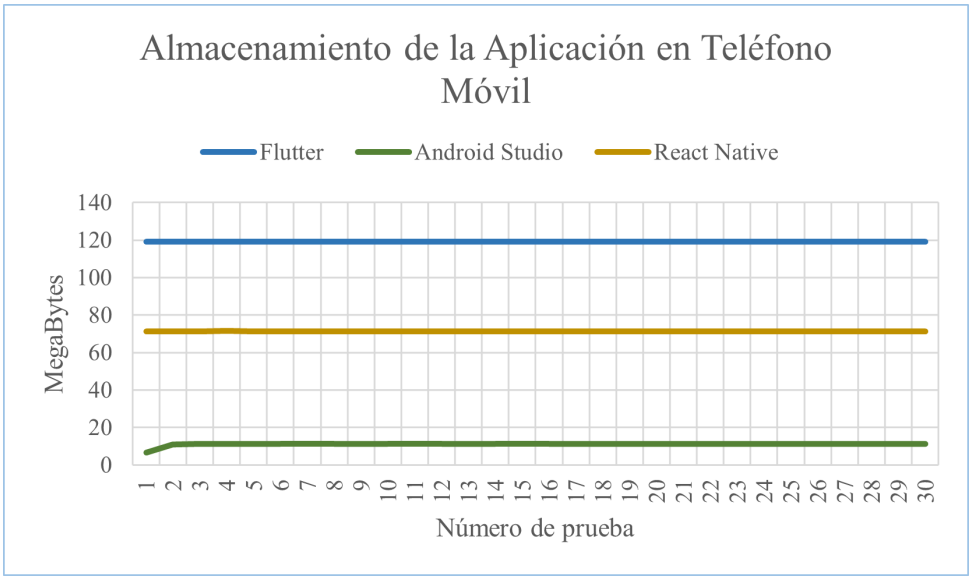


Figura 2.12: Ejecución de aplicaciones almacenamiento en teléfono

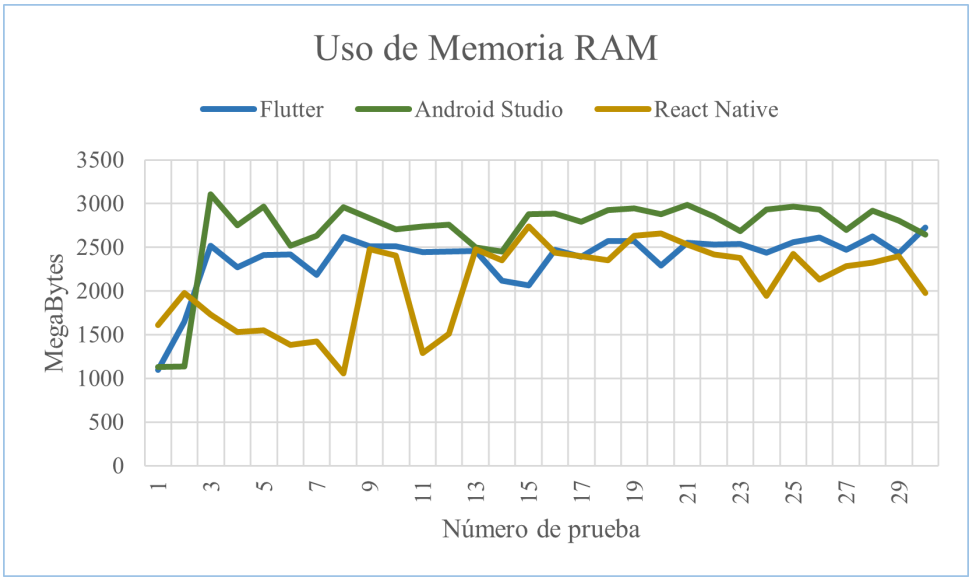


Figura 2.13: Consumo de memoria RAM en primera ejecución

3.- La tecnología de React Native, tarda más en crear la aplicación, consumiendo menos memoria RAM y se optimiza al momento de reservar Memoria ROM en el Teléfono.

Como se puede observar en la Tabla 2.8 la tecnología que atrae en cuestión de ser ágil es Flutter por coincidir con la tecnología nativa en consumo de memoria ram,

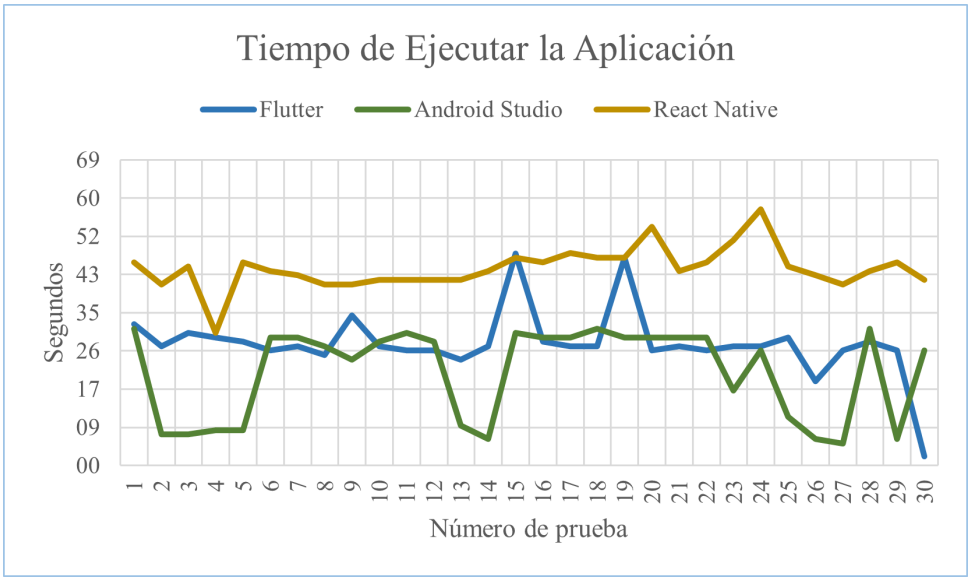


Figura 2.14: Tiempos de ejecución normal de la aplicación

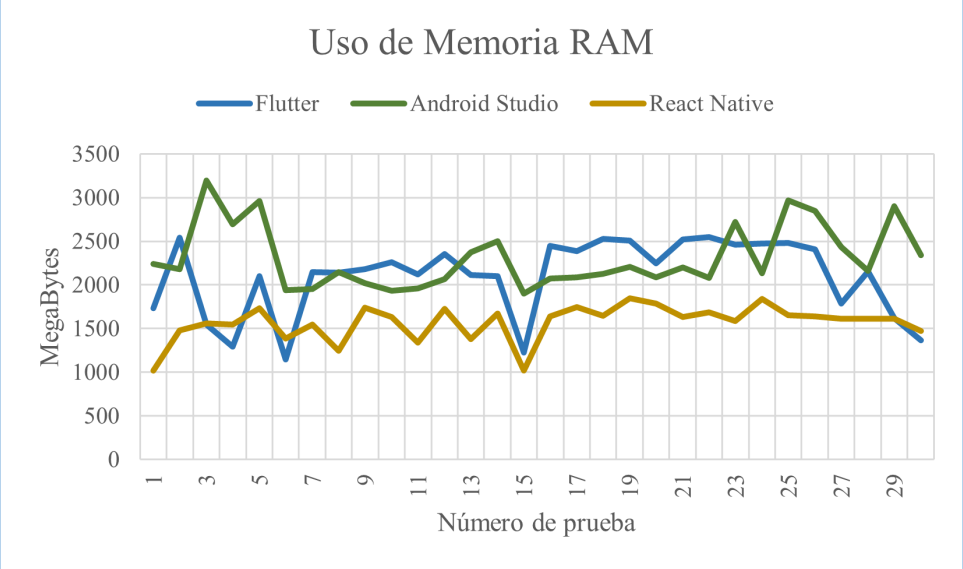


Figura 2.15: Consumos de memoria RAM en ejecución normal

además implementa de mejor manera la tecnología de hot reload y hot restart, estas dos últimas tecnologías agilizan el proceso de desarrollo ya que los cambios se pueden mostrar de manera instantánea; la tecnología que le sigue es React Native debido a que optimiza la memoria RAM, usar la herramienta de hot restart y muestra los cambios en cuestión de segundos, la herramienta de hot restart es una de las tecnologías más

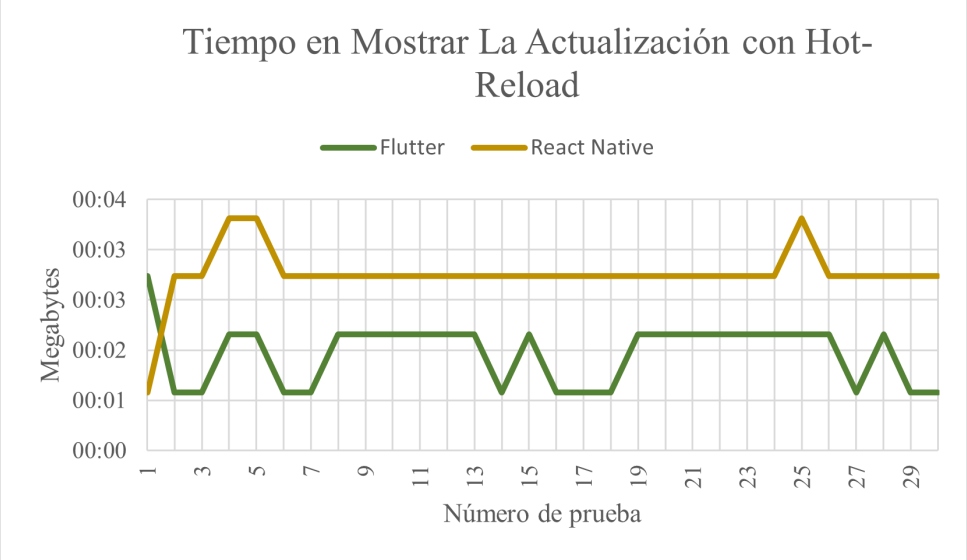


Figura 2.16: Tiempos de ejecución normal hot-reload

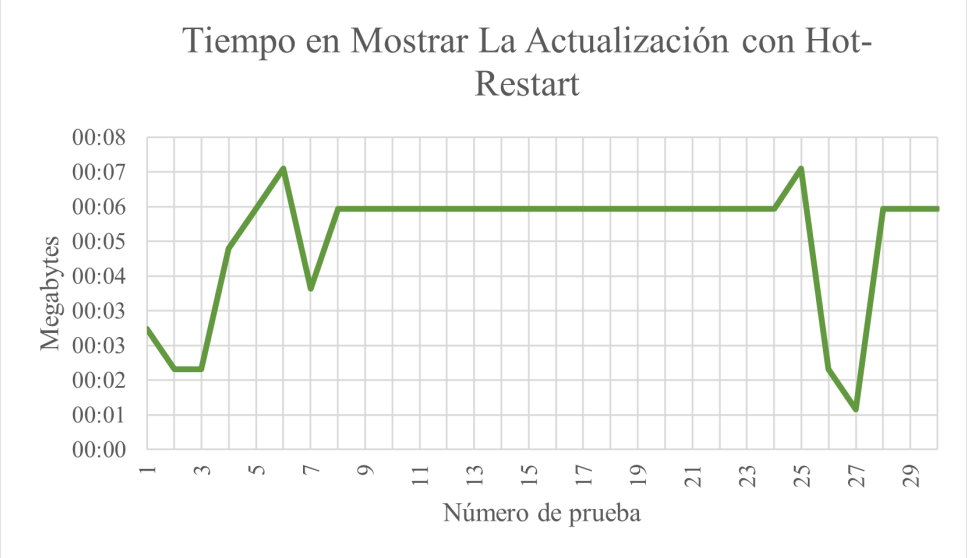


Figura 2.17: Tiempo de ejecución hot-restart

utilizadas en etapa de desarrollo.

