UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Crear y desarrollar una aplicación de alto rendimiento con bajo coste utilizando Flutter y Firebase

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: David Villalón Pardo

**Tutor**: Eliseo Marzal Calatayud

2020-2021

# Dedicatoria

Dedico con todo mi corazón mi trabajo de final de grado a mi madre. Gracias a ella y su persistencia, estos trabajos se han llevado a cabo como una de las mejores formas de honrarla. En cada una de las páginas está tu recuerdo y mi esfuerzo por conseguir ,allá donde estés, sacarte una sonrisa de orgullo. ¡Lo  he conseguido mamá!

# Resumen

El presente trabajo se centra en las grandes diferencias entre el desarrollo de aplicaciones nativas y aplicaciones multiplataforma, con el uso de herramientas como Flutter y Firebase para aplicaciones multiplataforma y el uso de Android Studio o de Xcode para aplicaciones nativas, así viendo las grandes diferencias de estos dos mercados a los que se pueden dirigir los desarrolladores de software.

El camino que toma este trabajo, se basa en el crecimiento de la demanda de aplicaciones multiplataformas a nivel mundial, mediante un análisis previo del uso masivo de dispositivos de diferentes marcas y sistemas, teniendo en cuenta que el desarrollo multiplataforma puede abarcar todos los mercados ya sean móviles o webs y la relación que tiene con la reducción de altos tiempos y costos de desarrollo.

El ambiente de desarrollo multiplataforma es muy amigable y entendible gracias a que las herramientas como Flutter nos da apertura para utilizar el código base de nuestra aplicación para distintas plataformas, como también las diferentes herramientas que nos proporciona Firebase con su autenticación y su almacenamiento de datos por parte de Google Cloud Store.

El uso de aplicaciones multiplataforma y la constancia de la evolución de la tecnología y su ejecución en diferentes entornos da apertura a ambientes cambiantes y adaptables en los cuales estas limitaciones no existan.

**Palabras clave:** Adaptable, Coste, Multiplataforma. Google Cloud Store, Xcode, Android Studio.

# Abstract

This work focuses on the great differences between the development of native applications and cross-platform applications, with the use of tools such as Flutter and Firebase for cross-platform applications and the use of Android Studio or Xcode for native applications, thus seeing the great differences of these two markets that software developers can target.

The path that this work takes is based on the growth in demand for multiplatform applications worldwide, through a prior analysis of the massive use of devices from different brands and systems, taking into account that multi platform development can cover all markets already whether mobile or web and the relationship it has with the reduction of high development times and costs.

The multiplatform development environment is very friendly and understandable thanks to the fact that tools such as Flutter give us openness to use the base code of our application for different platforms, as well as the different tools that Firebase provides us with its authentication and data storage by part of the Google Cloud Store.

The use of multiplatform applications and the constancy of the evolution of technology and its execution in different environments opens up changing and adaptable environments in which these limitations do not exist.

**Keywords :** Adaptable, Cost, Multi Platform, Google Cloud Store, Xcode, Android Studio.

# Resum

El present treball se centra en les grans diferències entre el desenvolupament d'aplicacions natives i aplicacions multiplataforma, amb l'ús d'eines com Flutter i Firebase per a aplicacions multiplataforma i l'ús d'Android Studio o de Xcode per a aplicacions natives, així veient les grans diferències d'aquests dos mercats als quals es poden dirigir els desarrolladors.

El camí que pren aquest treball, es basa en el creixement de la demanda d'aplicacions multiplataformes a nivell mundial, mitjançant una anàlisi prèvia de l'ús massiu de dispositius de diferents marques i sistemes, tenint en compte que el desenvolupament multiplataforma pot abastar tots els mercats ja siguen mòbils o webs i la relació que té amb la reducció d'alts temps i costos de desenvolupament.

L'ambient de desenvolupament multiplataforma és molt amigable i comprensible gràcies a que les eines com Flutter ens dona obertura per a utilitzar el codi base de la nostra aplicació per a diferents plataformes, com també les diferents eines que ens proporciona Firebase amb la seua autenticació i el seu emmagatzematge de dades per part de Google Cloud Store.

L'ús d'aplicacions multiplataforma i la constància de l'evolució de la tecnologia i la seua execució en diferents entorns dona obertura a ambients canviants i adaptables en els quals aquestes limitacions no existisquen.

**Paraules Clau:** Adaptable, Cost, Multiplataforma, Google Cloud Store, Xcode, Android Studio.

# Tabla de contenidos

# Índice de figuras

# Índice de tablas

# Introducción

Los avances tecnológicos en los teléfonos han abierto un extenso mercado al desarrollo de aplicaciones móviles gracias al acceso a internet, estos dispositivos cuentan con una serie de sistemas operativos que facilita el desarrollo de aplicaciones gratuitas que se pueden instalar en cualquier dispositivo que cumpla las características mínimas requeridas sin ningún problema. Al realizar un análisis sobre los medios por los cuales se desarrollan las aplicaciones hoy en día los cuales son:

- JAVAXCODE
- NETBEANS
- ECLIPSE

Estos entornos de desarrollo proporcionan una variedad de herramientas para desarrollar aplicaciones móviles basadas en Java cuales se ejecutan en máquinas virtuales, esto conlleva a que las aplicaciones sean dependientes de esta herramienta ya que no se puede ejecutar su programa sin una máquina virtual. El progreso de las aplicaciones móviles para Android y iOS es totalmente diferente si nos enfocamos en los entornos de desarrollo nativo los cuales serían Java para Android ya que este está basado en una máquina virtual del mismo y así nos facilita la implementación de aplicaciones desarrolladas en este lenguaje y Swift para iOS un lenguaje de programación que nos brinda varias facilidades al momento de crear todo tipo de apps con un rendimiento superior del 2.6 a Objetive-C y un 8.4 a Python.

Ante la situación y al emplearse las grandes ventajas que nos brinda el uso del framework Flutter y la plataforma Firebase de Google que nos brindan una alta compatibilidad en Android y iOS que hace que el desarrollo de aplicaciones al usar estos entornos funcionen de igual manera que al hacerlo en su lenguaje nativo, conociendo que la petición de aplicaciones que funcionen en los dos sistemas operativos principales del mercado y así evitar mayores gastos en las etapas de desarrollo por hacer la misma aplicación para dos sistemas diferentes.

El enfoque de el trabajo se realizada desde un puto de vista de investigación documental acerca de la evolución del desarrollo móvil con el objectivo de tener una compresión crítica y analítica del cambio que se está produciendo en la industria y del impacto que va a tener con el fin de generar nuevos conocimientos y compresiones acerca de esta emergente tecnología. Permite, por lo tanto, adoptar o desarrollar una perspectiva teórica a partir de la revisión, análisis crítico e interpretación de documentos existentes. Para que sea posible, se comienza en el capítulo 1 abordando el problema existente desde todas las perspectivas necesarias. En el capítulo 2 se establecen los objetivos generales y específicos proseguido del capítulo 3 que define las bases teóricas necesarias para poder comprender el trabajo así como el entrelazado, relación y comparación de todos los componentes que lo forman. Finalmente se cierra con un capítulo 4 con las conclusiones y recomendaciones.