A manera de conclusión de las comparaciones se expone un resumen en la Tabla 2.9

Como se pude observar la tecnología que tiene mayor optimización con el hardware es Flutter, con la ayuda del editor de código Visual Studio Code da buenos resultados en tiempo de lectura y procesamiento; además la facilidad de implementar las herramientas

TABLA 2.8: Mediciones de proceso para ejecución de la app en teléfono después de la creación.

Característica	Android Studio	Flutter	React Native
Tiempo de ejecución	11seg. a 31seg	20seg. a 36seg.	40seg. a 50seg.
Uso de memoria RAM en PC	1948MB. a 2678MB	1666MB a 2528MB	1362MB a 1773MB
Hot Reload	No	1seg. a 3seg.	3seg
Hot Restart	No	3seg. a 7seg	No

TABLA 2.9: Resumen de comparación

Característica	Android Studio	Flutter	React Native
Apertura de entorno de desarrollo	— 2 —	— 1 —	— 1 —
Crear aplicación	— 2 —	— 1 —	— 3 —
Primera ejecución	— 1 —	— 2 —	— 3 —
Ejecución normal de la aplicación	— 3 —	— 1 —	— 2 —

de cada tecnología; Para la creación de la aplicación también se realiza de manera más eficiente en comparación con cada tecnología; Al momento de hacer ejecuciones y compilaciones también es más rápido que las otras opciones. La siguiente es Android por la razón ser más ágil el proceso de ejecutar la aplicación por primera vez y ser más rápido que React Native. Por último, La tecnología de React Native es buena al momento de estar viendo las actualizaciones en etapa de desarrollo, también por la facilidad de ser ejecutado en cualquier editor de código ya que lo hace ser amigable con cualquier editor.

2.7 Estructura general de proyectos en las tecnologías de Android, Flutter y React Native

Para poder entender de manera clara los paquetes que generan las tecnologías, se dará una explicación de las estructuras de las aplicaciones que generan Andorid nativo, Flutter y React Native respectivamente.

A continuación se de detalla la estructura básica de la tecnología Android nativo

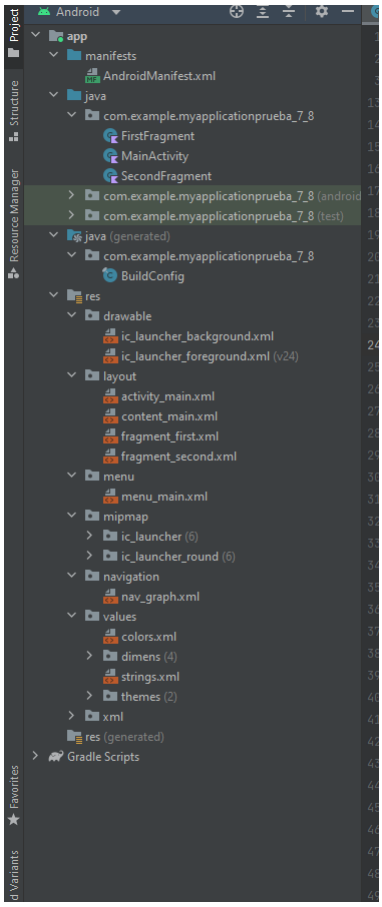


Figura 2.18: Estructura de Proyecto en Android

que se muestra en la Figura 2.18:

- El proyecto se inicia desde la capeta nombrada app, esta carpeta se compone de lo siguiente:
  - Una carpeta llamada manifest que contiene un archivo llamado AndroidManifest.xml en el que se configuran alguna propiedades de la aplicación como: el icono de la aplicación, el nombre de la aplicación o el tema de la aplicación.
  - La carpeta con el nombre de java incluye 3 carpetas con el nombre “com.example.myapplication7.8 ” que se describen a continuación:
    * La carpeta con el nombre “com.example.myapplication7.8 ” contiene los

archivos con extensión “kt ” donde prácticamente va la parte lógica de la aplicación.

* La carpeta con el nombre “com.example.myapplication7.8(androidTest) ”contiene archivos con extinción “kt ” que realiza pruebas instrumentadas, estas pruebas don de manera general, para toda la aplicación.
* La carpeta con el nombre “com.example.myapplication7.8(Test) ” contiene los archivos necesarios para realizar pruebas por cada funcionalidad que se tenga en la aplicación.

– En la carpeta “java(generated)” contiene el archivo“BuildConfig” que guarda las variables de configuración en código de java como puede ser: nombre de la aplicación, Id de la aplicación, versión de código.

– En la carpeta “res” se guardan todos los recursos de la aplicación y se clasifica en las siguientes carpetas:

* “drawable” contiene las figuras dibujadas en código xml.
* “layout” se guardan todas las interfaces gráficas en código xml. Se puede representar de dos maneras: por código y diseño gráfico.
* “menu” se guarda un archivo “menumain.xml” este archivo nos permite crear la interacción de la aplicación.
* “mipmap” se guardan las imágenes con sus respectivas densidades.
* “navigation” contiene el archivo “navgraph.xml” que contiene el orden de navegación en código xml
* “values” guarda el nombre de las variables para poder usar en código xml. Esta carpeta contiene colores, dimensiones y nombre de cada uno de los componentes en xml.

La siguiente estructura es del Framework Flutter. En esta estructura únicamente se hablará de los archivos más importantes para el desarrollo de la aplicación. En la

Figura 2.19 se puede observar cómo está compuesto el proyecto. A continuación se detalla la estructura del proyecto:

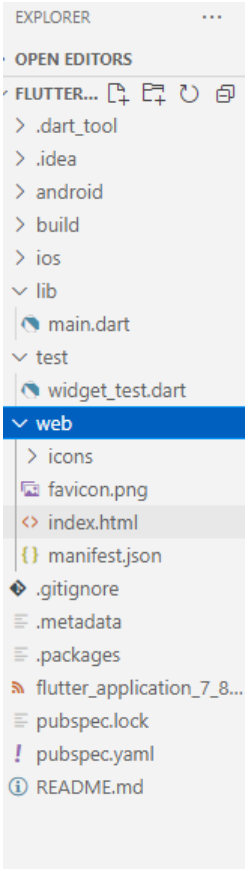


Figura 2.19: Estructura de Proyecto en Flutter

- En la carpeta "Android" se guardan los archivos necesarios que contiene la aplicación en Android nativo para usarse en el momento que se requiera, estas configuraciones pueden ser para utilizar hardware especifico como el gps, implementar Google maps o utilizar la huella digital. Se puede trabajar con Java o Kotlin

- En la carpeta "iOS" contiene los archivos necesarios para configurar la aplicación en el sistema operativo de manera nativa. Se trabaja con Swift.

- En la carpeta "lib" se guardan todos los archivos necesarios para crear la aplicación con la extensión de Dart. Esta carpeta nos permite utilizar el

framework de Flutter. El archivo contiene el nombre "main.dart"que representa el punto inicial de la aplicación.

- En la carpeta "test" se guardaran los archivos necesarios para realizar la pruebas unitarias de la aplicación. Pueden ser paquetes internos o externos. Se recomienda que los test tengan la misma estructura de la aplicación en la carpeta lib.

- En la carpeta "web" se guardan los archivos necesarios para ejecutar la aplicación en un navegador web como puede ser chrome o firefox.

- El archivo ".gitignore" no ayuda para cuando se requiera hacer un repositorio en GitHub no se puedan subir o ignore los archivos o directorios que contenga este archivo, de esta manera nos ayuda a ahorrar espacio en nuestra aplicación.

- El archivo "pubspec.yaml" es importante y delicado porque mantiene toda la información de configuración de la aplicación, como puede ser: versión de SDk de la aplicación, la versión de cupertino que se está ocupando, dependencias de desarrollo, recursos de la aplicación.

- El archivo "README.md" contiene las indicaciones de lo que hace la aplicación o el funcionamiento de la misma.

- El archivo "pubspec.lock" nos dice como esta está construida cada una de las dependencias que se usan en el proyecto.

La ventaja que tiene el uso de este framework es utilizar un solo código en la carpeta lib, este se compila para 4 tecnologías, que son: en Flutter, Android, iOS y Web.

Por último, se explicará la estructura del framework React Native que se muestra en la Figura 2.20

- En la carpeta "tests" se guardarán los códigos necesarios para realizar las pruebas de la aplicación.

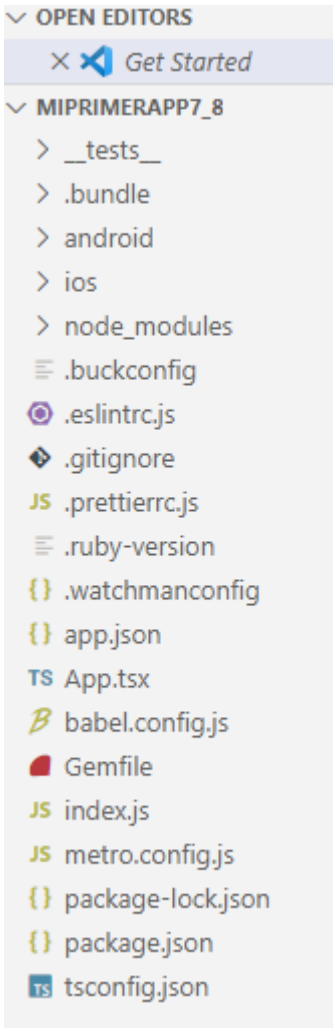


Figura 2.20: Estructura de React Native

- En la carpeta "android" se guardan los archivos requeridos para compilar la aplicación de Android de forma nativa. Los archivos para realizar la lógica de la aplicación vienen precargados en lenguaje java.

- La carpeta "iOS" se encarga de alojar los archivos o carpetas necesarios para ejecutar la aplicación de manera nativa. Únicamente funciona cuando se está compilando en una computadora con sistema operativo Mac-OS.

- En la carpeta "node modules" se encarga de guardar las dependencias de la aplicación, esta carpeta se encarga de alojar cualquier paquete que requiera el

proyecto.

- El archivo “gitignore” nos ayuda a eliminar algunos archivos al momento de alojar la aplicación en GitHub.
- El archivo “App.tsx” contiene el código para realizar la aplicación de React Native.
- El archivo “app.json” contiene información de la aplicación, nombre o sus plataformas soportadas.
- El archivo “package.json” se encarga de guardar la configuración de las dependencias del proyecto.

2.7.1 Conclusión del capítulo

Al analizar las estructuras de cada tecnología podemos entender que la más simple es React Native, por la razón de que al requerir entender la aplicación su estructura no es compleja y no se necesita buscar de manera detallada a cada archivo ya que los archivos principales se encuentran alojados en la parte principal del proyecto.

La siguiente tecnología es Flutter porque sí se requiere un archivo de configuración para agregar dependencias, se encuentra e la raíz del proyecto; si se requiere modificar la interfaz únicamente debemos conocer TyoeScript o JavaScript y conocer un poco de los archivos json , sin embargo si se requiere modificar la interfaz de la aplicación, este se localiza en la carpeta“lib” ; en un archivo llamado “main.dart” y ser cuidadoso con el archivo pubspect.yaml.

Por último tenemos a Andorid nativo, pero su estructura es más compleja por ejemplo si se requiere programar un interfaz la parte lógica, se encuentran en la carpeta “java” en el archivo “mainActivity”, cuenta con un archivo de navegación para el cambio de vistas en la carpeta “navigation” y el texto de la aplicación se encuentra en la carpeta “values ”; que contiene una carpeta llamada “dimens”; que a su vez tiene un archivo “strings.xml”.

2.8 Comparación de aplicación calculadora en cada tecnología

Para poder entender a fondo la comparación de las tecnologías de desarrollo móvil se va a desarrollar una aplicación en Android, Flutter y React Native , esto nos va a permitir profundizar un caso real o similar a cualquier aplicación de tal manera que se pueda realizar un analizáis de sintaxis.

Al realizar una aplicación podemos encontrar los elementos básicos, como puede ser: diseño de una interfaz, procesos que realiza la aplicación, resultados que se pueden obtener del procesado de la información.

La aplicación a crear es una calculadora, esto nos va a permitir dar solución problemas aritméticos básicos, de igual manera nos deja manipular dos componentes de la programación elementales que son Strings y números.

En la Figura 2.21 se muestra la interfaz de la aplicación, el cual se va a dividir en 2 secciones:

- De color rojo se representa los resultados de los números seleccionados: el primer número inferior muestra el número que puede ser editado, el segundo cero muestra el segundo número que se va a utilizar en la operación, arriba de los 2 ceros se encuentra otro componente que permite mostrar las operaciones realizadas anteriormente.

- De color amarillo se encuentran todos los elementos de entrada. La interfaz contiene 20 elementos que permiten interactuar con la aplicación, de los cuales se incorporan 5 operaciones(división, multiplicación , resta, suma y porcentaje), un botón "C" que permite inicializar la calculadora en ceros, el botón "+/-" identifica si el número es negativo o positivo, botón "Del" elimina el último dígito ingresado de el número que se está ingresando, el botón "=" permite realizar la operación con los números ingresados, un punto decimal y los diez números elementales para crear cualquiera.

Figura 2.21: Secciones de Calculadora

2.8.1 Aplicación calculadora en Android

La aplicación fue desarrollada en lenguaje Kotlin con el entorno de desarrollo de Android Studio. Los componentes que se utilizaron fueron: RelativeLayout, LinearLAyout, TextView y Boton. Estos componentes se muestran en la Figura: 2.22 y a continuación se describe el uso que se les dio:

- RelativeLayout va a permitir colocar los elementos empezando desde arriba, utilizando todo el ancho y alto de la pantalla [1].
- 3 TextView que nos va a servir para representar los números de las operaciones [5].
- 6 LinearLayout el tipo de layout que se ocupó 1 vertical para poder realizar las columnas y 5 horizontales para poder realizar filas de los botones [4].

- 20 Botones los cuales se van a encargar de realizar una acción con los elementos de la calculadora.

El total de elementos que se usaron para mostrar únicamente la interfaz son 30.

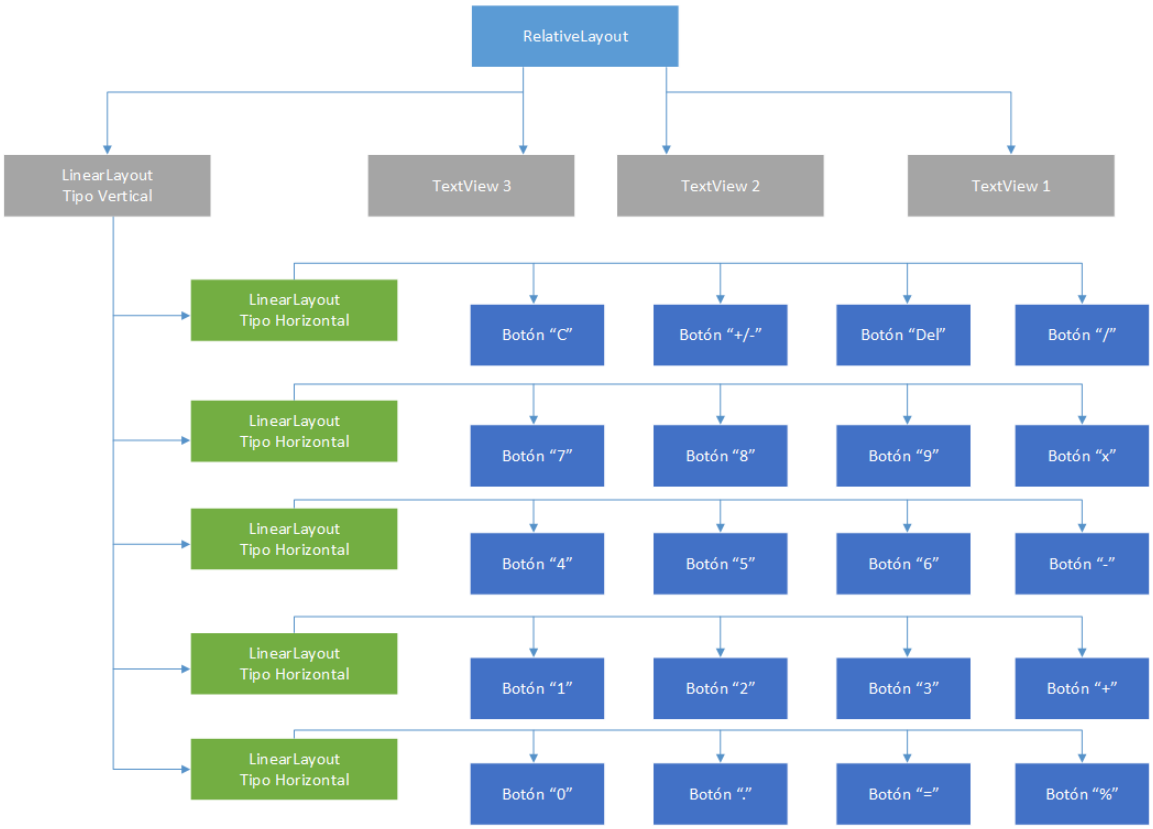


Figura 2.22: Diagrama de componentes de Android

Código de la aplicación en Kotlin

Para poder desarrollar la aplicación, cada elemento de la interfaz en la calculadora se creó uno por uno, ubicados en el archivo "Activity main.xml". el elemento TextView está compuesto por el código que se muestra en el código 2.1.

```
1        android:id="@+id/resultadotext"
2        android:layout_width="match_parent"
3        android:layout_height="wrap_content"
```

```
4         android:layout_above="@+id/num2text"
5         android:layout_marginStart="5dp"
6         android:layout_marginTop="5dp"
7         android:layout_marginEnd="5dp"
8         android:layout_marginBottom="5dp"
9         android:textAlignment="textEnd"
10        android:scrollbars="vertical"
11        android:text=""
12        android:textColor="@color/black"
13        android:textIsSelectable="true"
14        android:textSize="50dp"></TextView>
15    <TextView
```

Código 2.1: Código xml TextView

A continuación se explican las principales líneas del código :

- La línea de código No. 2 se encarga de poner un identificador al texto, dando referencia al elemento para poder utilizarlo.
- La línea No. 3 se encarga de abarcar todo el elemento de forma horizontal.
- En la línea No. 4 se encarga de cubrir el tamaño de lo que va a representar el elemento.
- La línea No. 5 especifica la ubicación del TextView.
- De la línea No. 6 hasta la línea No. 9 se coloca un margen de 5 píxeles.
- La línea No. 10 se encarga de colocar el texto alineado de lado izquierdo.
- La línea No. 11 coloca un ScrollBar para poder deslizar los diferentes resultados al momento que el tamaño del texto sea superior al del TextView.
- La línea No. 12 Coloca el texto a representar en el TextView.

- La línea No. 13 se encargar de colocar en negro el color del texto a mostrar en el TextView.
- LA línea No. 15 se encarga de colocar el tamaño del texto en 50 píxeles.

A continuación se analizará las líneas de código que se usa para crear el botón en el Código 2.2

```
1            <com.google.android.material.button.MaterialButton
2                android:id="@+id/botonmasmenos"
3                style="@style/Widget.MaterialComponents.
4        ExtendedFloatingActionButton"
5                android:layout_width="87dp"
6                android:layout_height="87dp"
7                android:layout_margin="8dp"
8                android:backgroundTint="#D94600"
9                android:text="+/-"
10               android:textColor="@color/white"
11               android:textSize="25dp"
12               app:cornerRadius="48dp">
13           </com.google.android.material.button.MaterialButton>
```

Código 2.2: Código de botón en xml

- La primera línea describe el botón que se ocupa del tipo material.
- La segunda línea se encarga de colocar el identificador del botón. para poder ocuparlo.
- La tercer línea se se establece el estilo del botón de tipo Floating Action button.
- La cuarta línea se encarga de colocar el ancho de botón en 47 píxeles.

- La quinta línea se encarga de establecer la altura del botón en 47 píxeles.
- La sexta línea coloca un margen de 6 píxeles.
- La séptima línea fija el componente de color anaranjado.
- La octava línea coloca el texto del botón.
- La novena línea establece el tamaño del texto en 30 píxeles.
- La décima línea pinta el texto de color blanco.
- La undécima línea se encarga de colocar el las curvas del botón en 48 píxeles.

Los elementos anteriormente mostrados nos ayudan a entender la complejidad de crear y mostrar los 27 elementos de la interfaz gráfica; estos elementos están constituidos por 11,394 caracteres que están distribuidas en 351 líneas de código.

Se analizaran los elementos necesarios para realizar la operaciones en el lenguaje de programación, usando Kotlin.

La función principal para realizar la operación aritmética se muestra en el Código 2.3:

```
1        var num:String=text_nu1.text.toString()
2        var num2:String=text_nu2.text.toString()
3        var numero1:Double = num.toDouble();
4        var numero2:Double = num2.toDouble();
5        var resanterior:String=resu.text.toString()
6        var resultados:Double = 0.0;
7        when (operador) {
8             "+" -> {
9            resultados = numero1 + numero2;
10           text_nu1.setText(resultados.toString())  }
11           "-" -> {
12               resultados = numero2 - numero1
13               text_nu1.setText(resultados.toString())
14           }
```

```
15            "/" -> {
16                if (numero1 == 0.0 && numero2 == 0.0) {
17                     text_nu1.setText("0")
18                } else {
19                    resultados = numero2 / numero1;
20                    text_nu1.setText(resultados.toString())
21                }
22            }
23            "%" -> {
24                resultados = ((numero1 * numero2) / 100)
25                text_nu1.setText(resultados.toString())
26            }
27            "x"-> {
28                resultados = numero1 * numero2;
29                text_nu1.setText(resultados.toString())
30            }
31        }
32        text_nu2.setText("0")
33        if (numero1 != 0.0 && numero2 != 0.0 &&
34    resu.text.toString()== "") {
35            resu.setText(
36                " " + numero2.toString() + " " + operador + " " +
37                        numero1.toString() + "="+
38                        resultados.toString()+" " + resanterior)
39        } else {
40            resu.setText(
41                " "+ numero2.toString() + operador + " " +
42                        numero1.toString() + "=" +
43                        resultados.toString()+", "  +resanterior  )
44        }
45        if ((numero1 == 0.0 && numero2 == 0.0) ||
46            (numero1 == 0.0) || (numero2 == 0.0)) {
47        }
48        if (resu.text.toString().endsWith(".0")) {
```

```
49              resu.setText(resu.text.toString().substring(
50                  resu.text.toString().length - 2)) ;
51              text_nu1.setText(text_nu1.text.toString().substring(
52                  text_nu1.text.toString().length-2))
53          }
54      }
```

Código 2.3: Código Función "resutado"

A continuación se dará una pequeña descripción para entender como se ha construido el código 2.3:

- De la línea 2 a la línea 7 se va a encargar de asignar variables para obtener numero 1 y numero 2, se guardan los resultados anteriores en la variable resanterior y el resultado se inicializa en 0.0
- De 8 a la línea 32 se va a encargar de verificar que operación se va a realizar, dependiendo la opción que se escogió, se va a encargar de realizarla. Para realizar esta operación se utilizo la instrucción "When".
- Desde la línea 34 hasta la línea 52 se evalúan los datos para ver como se muestra el resultado de la operación en pantalla.

El archivo contiene 4,885 caracteres en 224 líneas.

Estos dos archivos fueron principalmente modificados para la realizar la aplicación de calculadora, las características principales de los dos archivos es de un total de 16279 caracteres y 575 líneas.