# 1. Problema

## 1.1. Antecedente

El desarrollo de aplicaciones móviles se remonta a finales de los 90 con la llegada de los primeros teléfonos móviles en los que ya se podían instalar una aplicación en su sistema operativo. Estas son lo que conocemos como las agendas, los juegos de arcade, los tonos de llamada, etc. Sus elementos y su diseño son fáciles de entender. Las aplicaciones han crecido exponencialmente con el advenimiento de la tecnología estándar de Protocolo de Aplicación Inalámbrica (WAP) utilizada en aplicaciones de comunicación inalámbrica como teléfonos móviles e Internet.

Es aquí cuando surgieron aplicaciones de todo tipo ya sean de uso personal o de uso colectivo  al alcance del ser humano en tan pequeños dispositivos, antes de la llegada de todos estos entornos llegó una de las aplicaciones más populares en 1997 en el teléfono Nokia 6110 denominada “La Serpiente”, antes de la existencia de WAP que permitía a los usuarios acceder a páginas web en versiones adaptadas para móviles, al utilizar el Protocolo de marcado inalámbrico (WMP) en lugar del Protocolo de transferencia de hipertexto (HTTP), muchos desarrolladores se enfrentaron a desafíos importantes para adaptar diferentes tipos de interfaces a pantallas móviles.

El 29 de junio del 2007 fue el lanzamiento del iPhone por Steve Jobs con la llegada de la Apps Store que arrancó con una oferta de 500 aplicaciones, el otro gigante de la tecnología Google entró en pánico y lanzó al mercado Android Marker. Con la llegada de estas dos tiendas virtuales de aplicaciones móviles los programadores tenían que tomar la decisión de desarrollar para Android o para iOS así dividiendo el mercado en dos y obligando a que si una aplicación quería ser lanzada en ambos sistemas debía ser desarrollada en dos entornos totalmente diferentes.

## 1.2. Problema

Con la llegada de dos mercados Android y iOS los grupos de desarrolladores e inversores debían tomar en cuenta los altos costos que conlleva el desarrollo para ambos mercados así monopolizándolo por ver cuál de los dos sistemas lleva las mejores aplicaciones. Cuando hablamos de acaparar el mercado es porque algunas aplicaciones que fueron lanzadas en App Store no vieron la luz en Android por los altos costos de desarrollo o en otro caso los tiempos exuberantes que conlleva el desarrollo a la par de una misma aplicación para dos entornos teniendo en cuenta el soporte y actualizaciones.

## 1.3.Análisis del problema

El desarrollo de aplicaciones móviles que funcionen en ambos entornos no debería ser un problema si lo que se busca es brindar un servicio y generar ganancias del mismo, así escogiendo los mejores medios por los cuales llevarlo a cabo y poder alcanzar a la mayor cantidad de personas posibles.

En años atrás y en algunos lugares del mundo hoy en día el proceso de desarrollo de aplicaciones móviles es relativamente anticuado y es llevado de la misma manera por varios años por temor al cambio, únicamente al desarrollar apps se decide para cual sistema va destinada ya sea Android o iOS con el fin de establecer los tiempos de desarrollo como también los costos, teniendo en cuenta la pérdida de ganancias que conlleva el que la aplicación no llegue al otro sistema que no eligieron ya sea por cantidad de personas o por el no saber programar para ese sistema. Hoy en día todo este proceso no es nada bueno para los desarrolladores e inversores que quieran lanzar una aplicación funcional en ambos sistemas y así abarcar la mayor cantidad de personas y dispositivos para así brindar su servicio o entretenimiento.

## 1.4.Formulación del problema

¿Cómo desarrollar eficientemente aplicaciones que funcionen en ambos sistemas? En la actualidad, siempre ha sido necesario para desarrollar eficientemente aplicaciones para ambos sistemas el desarrollarlas en lenguajes de programación distintos para Android (Java/Kotlin), iOS(Swift y Objective C en el pasado). Sin embargo, esto implica tener que comprender dos tipos de sintaxis distintas, distribución de ficheros, gestión de datos, certificados necesarios para compatibilidad móvil… Y además, obliga a que exista una simetría y coordinación de desarrollo en empresas que, de forma nativa, quieran desarrollar ambas aplicaciones. Si se pretende escalar con un producto digital y una organización, esta duplicidad es un gran problema puesto que produce que sea necesario aumentar todos los recursos por dos (desarrolladores, equipos…) siendo necesario para ello una mayor inversión y coste.

## 1.5.Sistematización del problema

- ¿Qué actividades pueden mejorar con la aplicación de Flutter y Firebase?

Dentro del proceso de desarrollo, de lanzamiento de producto, de resolución de errores… Existen múltiples actividades en las que la aplicación de Flutter, en combinación con Firebase, soluciona problemas como los mencionados en el punto anterior.

- ¿Cuáles son los beneficios que se obtendrán al desarrollar aplicaciones con ambas herramientas?

Mejorar actividades tiene un impacto positivo no solo en los procesos, sino también en todo lo que repercute a costes, contratación y gestión del personal, calidad del producto, agilidad a la hora de adaptarse al mercado y lanzar funcionalidades o solventar bugs...

## 1.6.Resolución del problema

Basado en las cuestiones planteadas y su análisis las aplicaciones móviles que sean compatibles en Android y iOS por medio del framework Flutter y la plataforma Firebase de Google nos facilita el desarrollo de las mismas y también su compatibilidad dentro de ambos sistemas así abarcando un mayor mercado de personas y de dispositivos, evitando así el desarrollo de una sola aplicación para ambos entornos y minimizando los costos de desarrollo como también los tiempos ya sea por mala organización o por falta de conocimiento en los lenguajes aplicados para cada sistema.

Por otro lado, el desarrollo de aplicaciones compatibles con ambos sistemas ha brindado a los usuarios una gama más extensa de aplicaciones que antes sólo funcionaban en un solo sistema y no eran compatibles con otros ya sea porque son nativas del sistema como agenda, teléfono entre otras que solo funcionan en su sistema por temas de mercado y diferenciación de los dispositivos, pero ya no abarcando las demás aplicaciones que llegaron después y que ahora son utilizadas en ambos sistemas ya sea por el desarrollo tradicional y desarrollándose en su lenguaje nativo o aplicando las nuevas tecnologías que se proponen en este contexto.

## 1.7.Beneficiario

La propuesta va dirigida a todos los grupos de desarrolladores que aún programan aplicaciones móviles en su lenguaje nativo teniendo en cuenta los tiempos de producción y los costos que conlleva el hacerlo para ambos sistemas y en el caso de que solo estén enfocados en uno se debe tener en cuenta la pérdida de usuarios y de ganancias al momento de no aplicarlo para ambos sistemas.

Esta propuesta tiene como objetivo no solo beneficiar a los desarrolladores, sino también a los usuarios, ya que utilizan las aplicaciones móviles en un rol más directo y evalúan y observan el desempeño de las aplicaciones.

### 1.8. Alcance del proyecto

Esta propuesta tiene como objetivo mecanizar el desarrollo de aplicaciones móviles del país, proceso que viene siendo de un desarrollo para ambos sistemas de manera separada, por lo cual se utilizará el framework Flutter y la plataforma Firebase de Google para el desarrollo de aplicaciones que sean compatibles para ambos sistemas.

Cabe aclarar que el desarrollo de apps para dispositivos móviles con estas herramientas es accesible para todo tipo de personas que quieran desarrollar aplicaciones y quieran compatibilidad en los dos sistemas.