En el equipo de cómputo donde fue creado, la aplicación tiene un peso de 29.2 MB y una vez instalado en el teléfono ocupa un espacio de 13.96 MB

Analizáis del desarrollo de la aplicación como programador

Al momento de desarrollar la aplicación, las palabra reservadas del lenguaje son muy similares a otros lenguajes de programación como Java y c++, la ayuda que se tenia ocupando Android Studio fue muy favorable, ya que daba sugerencia de que tipo de palabra permite o simplemente te da la opción de auto-completar las sentencias.

El entender la sintaxis del lenguaje es buena ya que al analizar o reconfigurar algún valor, es fácil identificarlo, aunque influye mucho la organización del proyecto.

Al momento de manejar los elementos de la interfaz, los operadores lógicos, aritméticos, condicionales, y métodos iterables; son muy similares a Java y c++. Pero, lo que realmente sobresale, es que Kotlin permite crea sentencias sin tener la obligación de finalizar con el carácter ";" , ya que si se coloca no afecta a la sintaxis del programa y no genera ningún error. Esto habla muy bien del lenguaje ya que sí el programador esta acostumbrado a colocar ese carácter, no afecta; visto desde otro ángulo, sí no esta acostumbrado no afecta, y no se ve obligado a adaptarse a ese tipo de sentencias, que ya tienen mucho tiempo y requiere de practica para adaptarse.

El lenguaje, en comparación con Java, tiene pequeñas diferencias al momento de declarar variables, al manejar ciertas funciones especificas para el manejo de datos de tipo String, como pueden ser: startWith, replacefirst, contains.

El factor que tiene mayor cambio con otros lenguajes al momento que se está aprendiendo a programar es, la estructura del proyecto ya que es compleja cuando se está iniciando. Los datos están ubicados en un directorio, las imágenes están en un directorio, la estructura general de la aplicación está en otro, los colores se encuentran ubicados en otro archivo. Esto no quiere decir que está mal, ya que la estructura de la aplicación está diseñada para ser funcional y debe mantener un mayor orden al momento de estar creando la aplicación. Esta estructura puede ocasionar un pequeño conflicto ya que se puede requerir de tener 3 o más archivos para realizar una configuración y esta se realice en la aplicación.

Al momento de realizar la interfaz los elementos en el archivo XML, los elementos

que crean la interfaz son fáciles de entender, esto es gracias a la comunidad que también ayuda a tener una buena documentación, ya sea oficial o consultando de manera externa.

Existe algo que no se logro identificar para realizar la interfaz, y fue, hacer una estructura para poder crear los botones, se dio, porque existe información que en todos los botones no cambia, los datos se deberían crear de manera automática; y los datos que únicamente cambian, tendrían que ser información que se ingresan al momento de crear la interfaz. Esto ayuda a reducir procesos al momento que se ejecuta la aplicación, estos procesos puede ser mínimo, pero puede servir para otras más complejas.

2.8.2 Desarrollo de aplicación calculadora en Flutter

Al desarrollar la aplicación se ocupo la versión de Flutter 2.12.0, con el Editor Visual Studio Code. Los Widgets que se implementaron son: MaterialApp, Container, Column, ListView, Text, Visivility, Row y Center.

Para facilitar y hacer entendible la construcción de la aplicación, se realizaron dos componentes: Resultados y Botón. El elemento botón está compuesto por un Stateless dos container, Center y Text; el elemento resultados está hecho por, Column, visivility, tres container, tres ListView y tres text. Los componentes se pueden observar en la Figura: 2.23

Como se mencionó en la sección anterior, la mayoría de elementos son Wigets, a continuación se describirá de manera ordenada el uso que se dio a los Wigets anteriormente mencionados:

- El primer componente es Materialapp, permite llamar a una clase llamada "calculadoraScreen" que extiende de un StatelessWidget, esta clase contiene toda la interfaz gráfica de la aplicación.

- Container es un Wiget implementado en la clase "CalculadoraScreen", este widget nos va a permitir usar como contenedor toda la pantalla del dispositivo que ejecute la aplicación, independientemente de su resolución. [3]

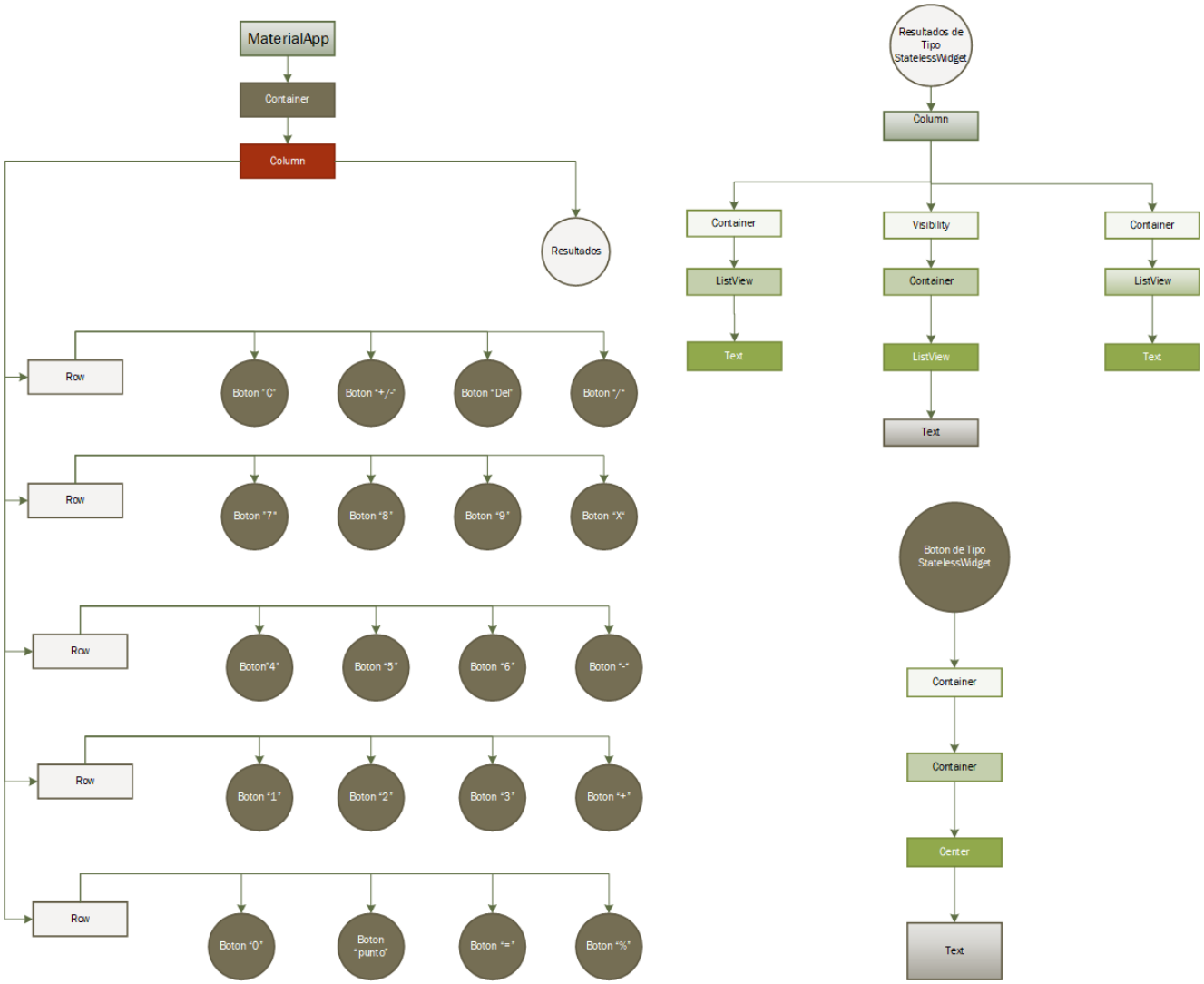


Figura 2.23: Diagrama de componentes de Flutter

- Column es un Widget hijo de Conteiner. Column permite heredar varios hijos, dividiendo el espacio por igual a cada Widget que tenga como hijo de manera horizontal. [2]

- El Widget Resultados es una clase que está compuesta por:

  – 3 Container, estos Widgets sirven como contenedor para mostrar un texto, se uso, porque permite alinear hacia la izquierda al Widget hijo que contenga.

  – Al Widget Visibility es un hijo de container, se uso, para ocultar y mostrar

al Widget que contiene el número guardado al momento de usar la aplicación y realizar la operación; sí el número que contiene el texto es 0 no se muestra y si es diferente de 0 se muestra el Widget. [9]

  - 3 ListView que pueden ser heredado por container o Visibilty, permitiendo hacer scroll para poder mostrar a elementos mas grandes del tamaño establecido.
  - 3 Text que nos permite mostrar texto o números, dando diferentes estilos para poder ser manipulados en la aplicación. [5]

- Se usan 5 row, este Widget permite tener más de un hijo y repartir el espacio por igual de manera vertical.
- Se implementan 20 Botones como hijos de varios row que se extienden de StatelessWidget. Este elemento se construye como clase incluye;
  - 2 Container. El primero se encarga de marcar el espacio ocupado que le toca por el elemento row, el segundo permite darle el tamaño a la figura circular que va a ocupar el botón.
  - Center permite centrar el texto que va a contener el circulo que estar representado como el botón.
  - Text Permite mostrar Texto al botón, puede ser el numero del botón o texto.

Para construir la interfaz de esta aplicación se utilizaron 49 Widgets.

Código de la aplicación en Flutter

Al momento de crear la aplicación "Calculadora " se generaron cinco archivos, esto funciona para poder comprender mejor el código de la aplicación, permitiendo hacer pequeños módulos para que cuando se requiera usar, no se generó código repetitivo. Los archivos son: archivo principal "main.dart", "calculator scren.dart", "math results.dart", "calc boton.dart", "calculator controller.dart".

A continuación se describe las funciones principales de los archivos anteriormente mencionados, la descripción se realizará de manera en como se fue desarrollando la aplicación.

- El archivo principal es "main.dart" ya que requiere de una raíz para iniciar a construir el árbol de Widgets.La función principal de este archivo es poder manejar las ruta que tiene la aplicación, a continuación se muestra el Código 2.4.

```
1  import 'package:calculadora/screens/calculator_screen.dart';
2  import 'package:flutter/material.dart';
3  import 'package:flutter/services.dart';
4
5  void main() => runApp(MyApp());
6
7  class MyApp extends StatelessWidget {
8    @override
9    Widget build(BuildContext context) {
10     SystemChrome.setPreferredOrientations([
11       DeviceOrientation.portraitUp,
12     ]);
13     return MaterialApp(
14       debugShowCheckedModeBanner: false,
15       home: CalculatorScreen(),
16       theme: ThemeData.dark().copyWith(
17     scaffoldBackgroundColor: Colors.black),
18     );
19   }
20 }
```

Código 2.4: Código "main.dart"

- De la línea 1 a la línea 3 se declaran los paquetes que son requeridos para esta clase.
- En la línea 5 se ejecuta el archivo main mandado a ejecutar a la clase MyApp.
- De la línea 7 a la línea 20 se describe la clase MyApp.
- En la línea 10 y 11 se hace una configuración para poder evitar rotar la pantalla.
- La línea 15 manda a llamar a la clase CalculetoScreen.

El archivo cuenta con 458 caracteres y 20 líneas.