# 2. Objetivos

## 2.1. Objetivo general

- Elaborar aplicaciones móviles, que cuentan con compatibilidad en Android y iOS para así reducir los tiempos y costos de desarrollo, y así poder tener un mejor alcance en cada dispositivo existente con la implementación de Flutter y Firebase.

## 2.2. Objetivos específicos

- Proporcionar documentación necesaria para comenzar con la utilización de Flutter y Firebase.
- Diseñar una aplicación móvil con Flutter y Firebase y así garantizar la compatibilidad en ambos sistemas.

- Proporcionar estadísticas sobre la reducción de tiempos con el uso de Flutter y Firebase.

## 2.3.Motivación

Desde principios de carrera llevo trabajando y colaborando desde un punto de vista de gestión de producto digital (Product Manager) en el desarrollo de aplicaciones tanto para empresas como instituciones e individuos. Con los años, he ido aprendiendo que para asegurar el crecimiento y escalabilidad de los proyectos, es necesario un concepto clave: la agilidad. Entendiendo como agilidad a la capacidad de iterar rápidamente desarrollos, adaptarlos a nuevas necesidades o realizar tests sobre aprendizajes de los usuarios y los datos. El tener desarrollos nativos, complejos y duplicados de las aplicaciones no solo implica mayor posibilidad de imprevistos si no que multiplica muchísimo el impacto que tiene en la agilidad de los equipos. Es por ello que mi motivación parte de haber estado años explorando soluciones de desarrollo híbridas que parecían estar siempre por detrás del desarrollo nativo en rendimiento y capacidades hasta que llegó Flutter.

# 3. Marco Teórico

## 3.1. Arquitectura cliente-servidor

Esta arquitectura en la cual varios clientes solicitan y reciben información de un servidor centralizado.

Se proporciona una interfaz que permite a los usuarios de cualquier equipo con conexión a la red solicitar información del servidor y ver los resultados devueltos por el mismo. El servidor espera una solicitud del cliente antes de responder.

Idealmente, el servidor proporciona al cliente una interfaz transparente y estandarizada para que el cliente no tenga que conocer los detalles del sistema (es decir, hardware y software). El cliente suele estar ubicado en una estación de trabajo o computadora personal, pero el servidor está ubicado en lugares adecuados ya que su capacidad y tamaño son exuberantes.

Este modelo de procesamiento es particularmente útil cuando el cliente y el servidor tienen diferentes tareas que realizan a diario. Por ejemplo, el enviar un formulario de registro por parte del usuario en el que todas las validaciones son parte del servidor así al usuario solo entregándolo el acceso a distintas actividades como también la información requerida por el mismo. (Redhat, 2020)

El modelo es completamente distinto al modelo antes usado de mainframe, donde un solo mainframe centraliza toda la actividad en el dispositivo, ya que el cliente y el servidor se ven como dispositivos separados. Una terminal "estúpida" relacionada. Simplemente se comunica con la computadora central.

## 3.2. Tecnología móvil

La tecnología móvil es una tecnología que los usuarios utilizan en todas partes ya sea que se conecten a internet o no.

Hoy en día, la tecnología móvil se caracteriza por dispositivos conectados a Internet, como teléfonos inteligentes, tabletas y relojes. Estos son los productos más recientes, como buscapersonas bidireccionales, computadoras portátiles, teléfonos celulares (celulares plegables) y navegadores GPS.

La red de comunicación que conecta estos dispositivos se llama tecnología inalámbrica. Permite que los dispositivos móviles compartan voz, datos y aplicaciones (aplicaciones móviles).

## 3.3. Aplicación móvil

Una aplicación móvil, comúnmente llamada app, es un tipo de software diseñado para ejecutarse en teléfonos inteligentes, tabletas, etc. Las aplicaciones móviles se utilizan a menudo para proporcionar a los usuarios servicios similares a los disponibles en las computadoras. Una aplicación es generalmente un software pequeño con funcionalidad limitada. Este software de aplicación fue utilizado originalmente por Apple Inc.

Las aplicaciones móviles se están alejando de los sistemas de software integrados que se encuentran comúnmente en las PC. En cambio, cada aplicación ofrece funciones limitadas y distintas, como juegos, calculadoras y navegación móvil. Las aplicaciones pueden evitar la multitarea debido a los recursos de hardware limitados de los primeros dispositivos móviles, pero la especificidad de la aplicación ayuda porque los consumidores pueden seleccionar manualmente las funciones del dispositivo.

La aplicación móvil más simple es obtener una aplicación para PC y transferirla a su dispositivo móvil. A medida que las aplicaciones móviles se vuelven más poderosas, esta técnica suele fallar. Los enfoques más sofisticados incluyen desarrollos específicos para entornos móviles que aprovechan tanto las limitaciones como los beneficios de los entornos móviles.

## 3.4.Sistemas operativos móviles

El sistema operativo es la interfaz de software responsable de administrar y manipular las unidades de hardware y ponerlos a disposición de los usuarios. Para los teléfonos móviles, el sistema operativo se desarrolló para permitir a los usuarios usar el teléfono de la misma manera que usaban una computadora personal hace 10 o 20 años.

Los sistemas móviles más usados a nivel mundial son Android, iOS y Windows Phone OS.

## 3.5.Comparativa sistemas operativos

La comparativa se realizará entre los tres sistemas operativos más utilizados.

### Android

Android es un sistema operativo basado en Linux que usa Linux 2.6 para proporcionar servicios básicos como seguridad, administración de memoria, administración de procesos, etc. Proporciona una amplia gama de bibliotecas que permiten a los desarrolladores de aplicaciones crear muchos tipos de acuerdo a las necesidades del usuario. Las aplicaciones de Android están escritas en el lenguaje de programación Java.

### iOS

Apple iOS es un sistema operativo de teléfono móvil de código cerrado desarrollado por Apple en 2007. Solo se utiliza con productos Apple (iPhone, iPod, iPad). La arquitectura de iOS se basa en tres capas integradas. Cocoa touch es la capa que proporciona la infraestructura subyacente utilizada por su aplicación. La segunda capa es la capa multimedia, que proporciona dibujos en 2D (2D) y 3D, así como servicios de audio, videos animados, formatos de imágenes y documentos, y soporte de audio y video. El tercer nivel es el sistema operativo central, que proporciona servicios básicos como tipos de datos de bajo nivel, servicios de arranque, redes y acceso

### Windows Phone

Este sistema operativo móvil de código cerrado desarrollado por Microsoft Corporation y utilizado por muchos dispositivos inteligentes (PDA, teléfonos

inteligentes y dispositivos táctiles). El sistema operativo Windows Phone se basa en una versión compacta de .Net Framework, por lo que puede comenzar de inmediato. Aplicaciones móviles orientadas a la web.