- EL archivo "calculator screen.dart" es el archivo que nos va a dejar manipular toda la pantalla del teléfono para crear la interfaz gráfica. El Código 2.5 se muestra para dar una guía de cómo se diseñó la interfaz.

```
1  import 'package:flutter/material.dart';
2  import 'package:get/get.dart';
3  import 'package:calculadora/controllers/calculator_controller.dart';
4  import 'package:calculadora/widgets/math_results.dart';
5  import 'package:calculadora/widgets/calc_boton.dart';
6
7  class CalculatorScreen extends StatelessWidget {
8    final calculatorCtrl = Get.put(CalculatorController());
9
10   @override
11   Widget build(BuildContext context) {
12     return Container(
13       color: Colors.blue[50],
14       padding: EdgeInsets.only(top: 20.0, bottom: 20),
15       child: Column(
16         mainAxisSize: MainAxisSize.min,
17         mainAxisAlignment: MainAxisAlignment.end,
18         children: [
```

```
19          MathResults(),
20          Row(
21            children: [
22              SizedBox(
23                height: 15,
24              ),
25              Expanded(
26                  child: Boton(
27                texto: 'C',
28                color: Color(0xffD94600),
29                onPressed: () => calculatorCtrl.resetAll(),
30              )),
31              Expanded(
32                  child: Boton(
33                texto: '+/-',
34                color: Color(0xffD94600),
35                onPressed: () => calculatorCtrl.negativoPositivo(),
36              )),
37              Expanded(
38                  child: Boton(
39                texto: 'del',
40                color: Color(0xffD94600),
41                onPressed: () => calculatorCtrl.borar(),
42              )),
43              Expanded(
44                  child: Boton(
45                texto: '/',
46                color: Color.fromRGBO(17, 96, 0, 1.0),
47                onPressed: () => calculatorCtrl.operacion('/'),
48              )),
49            ],
50          ),
51          SizedBox(
52            height: 10,
```

```
53            ),
54            Row(
55              children: [
```

Código 2.5: Código CalculatorScreen

- De la línea 1 a la línea 5 se hacen las importaciones necesarias que requiere el archivo.
- El archivo cuenta con 180 líneas de código, y la descripción de la clase es de la línea 11 hasta la línea 180, aunque estas líneas no se muestran porque son repetitivas.
- En la línea 8 se crea una variable calculatorCtrl que se va a encargar de buscar la instancia de la clase "CalculatorControlorer"
- La estructura de la interfaz se basa con el Widget con el nombre Container que retorna en la línea 12.
- El wwidget Column se declara en la línea 15.A partir de Aquí, los componentes ya empiezan a ser visibles, los hijos se empezarán a crear en filas
- El primer elemento a mostrar es una instancia de la clase MathResults en la línea 19, esta clase se detallará más adelante. Su principal característica es mostrar 3 Widget que muestren el Texto de 1 variable cada uno, la primera es el numero a editar, la segunda muestra el numero guardado, y la tercera se encarga de mostrar el resultado de las operaciones que ya se realizaron. Al mismo tiempo la clase se encarga de realizar las operaciones que manden los botones de esta clase.
- Ahora toca construir las primeras filas de Botones, para eso, se explicarán las siguientes lineas de códigos.

* Row es un hijo del Widget Column que se encuentra en la línea 20, se encargara de distribuir los Widgets que tiene dentro de filas, estas filas serán los botones que usa la calculadora; el orden de ordenamiento es de arriba hacia abajo, de izquierda a derecha.
* sizedBox es un widget ubicado en la línea 22, que permite dar un espacio en blanco para que las filas no se muestren iniciando desde el lado izquierdo.
* En la línea 25 el Expanded se utiliza cuando un elemento quiere tener un tamaño especifico sin importar cual es el tamaño del widget padre.
* Boton es un Widget de tipo StatkessWidget que se ubica en las líneas 26 a 30 , donde únicamente se mandan los datos: el texto que tendrá el botón, el color y la función que realizara el botón. Estos botones se repiten 4 veces.

Estos elementos se repiten cinco veces, permitiendo mostrar todo el tablero de color amarillo que se muestra en la Figura 2.21,

El total de líneas que tiene este archivo son 177, 2824 caracteres.

- El archivo "calc boton.dart" se encarga de establecer la estructura del botón, evitando repetir muchas lineas de código que no son obligatorias. En el código 2.6 se muestra el contenido de este archivo

```
1 import 'package:flutter/material.dart';
2
3 class Boton extends StatelessWidget {
4   final Color color;
5   final bool big;
6   final String texto;
7   final Function onPressed;
8
```

```
Boton({
  Key? key,
  color,
  this.big = false,
  required this.texto,
  required this.onPressed,
})   : this.color = color ?? Color(0xff333333),
       super(key: key);

@override
Widget build(BuildContext context) {
  // Button
  final buttonStyle = TextButton.styleFrom(
    backgroundColor: this.color,
    primary: Colors.white,
    shape: CircleBorder(),
  );
  return Container(
    margin: EdgeInsets.only(bottom: 1, right: 5, left: 5),
    child: TextButton(
      style: buttonStyle,
      child: Container(
        width: this.big ? 70 : 70,
        height: 70,
        child: Center(
            child: Text(
          this.texto,
          style: TextStyle(fontSize: 30,
    fontWeight: FontWeight.w300),
        )),
      ),
      onPressed: () => this.onPressed(),
    ),
  );
```

```
43   }
44 }
```

Código 2.6: Código Calc_ buton.dart

- En la línea 1 se importa la librería de material, esta biblioteca incluye la mayoría de widgets de manera oficial por parte de Google.
- De la línea 3 a la línea 47 se construye el cuerpo de la clase Boton.
- De la línea 4 a la línea 7, se declaran las variables color, texto y onPress.
- De la línea 9 hasta la línea 15 se crea el constructor de la clase, esto funciona para decir las variables que son obligatorias con la calase.
- La línea 15 aplica una condición, sí no recibe ningún color el botón se pintará de color negro, este código se puso a prueba por si se requería dejar un espacio.
- De la línea 20 a la línea 23 se crea un estilo del botón, los valores que establece es el color del botón y del texto, el circulo que llevara el botón.
- De la línea 26 a la línea 41 se construye el botón, y coloca todos los valores obtenidos por el Widget padre que usa a este elemento.

Este archivo contiene 42 líneas de código, 710 caracteres.

- El archivo "calculator controler.dart" se encarga de manipular los valores más usados por la aplicación, estos valore son: numero1, numero2, result(resultado) y el operador(guarda el carácter de la operación). Al obtener estos valores, se podrán hacer todas las operaciones aritméticas de la calculadora.

  A continuación, se explicaran algunas partes de código que se concederán son importantes del Código: 2.7

```
1 import 'package:get/get.dart';
2
3 class CalculatorController extends GetxController {
4   var num1 = '0'.obs;
5   var num2 = '0'.obs;
6   var resul = ''.obs;
7   var operador = ''.obs;
8   bool isVisible = false;
```

Código 2.7: Código CalculatorController

– La línea 1 se encarga de cargar los repositorios del paquete Getcontroller, Estos paquetes son requeridos para hacer el manejo de estados en toda la aplicación.

En el Código 2.9 se presenta otra fracción de código, esta fracción se encarga de realizar las operaciones que realizara la calculadora.

```
1   resultado() {
2     double numero1 = double.parse(num1.value);
3     double numero2 = double.parse(num2.value);
4     String resanterior = resul.value;
5     String signo = "";
6     double resultado = 0;
7     switch (operador.value) {
8       case '+':
9         num1.value = '${numero1 + numero2}';
10        resultado = numero1 + numero2;
11        signo = "+";
12        break;
13
```

```
14      case '-':
15        num1.value = '${numero2 - numero1}';
16        resultado = numero2 - numero1;
17        signo = "-";
18        break;
19
20      case '/':
21        if (numero1 == 0 && numero2 == 0) {
22          return num1.value = '0';
23        } else {
24          num1.value = '${numero2 / numero1}';
25          signo = "/";
26          resultado = numero2 / numero1;
27        }
28        break;
29      case '%':
30        num1.value = '${(numero1 * numero2) / 100}';
31        resultado = (numero1 * numero2) / 100;
32        signo = "%";
33        break;
34
35      case 'X':
36        num1.value = '${numero1 * numero2}';
37        resultado = numero1 * numero2;
38        signo = "x";
39        break;
40
41      default:
42        return;
43    }
44    num2.value = '0';
45
46    if (numero1 != 0.0 && numero2 != 0.0 && resul.value == '') {
47      resul.value = " " +
```

```
48          resanterior +
49          ' ' +
50          numero2.toString() +
51          '' +
52          signo +
53          '' +
54          numero1.toString() +
55          '=' +
56          resultado.toString();
57    } else {
58      resul.value = resanterior +
59          ', ' +
60          numero2.toString() +
61          signo +
62          '' +
63          numero1.toString() +
64          '=' +
65          resultado.toString();
66    }
67    if ((numero1 == 0.0 && numero2 == 0.0) ||
68        (numero1 == 0.0) ||
69        (numero2 == 0.0)) {
70      resul.value = resul.value;
71    }
72
73    if (resul.value.endsWith('.0')) {
74      resul.value = resul.value.substring(0, resul.value.length - 2);
75    }
76    funcionVisible();
77  }
```

Código 2.8: Código función "resultado"

- De la línea 1 hasta la línea 77 se crea el cuerpo de la función resultado.
- La línea 2 y 3 se encargan de cargar los valores que tienen las variables del controlador.
- De la línea 4 a 6 se encarga de restablecer los valores iniciales para poder realizar otra operación.
- De la línea 7 a la 43 se encarga de verificar que función se está realizando, la validación se hace mediante signos de cada operación. Posteriormente realizara la operación seleccionada en las Variables: "num1", "resultado" y "signo".
- En la línea 44 el valor de la variable "num2" se reinicia a 0 para que pueda realizar otra operación en el caso de que se requiera.
- En la línea 46 a la 57, se encarga de verificar que existan 2 valores para realizar la operación requerida y sí la variable "resul1" está vacía, se colocan los datos de la operación y muestra el resultado en la variable "resul1". De lo contrario, también se almacenan los datos de la operación pero ahora se coloca una "," para separar cada operación.
- De la línea 67 a la 71, se encarga de verificar si un número falta la operación no se muestra y se queda el valor anterior.
- La línea 73 a la 75 analiza, sí la variable "resul1" termina en ".0"; entonces, elimina los últimos 2 dígitos.
- La línea 76 se encarga de llamar a una función llamada "funcionVisible" donde evalúa, sí la variable "num2"=0 entonces oculta el widget, de lo contrario la muestra.

El archivo "calculator controler.dart" incluye 171 líneas de código y 2679 caracteres.

El último archivo que se creo es "math results.dart" este archivo se va a encargar de mostrar en pantalla los relatados de las variables que se obtienen en el archivo

"calculator controler.dart". El Código 2.9 Básicamente se encarga de dibujar la interfaz que se encuentra de color rojo en la Figura: 2.21.

```
import 'dart:ui';
import 'package:calculadora/controllers/calculator_controller.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';

class MathResults extends StatelessWidget {
  final calculatorCtrl = Get.find<CalculatorController>();

  @override
  Widget build(BuildContext context) {
    return Obx(() => Column(
          children: [
            Container(
              // color: Color(0xfffefcf3),
              height: 140,
              alignment: Alignment.bottomRight,
              child: ListView(
                reverse: true,
                children: [
                  Text(
                    '${calculatorCtrl.resul}',
                    textAlign: TextAlign.end,
                    style: TextStyle(
                        decoration: TextDecoration.none,
                        color: Colors.black54,
                        fontSize: 35),
                  ),
                ],
              ),
            ),
```

```
32            Visibility(
33              visible: calculatorCtrl.isVisible,
34              child: Container(
35                  alignment: Alignment.bottomRight,
36                  height: 90,
37                  child: ListView(
38                    reverse: true,
39                    children: [
40                      Text(
41                        '${calculatorCtrl.num2}',
42                        textAlign: TextAlign.end,
43                        style: TextStyle(
44                            decoration: TextDecoration.none,
45                            color: Colors.black54),
46                      ),
47                    ],
48                  )),
49            ),
50            Container(
51                alignment: Alignment.bottomRight,
52                height: 88,
53                child: ListView(
54                  reverse: true,
55                  children: [
56                    Text(
57                      '${calculatorCtrl.num1}',
58                      textAlign: TextAlign.end,
59                      style: TextStyle(
60                          decoration: TextDecoration.none,
61                          color: Colors.black54),
62                    ),
63                  ],
64                )),
65          ],
```

```
66          ));
67    }
68 }
```

Código 2.9: Código Función "resultado"

- De la línea 1 a la línea 5 se importan los paquetes que requiere el funcionamiento de la clase.
- de la línea 7 a la línea 68 se declara el cuerpo de la clase MathResults donde básicamente es de tipo StatelessWidget.
- En la línea 8 se declara la variable calculatorCtrl,se encarga de buscar la instancia del tipo "CalculatorController" y ocupa los valores que este contenga, haciendo referencia al Nombre de la variable que se desea ocupar.
- En esta clase únicamente se regresa un widget de tipo Column, recordemos que este tipo de widget puede tener varios hijos.
- De la línea 14 a la línea 31 se construye todo lo que va a contener el Widget para mostrarlo en la interfaz, El texto que va a estar demostrando es la variable "resul" que se ubica en el archivo "calculator,controller,dart".
- La línea 32 a la línea 49 es un Widget diferente, es del tipo Visiibility, este widget contienene la propiedad "visible" de tipo boolean que le permite mostrarse o no, también representa el valor de la variable "num2"; esas variables son manejadas en el archivo "calculator controller.dart".
- De la línea 50 a la 54 se encarga de construir un Container para mostrar el valor de la variable "num1" que se encuentra en el archivo "calculator controller.dart"

El archivo "math results.dart" contiene 114 caracteres en 68 líneas de código.