*Tabla 1: Sistemas Móviles*

| Android | iOS | Windows 10 Mobile |
|---|---|---|
| Compañía | | |
| Open Handset Alliance | Apple Inc. | Microsoft |
| Licencia | | |
| Libre y de código abierto, pero por lo general se incluye con aplicaciones y drivers propietarios. | Propietaria excepto para componentes de código abierto. | Propietaria |
| Programado en | | |
| C, C ++, Java,Kotlin. | C, C ++ , Objective-C, Swift. | .NET C#, VB.NET, Silverlight, native C/C++, WinRTP (XMLA), DirectX |

| Tienda oficial de aplicaciones | | |
|---|---|---|
| Google Play | App Store | Windows Store |
| Coste de desarrollo para el OS móvil | | |
| Gratis | Gratis con Xcode 7 | Gratis |
| Soporte de impresoras | | |
| 4.4+ usando Google Cloud Print, pero no a través de USB | Sí (AirPrint) | 10+ |
| Motor de navegador web por defecto | | |
| Blink | WebKit | Trident (EdgeHTML después de la versión 10) |
| Navegadores webs disponibles | | |
| Chrome para Android, Opera, Firefox | Safari, Chrome para iOS, Opera Mini, Firefox | Internet Explorer, Opera Mini, UC Browser, Microsoft Edge |

| | | |
|---|---|---|
| Motor de búsqueda de los navegadores | | |
| Muchos (entre ellos Google) | Bing, Google, Yahoo! Search, DuckDuckGo | Muchos (entre ellos Bing) |
| Voz sobre IP | | |
| Protocolo SIP o software de terceros | FaceTime y software de terceros | 8+ Skype |
| Software de pago con tecnología NFC | | |
| Disponible en cualquier dispositivo compatible con el hardware. Android Pay para pagos NFC disponible en Play Store. | 8+: iPhone 6/6 Plus vía Apple Pay | 8+ |
| USB On-The-Go | | |
| 3.1 + | No | 10+ |
| Reproducción de audio | | |
| AAC LC/LTP 3GPP, HE-AACv1 (AAC+), HE-AACv2 (AAC+ mejorado) AMR-NB, AMR-WB, MP3, MIDI. | AAC, AAC protegido (del iTunes Store), HE-AAC, MP3, MP3 VBR, Audible (formatos 2, 3, 4, Audible. | MP3, WMA Std 9.2, WMA Pro, FLAC, AMR-NB, AAC-LC, AAC+, eAAC+ |

| Reproducción de vídeo | | |
|---|---|---|
| H.263, H.264 (hasta Baseline Profile), H.265 HEVC, MPEG-4 SP, DivX, XviD, VP8, VP9 | H.264 (hasta High Profile), MPEG-4, M-JPEG | H.263, H.264, WMV, MPEG4, MPEG4 @ HD 720p 30fps, DivX, XviD |
| Teclado Bluetooth | | |
| 2.3+, en versiones anteriores a través de software de terceros | Sí | 8.1u2+ |
| Cliente SSH | | |
| Sí | Sí | Sí |
| VPN | | |
| Sí | Sí | 8.1+ |
| Reconocimiento de voz | | |
| Sí | 5+ (Siri) | 8.1+ Microsoft Cortana |

## 3.6.Lenguaje de modelado unificado (UML)

UML es una serie de diagramas integrados que permiten a los desarrolladores definir, visualizar, crear y documentar cualquier sistema, cabe aclarar que no solo es útil para modelos de negocios y sistemas. También UML representa un conjunto de excelentes técnicas de ingeniería que han demostrado ser efectivas para modelar sistemas grandes y complejos.

## 3.7.Servidor web

**¿Qué es un servidor web?**

Un servidor web es un sistema con gran capacidad que brinda a sitios web la facilidad de manipular la información de cada usuario a través de los métodos HTTP.

**Manejo de información**

Cuando hace clic en un enlace en una página web, envía un formulario o realiza una búsqueda, el navegador envía una solicitud HTTP al servidor.

Esta solicitud incluye:

- La URL identifica el servidor y el recurso de destino (por ejemplo, un archivo HTML, un punto de datos específico en el servidor o una herramienta para ejecutar).
- Un método que define una acción requerida (por ejemplo, para obtener un archivo o para guardar o actualizar algunos datos). Los diferentes métodos / verbos y acciones asociadas se enumeran a continuación.

  - GET: Obtener un recurso específico
  - POST: Crear un nuevo recurso

- HEAD: obtenga información de metadatos sobre un recurso específico sin obtener un cuerpo como GET. Por ejemplo, puede usar una solicitud HEAD para ver cuándo se actualizó por última vez un recurso antes de usar una solicitud GET ("máscara"). Si el recurso ha cambiado, descargue el recurso.
- PUT: Actualizar un recurso existente
- DELETE: Elimina el recurso especificado.
- TRACE, OPTIONS, CONNECT, PATCH: Estos verbos son para tareas menos comunes / avanzadas, por lo que no los cubriremos aquí.

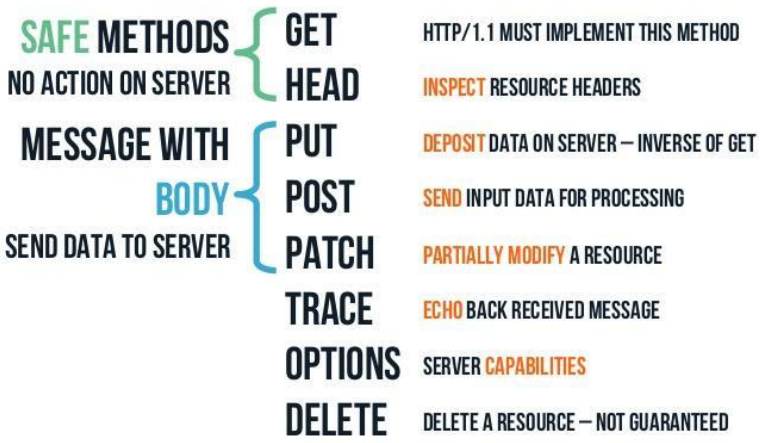



*Fig. 1: Métodos HTTP*

**¿Qué es un servicio web?**

El servicio web es un término general para las funciones de software interoperables entre máquinas alojadas en ubicaciones donde las direcciones de red están disponibles.

## 3.8.Base de datos

Base de datos es una estructura organizacional para cualquier entorno dirigida al manejo de información de manera en la cual los datos se organizan y modelan normalmente en filas y columnas en un conjunto de tablas. Ofreciendo diferentes métodos por los cuales el administrador puede actualizar, eliminar , ver e insertar datos a través de los métodos SQL.PSQL, etc.

## 3.9.Modelo vista controlador (MVC)

MVC significa lo siguiente:

- Modelo: El backend que contiene toda la lógica de datos
- Ver: El frontend o la interfaz gráfica de usuario (GUI)
- Controlador: Los cerebros de la aplicación que controla cómo se muestran los datos

El concepto de MVC fue introducido por primera vez por Trygve Reenskaug, quien propuso MVC como una forma de desarrollar interfaces gráficas para aplicaciones de escritorio.

Actualmente, el patrón MVC se utiliza en aplicaciones web modernas para hacer que las aplicaciones sean escalables y fáciles de mantener y desarrollar.

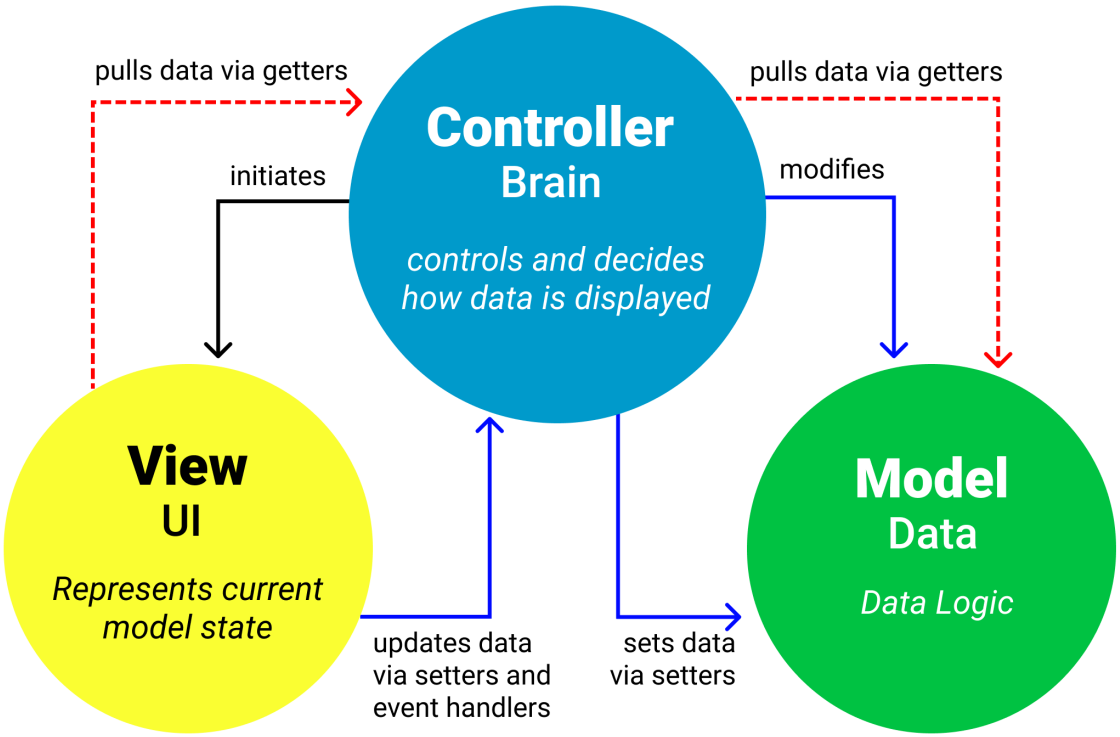



*Fig. 2: Modelo Vista Controlador*

El patrón MVC le ayuda a dividir el código de front-end y back-end en componentes separados. Esto hace que la gestión y modificación por ambas partes sea mucho más fácil sin interferir entre sí. Pero eso es fácil de decir, especialmente si varios desarrolladores necesitan actualizar, modificar o depurar una aplicación completa a la vez.

El modelo MVC administra los datos de entrada ya sea una base de datos, un API e incluso un objeto JSON.

## 3.10. Variables

### ¿Qué es una variable?

Las variables se utilizan para almacenar información que es referenciada y manipulada por programas de computadora. También proporciona una forma de

etiquetar sus datos con nombres descriptivos para que usted y nosotros podamos comprender mejor nuestro programa. Es conveniente pensar en las variables como contenedores de información. Su único propósito es etiquetar los datos y almacenarlos en la memoria. Estos datos están disponibles en todo el programa.

**Asignar valor a una variable**

Se sabe que la nomenclatura de una variable en la programación de computadoras. Al nombrar variables, piénselo dos veces. Haga todo lo posible para que los nombres que le da a sus variables sean fáciles de entender para otros lectores. Cuando vuelves a visitar un programa que escribiste hace meses o años, a veces los otros lectores son tú mismo.

Cuando se asigna una variable, se utiliza = símbolo =. El nombre de la variable va a la izquierda y el valor que desea almacenar en la variable va a la derecha.

```
1 | irb :001 > first_name = 'Joe'
2 | => "Joe"
```

*Fig. 3: Asignación de valor a una variable*

Las variables se dividen a su vez en dos tipos:

- Variables independientes
- Variables dependientes

## 3.11. Independientes

Una variable independiente significa que se detectará con una variable que no se puede cambiar a otra. Las variables independientes también se denominan predictores o factores.

### 3.11.1. Dependientes

Las variables dependientes son aquellas que se miden o prueban en un experimento. Este es el resultado de las acciones del participante y se puede modificar en función del resultado de la acción realizada por el participante.

### 3.11.2. Diferencias entre variable dependiente y variable independiente

*Tabla 2: Variables dependientes e independientes*

| Variable dependiente | Variable independiente |
|---|---|
| Su valor depende de otra variable | Su valor nunca depende de otra variable |
| Su cambio no afecta a la variable independiente | Su cambio afecta a la variable dependiente |
| Las variables independientes ahora son más fáciles de derivar y no requieren pasos complicados u observaciones matemáticas. | Las variables dependientes se obtienen a partir de estudios longitudinales o mediante la resolución de fórmulas matemáticas complejas. |

## 3.12. Plataformas para desarrollo móvil

Si se desea crear una aplicación que atraiga a una gran cantidad de usuarios, se debe comprender el tipo de sistema operativo que utiliza el público objetivo. Según Statccounter, un proveedor de servicios de análisis web en tiempo real, iOS y Android para dispositivos móviles son en realidad los sistemas operativos de teléfonos inteligentes y tabletas más populares del mundo. En 2019, Android representará el 75%de la cuota de mercado global y iOS el 22,52%.

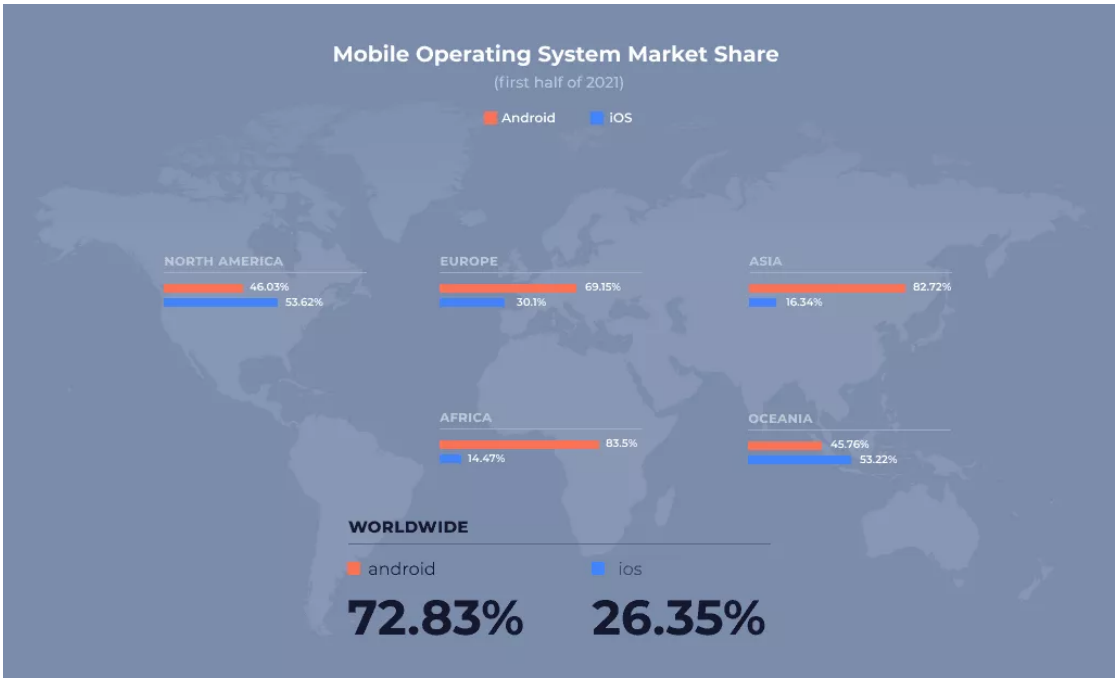



*Fig. 4: Sistemas operativos más populares en el 2021*

### 3.12.1. Android

Considerado un sistema operativo móvil que no se limita solo a ese mercado ya que lo podemos encontrar en diferentes dispositivos ya sean televisores, tablets entre otros, gracias a que está basado en un kernel de Linux le ha ayudado a expandirse a todos los ambientes tecnológicos. Proporcionándoles una amplia gama de herramientas que nos permite crear aplicaciones y juegos innovadores en su entorno de lenguaje Java.