Como pudimos observar la calculadora se distribuyó en 5 archivos diferentes, en total se usaron 6331 caracteres en 418 líneas, con un peso en computadora de 665MB, y la aplicación ya instalada en el teléfono tiene un peso de 148MB.

Análisis del desarrollo de la aplicación como programador

Para poder crear esta aplicación de una forma cómoda y con mayor orden, fue necesario dividirlo en diferentes archivos, esta técnica puede generar más líneas de código, pero, al momento que se usa el código es cómodo de entender y facilita a ser modificado ya sea por el creador o por algún otro programador.

El lenguaje Dart es fácil de entender, con la ayuda de visual Studio Code, realmente ayuda a que el código sea comprendido de una manera rápida y cómoda. Realmente el lenguaje no tiene problema, existe mucha documentación que ayuda a resolver diferentes problemas que se tengan, la complejidad viene cuando se intenta ocupar el framework de Flutter, ya que existen una cantidad grande de widgets y debes de tomar la decocción de cual quieres desarrollar o que widgets son, lo que se recomienda en estos casos es empezar desde los básicos y posteriormente conocer otros widgets más avanzados, en Flutter se puede entender muy bien que una interfaz no es una receta, y que todas deben de ser iguales, existen otros componentes que resulta ser mejor, igual o peor.

En Flutter se considera que debería de ser obligatorio dividir los archivos, porque se usan varios widgets al momento que se crea una aplicación. Es seguro que, al desarrollar esa aplicación, el árbol de widgets sea extenso, entonces eso ocasiona que un programador pueda tardar en entender la interfaz del programa.

Una vez que entiendes los widgets, la aplicación resulta ser un problema común y no presenta mayor problema para resolverlo. en este ejemplo de calculadora se tomó la decisión de ocupar un manejador de estados, para que los datos de las variables se pudieran implementar desde cualquier lugar de la aplicación, evitando que se generara más de una instancia y se pudieran perder los datos.

La sintaxis del lenguaje no resulta ser compleja para usarse, ni incomprensible para

una persona que apenas está iniciando, las palabras reservadas son similares a la de los lenguajes más usados como javaScript, c++, o Java, pero definitivamente se debe de ocupar uno de terminar las líneas con punto y coma, ya que sí no se hace, puede causar un error de sintaxis.

Para desarrollar esta aplicación únicamente se utiliza un lenguaje y un framework, la interfaz y la comunicación de los elementos entre clases buena, sí se deseará crear una interfaz dinámica es realmente sencillo crearla ya que se construye un modelo donde únicamente se repite el elemento. Como se mencionó anteriormente los widgets son complejos, pero la manipulación de la información es sencilla al momento de crear las aplicaciones.

Algunas de las ventajas más importantes de Flutter es que permite a la aplicación ejecutarse en un teléfono con Android o Iphone y en web. Lo único que se requiere es ejecutar la aplicación en el lugar que se requiera ocupar.

En la Figura: 2.24 se muestra cómo se ejecuta la aplicación en un teléfono móvil con el sistema operativo Android y en la Figura 2.25 se muestra la ejecución en un navegador Web, como si se tratara de una página cualquiera.

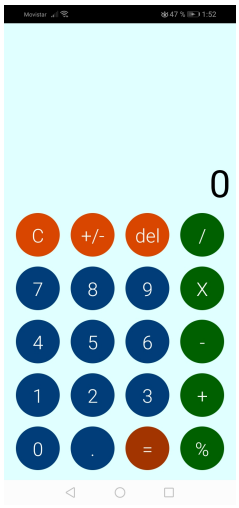


Figura 2.24: Calculadora ejecutando en Android

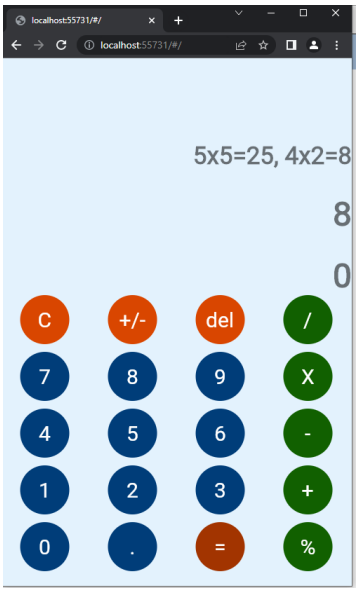


Figura 2.25: Calculadora ejecutando en web

2.8.3 Desarrollo de aplicación Calculadora en React Native

La aplicación esta desarrollada en React Native 0.69.1 con TypeScript 4.4.4, el editor de código que se implemento fue Visual Studio Code. En React Native los elementos que se utilizan son nombrados como "componentes". Los componentes que se implementaron son: SafeAreaView, View, Text, View(tipo row), touchableOpacity.

La aplicación requiere que se haga un componente de tipo boton, este componente contiene un TouchableOpacity, un View y un Text para poder dibujar el botón en la interfaz; un archivo, a continuación se muestra el árbol de componentes que se implementaron para la creación de esta aplicación en la Figura: 2.26

El uso que se dio a cada componente es:

- View Es un componente de tipo contenedor que permite mostrar otro tipo de componentes en la interfaz gráfica de la aplicación. [8]
- Text es un componente que permite manipular el texto de la aplicación en el tiempo de ejecución. [6]
- View(tipo row) permite hacer contenedores mediante filas en lo ancho de la

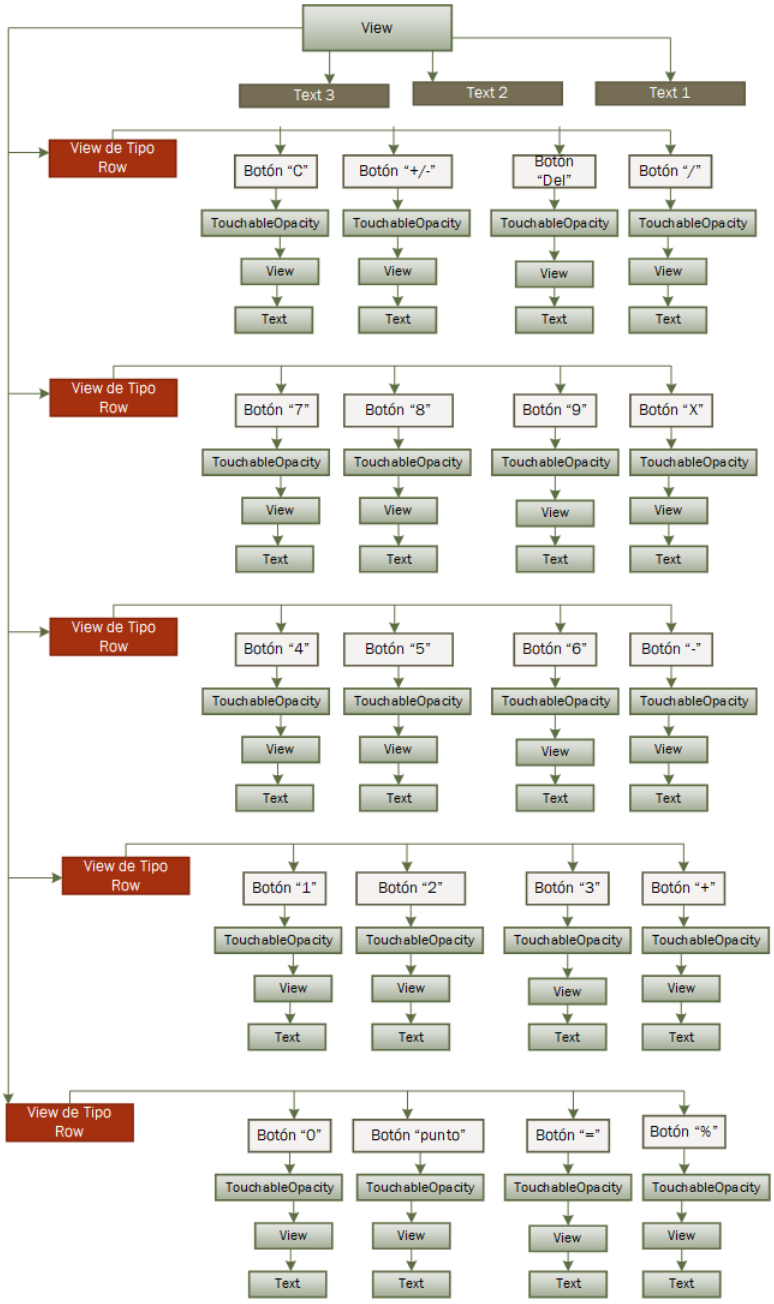


Figura 2.26: Diagrama de componentes de React Native

pantalla.

- touchableOpacity se encarga de habilitar la opción OnnPress que se encarga de reaccionar al momento que se le da clic al componente padre. [7]

El total de componentes que se utilizaron en esta interfaz son 85.

Código de la aplicación en React Native

Al crear esta aplicación se generaron 4 archivos:

- El primer archivo se encarga de crear toda la interfaz en un componente llamado SafeareView, este componente se va a encargar de dibujar cualquier otro componente que se encuentre en su interior. Este elemento manda a llamar al archivo "CalculadoraScreen.tsx",
- El archivo CalculadoraScreen.tsx se encarga de dibujar y manipular los datos de la interfaz gráfica de la aplicación.
- El archivo "tema de app.tsx" se encarga de establecer los estilos de la interfaz para que únicamente se ocupen en el archivo " CalculatorScreen tsx".
- El archivo botoonCalculadora,tsx se va a encargar de crear un componente personalizado que permite ser manipulado al antojo del programador, este archivo sirve para establecer la estructura de la interfaz del botón.

A continuación, se describen el funcionamiento elemental que tienen los 4 archivos de la aplicación calculadora hecha en react native con TypeScript:

- El archivo principal es "App.tsx". Este archivo se encarga de crear la función principal para la ejecución de la aplicación. A continuación se muestra el Código 2.12

```
1 import React from "react";
2 import { SafeAreaView, StatusBar, Text, View } from "react-native"
3 import { CalculadoraScreen } from "./src/screens/CalculadoraScreen";
4 import { styles } from "./src/temas/tema_de_app";
5
6 const App = () => {
7   return(
8     <SafeAreaView style={styles.fondo}>
```

```
9       <StatusBar
10        backgroundColor='black'
11        barStyle='light-content'
12      />
13      <CalculadoraScreen />
14    </SafeAreaView>
15
16  )
17 }
18
19 export default App;
```

Código 2.10: Código "App.tsx"

- De la línea 1 a la línea 4 se encarga de realizar las importaciones que necesita el archivo para poder ser ejecutado.
- De la línea 6 a la línea 17 se construye el cuerpo de la función App.
- En la línea 8 se crea un Componente SafeAreaView y se coloca un estilo de fondo negro, el estilo se encuentra estructurado en el archivo "tema de app.tsx"
- De la línea 9 a la línea 12, se colocó el componente StatusBar porque en algunas capas de personalizadas que tiene Android, la barra de estados puede ser blanca, mediante esas líneas de código se obliga a que sea negra.
- La línea 13 se encarga de llamar a la interfaz que se encuentra en el archivo "CalculadoraScreen.tsx".

El archivo "App.tsx" contiene 356 caracteres en 19 líneas de código

- Archivo "BotonCalculadora.tsx". Su función principal es crear un componente personalizado que se ocupe más de una vez. Esto ayuda a reducir algunas líneas

de código en los archivos. A continuación, se muestra el contenido del Código 2.11:

```
1  import React from "react";
2  import { SafeAreaView, StatusBar, Text, View } from "react-native"
3  import { CalculadoraScreen } from "./src/screens/CalculadoraScreen";
4  import { styles } from "./src/temas/tema_de_app";
5
6  const App = () => {
7    return(
8      <SafeAreaView style={styles.fondo}>
9        <StatusBar
10         backgroundColor='black'
11         barStyle='light-content'
12       />
13       <CalculadoraScreen />
14     </SafeAreaView>
15
16   )
17 }
18
19 export default App;
```

Código 2.11: Código "BotonCalculadora"

- De la línea 1 a la línea 3 se encarga de realizar todas las importaciones necesarias de los archivos
- De la línea 5 a la línea 11 se crea una interfaz que contiene las variables que necesita el botón para se construido. Las variables son: Texto, color, accion, y numboton.

– De la línea 13 a la línea 29 se crea la estructura que tendrá el botón. El tipo de componente es de tipo TouchableOpacity, debido a que permite reaccionar como botón gracias a su método OnPress. Posteriormente se asignan los parámetros como: función que realiza el botón, el texto que representara el botón y el color del botón. En el Código 2.12 se muestra la construcción del estilos del botón que se encuentra en el archivo "tema de app.tsx"

```
1    boton:{
2        height:80,
3        width:80,
4        backgroundColor:'#2D2D2D',
5        borderRadius:100,
6        justifyContent:'center',
7        marginHorizontal:10,
8    },
9    textoBoton:{
10       textAlign:'center',
11       padding:10,
12       fontSize:30,
13       fontWeight:'300',
14       color:'white'
15   },
```

Código 2.12: Código "App.tsx"

* De la línea 1 a la 8 se crea un estilo para el botón.
* En la línea 2 y 3 se establece el tamaño del botón.
* La línea 4 establece el color que tendrá el botón sí no se manda ningún color.
* La línea 5 se encarga de colocar dibujar el circulo para que se muestre

como botón.

* De la línea 9 a la línea 15 se crea un estilo para dar un formato al texto que se va a mostrar en el botón,
* La línea 10 se encarga de centrar el texto al tamaño del componente.
* En la línea 11 se da un espacio de 10 para que se cree un pequeño margen y no dibuje desde el lado izquierdo del botón.
* La línea 12 se establece el tamaño de la letra del texto.
* La línea 13 se encarga de establecer el grueso que dibujaran los caracteres para mostrar el texto.
* En la línea 14 se coloca el color que se colocara al texto del botón.

El archivo "BotonCalculadora.tsx" este compuesto por 502 caracteres en 31 líneas de código.

- Archivo "CalculadoraSceren.tsx". Es el núcleo de la aplicación, ya que este archivo se encarga de dibujar la interfaz, realizar operaciones, cambiar los estados de cada componente para que se muestren actualizados en todo momento. En el Código 2.13 se presentan algunas secciones del archivo para ser descritos y facilitar la comprensión del funcionamiento:

```
1 import React, { useRef, useState } from "react";
2 import { View, Text } from 'react-native';
3 import { BotonCalculadora } from "../componentes/BotonCalculadora";
4 import { styles } from '../temas/tema_de_app';
```

Código 2.13: Código CalculadoraScreen "importaciones"

  - De la línea 1 a la línea 4 del código 2.13 se encarga de importar los paquetes necesarios para tener un correcto funcionamiento.

```
1  const [numero ,setNumero]=useState('0');
2  const [numeroOperacion ,setNumeroOperacion]=useState('0');
3  const [operacion,setOperacion]=useState('');
4  const tipoOpercion =useRef<operdores>()
```

Código 2.14: Código CalculadoraScreen "Asignaciones"

– En la sección de código 2.14 , de la linea a 4 se declaran las variables, en este caso se uso un manejador de estados que provee React Native.

```
1  }
2
3  return(
4   <View >
5        <Text style={styles.contenidoOperacion}numberOfLines={3}
6           adjustsFontSizeToFit>{operacion}</Text>
7        {
8           (numeroOperacion !=='0')&&(<Text style={
9              styles.resultado}numberOfLines={1}
10             adjustsFontSizeToFit>{numeroOperacion}</Text>)
11       }
12
13
14       <Text style={styles.resultado}numberOfLines={1}
15          adjustsFontSizeToFit>{numero}</Text>
16
17       <View style={styles.fila}>
18          <BotonCalculadora
19          texto="C" color="#D94600" accion={limpiar}/>
20          <BotonCalculadora
```

```
21              texto="+/-"color="#D94600" accion={pososi tivoNegativo}/>
22              <BotonCalculadora
23              texto="del"color="#D94600" accion={btnDel}/>
24              <BotonCalculadora
25              texto="/"  color="#006000" accion={btnSividir}/>
```

Código 2.15: Código calculadoraScreen Interfaz Resultados

Se dará una breve explicación Código 2.15

- En las líneas 3 y 4 se declara un componente de tipo Text con un número máximo de 3 líneas para pintar el texto, y ajustar el texto al tamaño del componente de manera automática. La variable que representa es ”operación”.
- De la línea 6 a 8 se crea una condición, sí el número es diferente de ”0” muestre el resultado en el componente Text con la variable ”numeroOperacion”, Dependiendo del contenido de texto que tenga, se va a estar ajustando el tamaño de la letra al tamaño que tiene el componente Text. El número máximo de líneas es 1 para este elemento.
- De la línea 12 a la línea 13 se dibuja el ultimo Text con un ajuste automático de texto, un numero máximo de líneas de 1 y la variable que representa es ”numero”.
- En la línea 15 se declara un View para que los componentes los meta ordenen al tipo row, esto sirve para dibujar una fila de botones de la calculadora.
- En la línea 16 y 17 se crea el botón ”c” con el color naranja y ejecutando la función ”limpiar”.
- En la línea 18 y 19 se crea el botón ”+*” con color naranja y la función que realiza es ”positivonegativo”.

- En la línea 20 y 21 se crea el botón "del" de color naranja con la función "btndel".
- En la línea 22 y 23 se crea el botón "/" de color verde con la función "btnSdividir". Hasta ahí tenemos la primera fila de botones, esto se repite 4 veces para construir todo el tablero de la interfaz. Cabe notar, las líneas que se repiten son de la 15 a la 24.

En el código se analiza la función calcular, esta función permite realizar la operación y únicamente se manda a llamar cuando se aprieta el botón "=".

```
1  const calcular=() =>{
2     const numero1=Number(numero);
3     const numero2=Number(numeroOperacion);
4     const textoGuardado=operacion;
5     var signo="";
6     var resultado;
7     switch (tipoOpercion.current) {
8        case operdores.sumar:
9            setNumero(`${ numero1 + numero2 }` );
10           resultado= numero1+numero2;
11             signo=" + ";
12          break;
13       case operdores.restar:
14          setNumero(`${numero2 - numero1}`);
15          signo=" - ";
16          resultado= numero2-numero1;
17          break;
18       case operdores.multiplicar:
19          setNumero(`${numero1 * numero2}`);
20          resultado= numero1 * numero2
21          signo=" x ";
22          break;
23       case operdores.porcentaje:
```

```
24            setNumero(`${(numero1 * numero2)/100}`);
25            resultado= (numero1 * numero2)/100
26            signo=" %";
27            break;
28         case operdores.dividir:
29            if( numero1 === 0 && numero2 === 0) {
30                  return setNumero('0');
31              } else{
32               setNumero(`${numero2 / numero1}`);
33               signo=" / ";
34               resultado= numero;
35               break;
36            }
37
38         default:
39            break;
40      }setNumeroOperacion('0');
41      if (numero1!=0 && numero2!=0 &&operacion=='') {
42         setOperacion(' '+operacion+numero2+ signo+''
43         + numero1+'='+resultado+' ');
44      }else{
45         setOperacion(operacion+', '+numero2 +signo+ ' '
46         +numero1+'='+resultado+' ');
47      }
48      if ((numero1==0 && numero2==0 )||(numero1==0)||(numero2==0)) {
49         setOperacion(operacion);
50      }
51   }
```

Código 2.16: Código función "calcular"

– De la línea 1 a la 51 se crea el cuerpo de la función "calcular"

- De la línea 2 a la línea 6 se asigna el valor a las variables que realizaran las operaciones, En este caso son: numero1, numero2, textoguardado y resultado.
- En la línea 7, se crea una sentencia de tipo switch que permite evaluar el tipo de función que se va a realizar, dependiendo de la operación que se escoja, se ejecutaran diferentes secciones de código que permitan resolver la operación indicada.

  Nota: Todos los valores se guardan en la variable resultado y se muestran en la variable número

  * De la línea 8 a la 12 se realizan las sentencias para que se sumen los numero1 y numero2.
  * De la línea 13 a 17 se muestran las sentencias para realizar la operación resta, esta sentencias se hace "numeor2 - numero1".
  * De la línea 18 a la línea 22 se encaran de construir la opción multiplicar, multiplicando numero1 * numero 2.
  * De la línea 23 a la línea 27 se crean las sentencias para realizar la operación porcentaje, esta opción multiplica el numero1 * numero2 y luego lo divide entre 100.
  * De la línea 28 a la 26 se colocan las sentencias para realizar la división.
  * En la línea 29 a 36 se evalúa, sí los dos números son "0", entonces coloca la variable numero con el valor "0". De lo contrario se divide el numero2 con el numero1.
  * De la línea 40 se coloca el valor de la variable numeroOperacion en "0".
  * De la línea 41 a la 47 verifica que las variables no estén vacías, Sí es la primera operación, únicamente coloca en texto todos los datos. De lo contrario concatena los datos de las operaciones anteriores y coloca los nuevos después.

El archivo "CalculadoraScreen.tsx" contiene 4624 caracteres en 216 líneas de código.

En total se usaron 4 archivos, que en total contienen 6168 caracteres en 320 líneas de código. El peso de la aplicación en el equipo de cómputo es de 802MB y en el Dispositivo móvil ocupa 120MB

Analizáis del desarrollo de la aplicación como programador

Al momento que se desarrolló la aplicación se entendió de manera favorable con la sintaxis de otros lenguajes, un pequeño inconveniente fueron algunos operadores de comparación como "===", las funciones para tratar los datos de tipo String son poco diferentes con la de otros lenguajes, pero rápidamente se encuentra documentación para realizarlos. Las palabras reservadas están definidas de manera correcta.

La facilidad para encontrar algunos detalles de ocultar componente fue muy buena, ya que no requiere de hacer configuraciones largas en determinados puntos, realmente las sentencias son cortas y son muy fáciles de implementarlas. Como tecnología es muy amigable para el programador.

El punto y coma es opcional para este lenguaje y no requiere de adaptación para ser usado.