*Fig. 5:Android*

*Tabla 3: Características de Android*

| **Características** | **Descripción** |
|---|---|
| Conectividad | Compatibilidad con tecnologías GSM, IDEN, UMTS, Bluetooth, Wifi, LTE, etc. |
| Almacenamiento | SQLite para fines de almacenamiento de datos |
| Soporte de medios | Varios formatos de audio, video, imagen tales como: MPEG-4 SP, AMR, AAC, MP3, WAV, JPEG, PNG, etc. |
| Navegador | Basado en blink con un motor JavaScript V8 que admite HTML5 y CSS3 |
| Mensajería | Formatos compatibles como SMS, MMS y CD2M de Android Google Cloud. |
| Multitarea | Asignación de memoria en segundo plano para tener varias aplicaciones en ejecución. |
| Wifi | Tecnología que permite la conexión a redes inalámbricas para la búsqueda y para compartir información con otros usuarios. |
| Captura de pantalla | Combinación de teclas del botón de encendido con los de audio para tomar una fotografía de lo que tengamos en pantalla. |
| Multi-idioma | Compatibilidad con varios idiomas a nivel mundial para comodidad del usuario. |

*Tabla 4: Versiones de Android*

| 1.6 Donut | |
|---|---|
| Año de publicación | 15/09/2009 |
| Características | • GPS<br>• Grabar y reproducir videos<br>• Control de batería<br>• Conexión VPN |
| 2.3 Gingerbread | |
| Año de publicación | 06/12/2010 |
| Características | • Soporte para HTML5<br>• Bluetooth 2.1<br>• Soporte para-Facebook<br>• Mejor rendimiento y memoria<br>• Soporte para–Adobe Flash<br>• Soporte NFC<br>• Mejora en audio y gráficos para juegos |

| 3.2 Honeycomb | |
|---|---|
| Año de publicación | 22/02/2010 |
| Características | • Soporte para tablets<br>• Soporte para escritorio 3D<br>• Widgets rediseñados<br>• Google Talk |
| 4 Jelly Bean mrl | |
| Año de publicación | 19/10/2012 |
| Características | • Multiplataforma<br>• Fin de soporte para Flash<br>• Soporte para rotación de pantalla<br>• Mejoras en rendimiento<br>• Mejora en funcionalidades |
| 5 Lollipop | |
| Año de publicación | 12/09/2014 |
| Características | • Nuevo esquema de diseño Material Design<br>• Mejoras en las transiciones<br>• Mejoras en la interfaz |

| 7 Nougat | |
|---|---|
| Año de publicación | 17/06/2016 |
| Características | • Modo Multitarea<br>• Agrupación de aplicaciones<br>• Agrupación de notificaciones |
| 8 Oreo | |
| Año de publicación | 21/08/2017 |
| Características | • Cambio de arquitectura<br>• División en capas (Sistema, Drivers)<br>• Freno a la Fragmentación<br>• Mejora en las actualizaciones<br>• Mejora en la compatibilidad del sistema |
| 9 Pie | |
| Año de publicación | 06/08/2018 |
| Características | • Sustitución de botón home por una barra de gestos.<br>• Límite de usos de aplicaciones<br>• Mejora al modo silenciar<br>• Mejora al rendimiento de batería |

| Android 10 | |
|---|---|
| Año de publicación | 03/10/2019 |
| Características | • Tema oscuro<br>• Bloqueo de operadoras<br>• Soporte nativo para el reconocimiento facial<br>• Modo escritorio<br>• Soporte 5G |
| Android 11 | |
| Año de publicación | 08/09/2020 |
| Características | • Burbujas flotantes<br>• Función Nearby Sharing<br>• Envío de información de cualquier tipo sin conexión a internet y con AirDrop para iOS. |

Estas son las distintas versiones de Android de acuerdo a Miguel,M (2021)

### 3.12.2. Arquitectura de Android

La arquitectura de Android contiene un número diferente de componentes para admitir las necesidades de cualquier dispositivo Android. El software Android contiene un kernel linux de código abierto que tiene una colección de varias bibliotecas de C / C ++ que se exponen a través de un marco de servicios de aplicación.

De todos los componentes, el kernel de Linux proporciona la funcionalidad principal del sistema operativo del teléfono inteligente, y la máquina virtual Dalvik (DVM) proporciona la plataforma para ejecutar aplicaciones de Android.

Los principales componentes de la arquitectura Android son los siguientes: -

- Aplicaciones
- Marco de aplicación
- Tiempo de ejecución de Android
- Bibliotecas de plataforma
- Linux Kernel

La estructura de los componentes pueden verse en la figura 6.

**Aplicaciones. -**

Las aplicaciones son el siguiente nivel de la arquitectura de Android. Aplicaciones preinstaladas para casa, contactos, cámara, galería y más. Aplicaciones de terceros descargadas de Play Store, como aplicaciones de chat y juegos. Solo se instalan en este nivel. Funciona en el entorno de ejecución de Android utilizando clases y servicios proporcionados por el marco de aplicación.

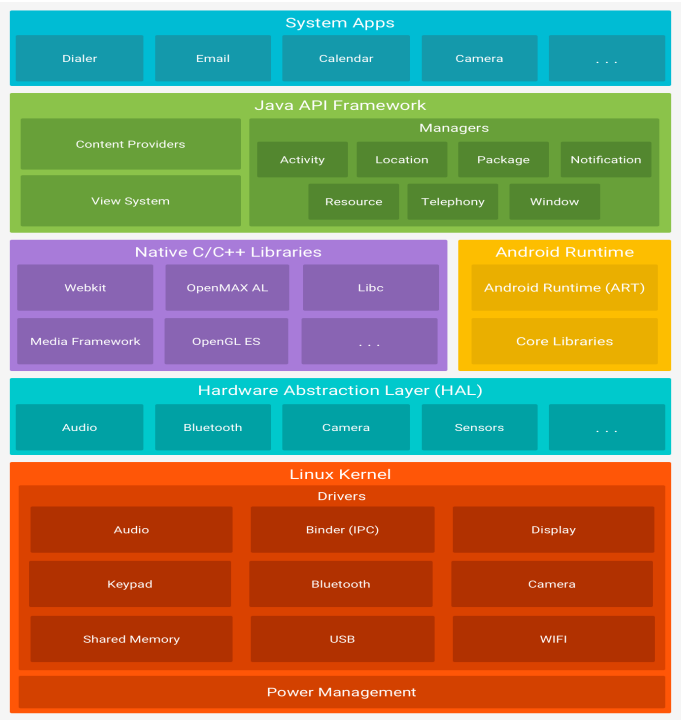



*Fig. 6: Pila de software de Android*

**Marco de aplicación. -**

El marco de la aplicación proporciona algunas clases importantes que se utilizan para crear aplicaciones de Android. Proporciona una abstracción general del acceso al hardware y también ayuda a administrar la interfaz de usuario mediante los recursos de la aplicación. Por lo general, crea una clase concreta para proporcionar un servicio que puede ayudarlo a construir su aplicación. (Arquitectura de la plataforma Android, 2020)

**Tiempo de la ejecución de la aplicación. –**

Una de las partes más importantes de Android es el entorno de ejecución de Android. Contiene componentes como la biblioteca principal y la máquina virtual Dalvik (DVM). Básicamente, proporciona la base para el marco de la aplicación y mejora la aplicación utilizando las bibliotecas principales. (Arquitectura de la plataforma Android, 2020)

**Bibliotecas de la plataforma. –**

La biblioteca de plataforma incluye varias bibliotecas C / C importantes y bibliotecas basadas en Java, como Media, Graphics, Surface Manager y OpenGL. Proporciona soporte para el desarrollo de Android.

**Kernel de Linux. –**

El kernel de Linux está en el corazón de la arquitectura de Android. Administre todos los controladores disponibles, incluidos controladores de pantalla, controladores de cámara, controladores de bluetooth, controladores de audio, controladores de memoria y más. Obligatorio en el momento de la ejecución. (Arquitectura de la plataforma Android, 2020)

### 3.12.3. ¿Por qué Android?

Existe un sin número de razones por el cuál elegir Android cuales son:

1. **Entornos de desarrollo**

   Entornos como Android SDK y Eclipse IDE, etc. Se pueden adquirir de manera gratuita.

2. **Código abierto**

   Android tiene una gran variedad de bibliotecas de código abierto. De esta manera, los desarrolladores tienen la libertad de proporcionar o ampliar la plataforma según sea necesario para crear aplicaciones móviles que se ejecuten en dispositivos Android.

3. **Soporte multiplataforma**

   El sistema operativo Android se puede ejecutar en muchos teléfonos, tabletas, televisores, etc. Incluso el desarrollo de apps móviles de Android puede ocurrir en Windows, Mac OS o Linux.

4. **Soporte multioperadora**

   En todo el mundo, un gran número de operadores de telecomunicaciones como Airtel, Vodafone, Idea Cellular, AT & T Mobility, BSNL, Movistar, Claro, etc. Son compatibles con teléfonos con Android.

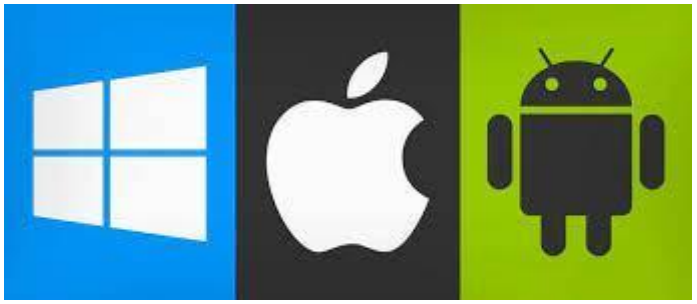


*Fig. 7:Windows Phone, Mac vs Android*

### 3.12.4. Ventajas y desventajas de Android

**Ventaja**

- Personalizable
- Rentable
- Soporte para guardar imágenes RAW
- Almacenamiento en la nube de Google

**Desventaja**

- Agotamiento de la batería por acumulación de procesos
- Fácil instalación de aplicaciones de terceros que pueden vulnerar su seguridad y sus datos
- Mayor demanda de código en Java que Objective-C
- Anuncios en sus aplicaciones

Existe otra condición que tiene sus ventajas y desventajas dentro de Android que es la posibilidad de publicar una app sin un exhaustivo proceso de evaluación como el que tiene iOS. Se considera ventaja por un lado porque agiliza los procesos de lanzamiento y customización de las aplicaciones, sin embargo, se considera desventaja

porque abre la puerta, pese a que existe una mínima revisión, a aplicaciones con intencionalidades maliciosas.

### 3.12.5. Framework Flutter

Es un marco para el desarrollo de aplicaciones de Android e iOS, desarrollado por Google y lanzado como un proyecto en el 2018, con algunos elementos estándar para la interfaz de usuario de Google, proporcionando una gran variedad de bibliotecas. También es útil para desarrollar aplicaciones de escritorio. En general, las aplicaciones desarrolladas en Flutter tienen la apariencia normal de una aplicación por sistema y funcionan como se espera sin que el programador tenga que considerar las características específicas de cada sistema. (Ionos, 2020)

**¿Cómo funciona Flutter?**

Flutter utiliza un motor gráfico, llamado Skia usado para descartar los componentes nativos y así replicándolos desde cero mediante el uso de librerías que facilitan la creación de los nuevos componentes.

El replicar y mantener actualizados todos los elementos gráficos es una tarea realmente grande, que solamente una empresa con los recursos necesarios como Google podría abarcar. Todo lo que se encuentra dentro de una aplicación de Flutter es un widget, desde un botón, menú, textos o cualquier componente que conforme el diseño de pantalla. Estos constan con una organización jerárquica para mostrarse en pantalla. Un widget puede contener a otro llamado así mismo container Widgets. Existen dos tipos de widgets los cuales son:

**Stateless Widget:**

Los widgets sin estado son útiles cuando la parte de la interfaz de usuario que está describiendo no depende de nada más que la información de configuración en el objeto mismo, pueden ser botones o cajas de texto, se considera usar Stateful Widget.

**Consideraciones de rendimiento:**

El método por el cual se compila un widget sin estado se lo realiza en tres situaciones: la primera vez que el widget se inserta en el árbol, la segunda vez cuando el padre del widget cambia su configuración y la tercera cuando un InheritedWidget depende de los cambios.

Si el widget padre cambia regularmente de configuración es importante optimizar el rendimiento del método de compilación para mantener el rendimiento de renderizado fluido.

Hay varias técnicas que uno puede utilizar para minimizar el impacto de la reconstrucción de un widget sin estado:

- Utilice widgets const siempre que sea posible y proporcione un constructor const para el widget para que los usuarios del widget también puedan hacerlo.
- Considere la posibilidad de refactorizar el widget sin estado en un widget con estado para que pueda utilizar algunas de las técnicas descritas en

StatefulWidget, como el almacenamiento en caché de partes comunes de subárboles y el uso de GlobalKeys al cambiar la estructura de árbol.

```
class GreenFrog extends StatelessWidget {
  const GreenFrog({ Key? key }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(color: const Color(0xFF2DBD3A));
  }
}
```

*Fig. 8: Esqueleto de una subclase de widget sin estado de llamada GreenFrog*

**StatefulWidget:**

Son widgets con estado que cambia de apariencia por ejemplo un checkbox, slider, form entre otros en respuesta a una acción por parte del usuario. El proceso de creación continúa de forma recursiva hasta que la descripción de la interfaz de usuario es totalmente concreta (por ejemplo, consta completamente de RenderObjectWidgets, que describen renderObjectconcreto s).

Consideraciones de rendimiento:

Hay dos tipos de categorías de StatefulWidgets.

- El primero es uno que asigna recursos en State.initState y los elimina en State.dispose, pero que no depende de InheritedWidgets o llamar a State.setState. Estos widgets se utilizan comúnmente en la raíz de una aplicación o página, y se comunican con subwidgets a través de ChangeNotifiers, Streams u otros objetos de este tipo.