Los componentes para crear la interfaz son fáciles de implementar. Los estilos son fáciles de aplicar, pero requiere de un poco de practica para acostumbrase a saber dónde colocarlos. Únicamente se trabaja con archivos tsx, y no tendrás problema para realizar la interfaz en un lenguaje y el arte lógica en otro lenguaje.

Al principio cuesta trabajo encontrar documentación de manera oficial para poder documentarse en React Native pero ya cuando se interactúa con el código es muy sencillo. Al momento de desarrollar esta aplicación se detectó que el framework no está actualizado a las últimas versiones de Android. Se desconoce el ¿por

qué? pero si preocupa que no hagan actualizaciones importantes para este tipo de tecnologías.

2.8.4 Comparativa de aplicación

Una vez desarrolladas las aplicaciones, se analizaran los datos que arroja cada aplicación.

TABLA 2.10: Características de Aplicaciones

Característica	Android Studio	Flutter	React Native
Archivos manipulados	2	5	4
Caracteres	16279	6331	6168
Lineas de código	575	418	320
Almacenamiento en teléfono	13.96MB	148MB	120MB
Peso en equipo de cómputo	29.2MB	665MB	802MB

Analizando los datos de la Tabla 2.10 se generan estos puntos de vista:

- En la aplicación desarrollada en Android, su código está optimizado para ocupar el mínimo espacio tanto en el equipo de cómputo como en el teléfono, el problema es que se tiene que escribir más código que en las otras tecnologías. Como mínimo se tocan 2 archivos con diferentes lenguajes, esto requiere de una curva de aprendizaje extensa.

- En Flutter se generaron 5 archivos, pero fue para optimizar las líneas de código. Se puede decir que es fácil de manipular la aplicación a capricho del programador. Es la segunda tecnología que tiene los caracteres más elevados, y eso se debe a que se implementó una herramienta para manejar estados en varios archivos. Su almacenamiento es exageradamente elevado comparado con Android nativo, se podría justificar un poco por las tecnologías que soporta como son: Android, IOs y Web. Se considera que esta aplicación fue la más compleja en realizar.

- En React Native la aplicación fui sencilla de desarrollar, es la segunda tecnología que usa pocos elementos para la interfaz. La estructura de lenguaje es fácil de

manipular para obtener buenos resultados en una aplicación. Que incluso, puede concluir que fue la aplicación más sencilla que las otras 2. El almacenamiento que tiene es el más elevado de todas las tecnologías.

Al momento de ejecutar la aplicación, la fluidez de la aplicación entre flutter y Android Nativo es similar; ya que al tocar varios botones se escucha y se ve muy similar la aplicación. y React Native se siente como la aplicación más lenta; ya que, al tocar los botones, no se percibe del todo bien la animación de los botones o el sonido, pero al momento de escribir no tiene mayor problema, ningún dato fue omitido por la ejecución rápida de la aplicación.

El orden de la comodidad para desarrollar la aplicación es 1.- React Native, 2.-Flutter y 3 Android Nativo.

Capítulo 3

Conclusiones

En este ensayo se ha hecho un estudio de las tecnologías de desarrollo móvil por secciones:

- En el capitulo 2 en la sección 2.1 se establecieron las tecnologías más importantes para desarrollo móvil, las cuales fueron: Andorid Nativo usando el lenguaje Kotlin, React Native implementando TypeScript y Flutter con Dart. Al mismo tiempo se buscó que editores de código se tenían que implementar para poder desarrollar las aplicaciones de manera cómoda. Clasificación por oferta laboral:
  - Como se puede observar en la sección 2.1 La tecnología que tiene mayor empleo es Android nativo con 8.3 %, se debe a que es la principal tecnología desarrollada por Google, eso definitivamente da un buen soporte para actualizar aplicaciones a futuro y no es muy compleja actualizarlas. Por eso las empresas se fijan es este tipo de tecnología.
  - La siguiente es React Native por la facilidad de implementar código, el soporte que da la empresa Facebook, pero en la actualidad tiene un detalle y es que ya no se sacan actualizaciones de componentes de manera oficial para realizar aplicaciones. Existen paquetes de terceros que lo permiten, pero al momento que react native saca una actualización la mayoría de paquetes ya no funciona. Y eso se ve afectado.

  - Flutter es una de las tecnologías más usadas a nivel mundial por la facilidad de manipular el framework, se agradece que Google apoye este tipo de tecnologías y no se enfoque en una sola. Esta tecnología tiene un pequeño detalle y son las actualizaciones, cuando se saca una actualización fuerte, esta tecnología se ve afectada por las versiones anteriores. Eso se debe a que constantemente están cambiando las funciones del Framework. Al momento de desarrollar este trabajo se lanzó la versión 3 de Flutter permitiendo desarrollar aplicaciones en equipos de computo con sistemas operativos Windows y Linux. Se espera que no se abandone en un futuro como sucedió con Windows Phone.

- Al observar en el capítulo 2, sección 2.6 Cada tecnología tienen diferentes requisitos para trabajar en equipos de cómputo. A continuación se detallan los recursos mínimos para que puedan ser realizados:

  - La tecnología de Recat Native está preparada para trabajar en equipos de cómputo que tengan como mínimo Procesador de 2 núcleos a 2GHZ, Memoria RAM 5GB, y 600MB de almacenamiento para la aplicación. El único detalle es que los procesos son más largos en comparación con las otras 2 tecnologías.
  - Android Nativo (Kotlin) Está preparada para ejecutarse en computadoras con las siguientes características mínimas: procesador de 2 núcleos a 2GHz, 6GB de memoria RAM y 20MB de almacenamiento para la Aplicación. Esta tecnología es la que tiene mayor desempeño al momento de realizar procesos, en almacenamiento es la más optimizada ya que las aplicaciones generadas pesan 11MB, en comparación con las otras tecnologías tienen un peso mayor a los 60MB.
  - Flutter. Los requisitos mínimos para trabajar con esta tecnología son: procesador de 2 núcleos a 2GHz, 6 GB de memoria RAM y 120MB de almacenamiento. Las características de esta tecnología son muy similares a la anterior, pero requiere de un almacenamiento mayor.

En cuestiones de hardware no llega a tener grandes diferencias para desarrollar aplicaciones, lo factores que realmente cambian son: el almacenamiento, la memoria RAM y Los tiempos de ejecución de la aplicación(ahí es donde tiene una debilidad React Native).

- En el capitulo 2 en la sección 2.8 se desarrolló una aplicación para desarrollo móvil en las tecnologías: Android Nativo, Flutter y React Native. La aplicación que se eligió fue una calculadora, ya que permite tratar o manipular diferentes tipos de datos como enteros o Strings, requiere de validaciones para un buen funcionamiento y una interfaz donde introduce datos y muestra resultados de información tratada.

  El objetivo de realizar esta aplicación es evaluar que tan complejo es realizarla en cada una de las tecnologías y poder tener una idea de como se realizan las aplicaciones. A continuación se mencionan los resultados obtenidos:

  - La tecnología que resultó ser más cómoda para desarrollar la aplicación calculadora fue React Native. Eso se debe a la fácil manipulación que se tiene con el lenguaje. Si deseas empezar por el desarrollo móvil, únicamente conociendo las base de la programación, React native es la opción correcta para iniciar. Como se menciono anteriormente, debes de tener cuidado si ocupas componentes desarrollados por terceros por que en cualquier actualización puedes tener problemas en tu aplicación y deberás corregirlos lo más pronto posible.
  - La tecnología de Flutter también es amigable para el desarrollo de aplicaciones, el diseño de las aplicaciones es muy personalizado, realmente el limite para crear una aplicación está en los desarrolladores o diseñadores. Requiere de un poco más complejidad que React Native por los manejadores de estado, pero ya cuando se entienden, resulta ser una tecnología para empezar por el desarrollo de aplicaciones móviles. Las actualizaciones no son tanto problema como lo son con React Native, pero existen menos y

tienen fallas al momento de pasarse a otra versión del framework. Flutter es una buena alternativa cuando se quiere empezar a desarrollar una aplicación únicamente con bases de programación en general. Se recomienda más cuando se quiera realizar un proyecto para tecnologías Android iOS o web.

- Por último la tecnología de desarrollo nativo, requiere de un proceso de aprendizaje mayor en comparación con las otras tecnologías. La estructura del lenguaje es fácil de comprender, el entorno de desarrollo Andorid Studio ayuda mucho al momento de crear la interfaz gráfica; las herramientas de diseño son correctas para el desarrollo de la aplicación. Debido a que Android ya tiene una estructura, se tiene uno que adaptar a la manera en que se crean las aplicaciones en esta tecnología. Desde el punto de vista, se considera que las interfaces gráficas tendrían que desarrollarse de la manera que se implementan en las tecnologias de Flutter o React Native. Eso facilita el desarrollo de una aplicación.

De manera general, no existe una tecnología perfecta para desarrollar aplicaciones móviles, existen diferentes tecnologías porque cada una tiene un enfoque diferente. La decisión correcta es saber cuales son tus cualidades o las implementaciones que tiene la aplicación, y en base a eso, se toma la decisión por qué tecnología ocupar.

- Si en una aplicación se busca rendimiento o fluidez, las mejores alternativas son Android nativo o Flutter.

- Si se busca que un código ejecute 2 aplicaciones o más y sea fácil de crear, Flutter y React Native son la opción correcta.

- si se prioriza la optimización y el almacenamiento la mejor tecnología es Android Nativo.

Se considera que la prioridad es aprender desarrollo nativo, al entenderlo, se podrán crear o adaptar las interfaces de cualquier tecnología Nativa a desarrollo Multiplataforma.

Se tien que considerar que las tecnologías están evolucionando frecuentemente, por ejemplo: en la tesis que desarrolló Awel Eshetu Fentaw [17], llega a la conclusión que ambos frameworks son similares, sin embargo, a finales del año 2022 considero que Flutter es superior a React Native, ya que actualmente tiene más componentes nativos, Flutter tiene mayor soporte de comunidad y abarca más plataformas que Android nativo y React native.

Apéndices

Apéndice A

Cuestionario

Apéndice B

Diagramas

Referencias

[1] Center class - widgets library - Dart API.

[2] Column class - widgets library - Dart API.

[3] Container class - widgets library - Dart API.

[4] Diseño lineal — Desarrolladores de Android — Android Developers.

[5] Text class - widgets library - Dart API.

[6] Text · React Native.

[7] TouchableOpacity · React Native.

[8] View · React Native.

[9] Visibility class - widgets library - Dart API.

[10] Android, D. Arquitectura de la plataforma — Desarrolladores de Android — Android Developers.

[11] Android, D. Ciclo de vida de procesos y aplicaciones.

[12] Androidall. La historia de Android.

[13] Codigofacilito. Qué es React Native.

[14] Delía, L. N., Galdámez, N., Thomas, P. J., and Pesado, P. M. Un análisis experimental de tipo de aplicaciones para dispositivos móviles.

[15] Dev, F. Flutter - Build apps for any screen.

[16] Dev, F. Introduction to widgets — Flutter.

[17] Fentaw, A. E. Cross platform mobile application development : a comparison study of React Native Vs Flutter.

[18] Jetbrains. El estado del ecosistema del desarollador 2020, 2020.

[19] Jetbrains. El estado del ecosistema del desarrollador 2021, 2021.

[20] Jetbrains. Herramientas esenciales para desarrolladores de software y equipos, 2022.

[21] Reactjs.org. Virtual DOM and Internals – React.

[22] Sandoval, R. Ciclo de vida de Flutter para desarrolladores Android e iOS — by René Sandoval — Medium.

[23] Singh, A. Lifecycle of React Native Component [2020 Edition] — by Amanpreet Singh — Medium.

[24] Stackoverflow. 2020 Developer survey, 2020.

[25] Stackoverflow. 2021 Developer survey, 2021.