- La segunda categoría son los widgets que utilizan State.setState o dependen de InheritedWidgets. Normalmente, estos se volverán a generar muchas veces durante la duración de la aplicación y, por lo tanto, es importante minimizar el impacto de volver a generar un widget de este tipo. D

Hay varias técnicas que uno puede utilizar para minimizar el impacto de la reconstrucción de un widget con estado:

- Utilice widgets constantemente siempre que sea posible. (Esto equivale a almacenar en caché un widget y volver a usarlo).
- Empuje el estado a las hojas. Por ejemplo, si su página tiene un reloj en marcha, en lugar de poner el estado en la parte superior de la página y reconstruir toda la página cada vez que el reloj marca, cree un widget de reloj dedicado que solo se actualice a sí mismo.

```
class YellowBird extends StatefulWidget {
  const YellowBird({ Key? key }) : super(key: key);

  @override
  _YellowBirdState createState() => _YellowBirdState();
}

class _YellowBirdState extends State<YellowBird> {
  @override
  Widget build(BuildContext context) {
    return Container(color: const Color(0xFFFFE306));
  }
}
```

*Fig. 9: Esqueleto de una subclase de widget con estado llamada YellowBird*

En ambos widgets utilizamos los siguientes términos:

- **StatefulWidget y State,** para widgets que se pueden crear de forma diferente varias veces a lo largo de su vida útil.
- **StatelessWidget,** para widgets que siempre compilan de la misma manera dada una configuración y un estado ambiente determinados.
- **InheritedWidget,** para widgets que introducen el estado ambiente que pueden leer los widgets descendientes.

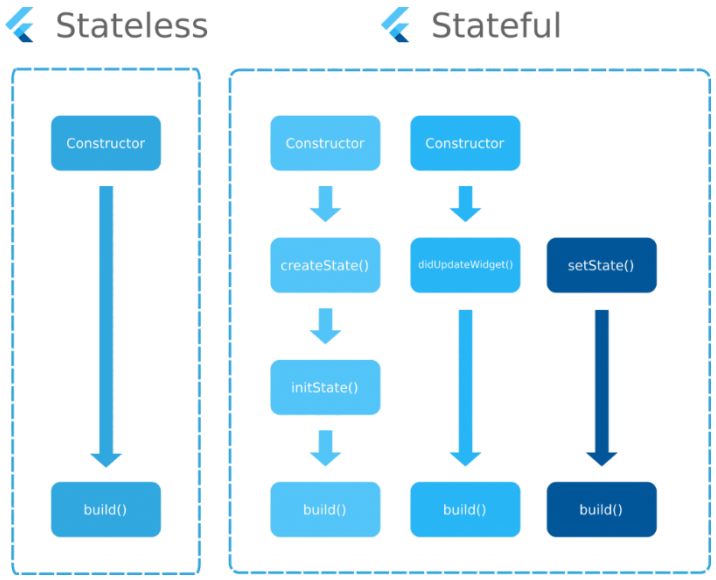



*Fig. 10:Stateless y Stateful*

**Estructura de archivos**

Es considerable que a medida que crece un proyecto, se vuelve cada vez importante el tener una estructura y un formato adecuado para el código, de lo contrario se pueden generar varias complicaciones las cuales son:

- No se encuentran las rutas de los archivos
- Difícil de entender
- En el peor de los casos una aplicación de bajo rendimiento

Normalmente la estructura de un nuevo proyecto es la siguiente:

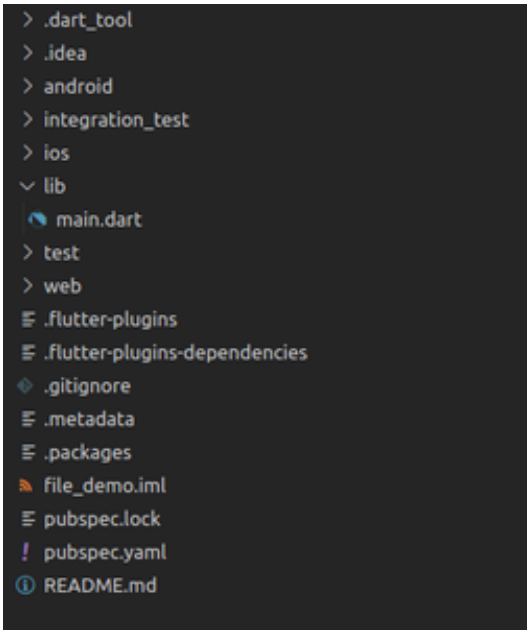



*Fig. 11: Estructura de un proyecto en Flutter*

1. **Archivos estáticos de la aplicación.**

Estos son los archivos estáticos que se usarán en la aplicación como, por ejemplo: iconos, fuentes, imágenes de fondo, videos, etc. Lo recomendable es tener carpetas individuales para cada uno de los tipos de datos.

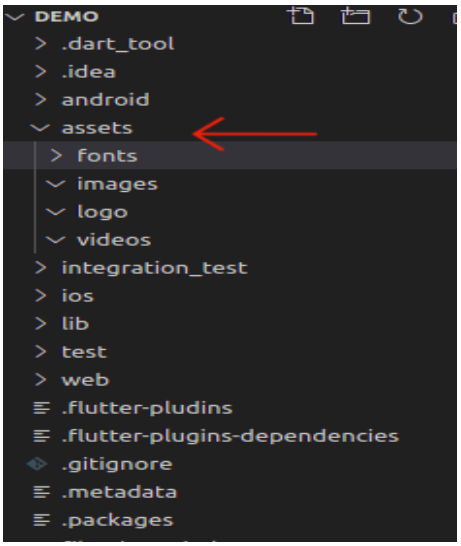



*Fig. 12: Archivos estáticos*

2. **Funciones en la nube**

Las funciones en la nube son el código back-end que se almacena en los servidores de Google Cloud, las funciones se ejecutarán al ocurrir un evento en específico. Uno de los ejemplos más prácticos es en la llamada de archivos de audio, video o texto y los almacena en el servidor para uso futuro.

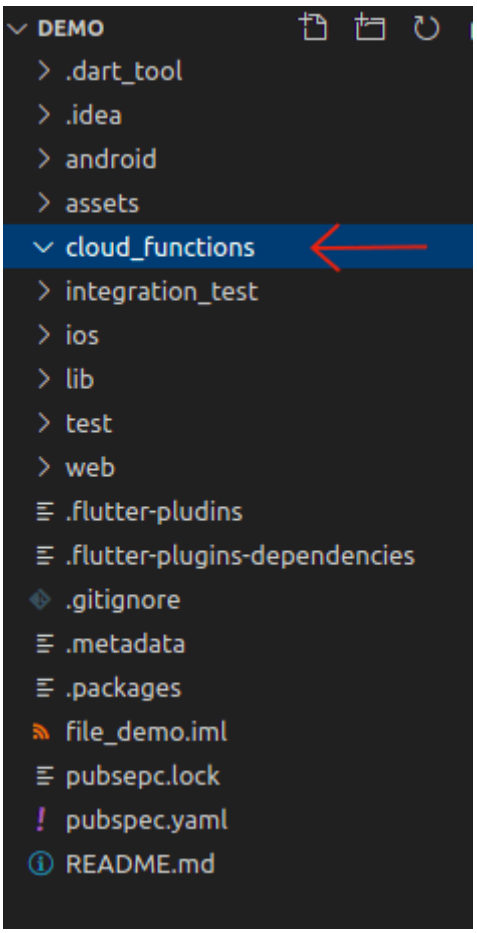



*Fig. 13: Funciones de la nube*

3. **Pantallas**

Este directorio contiene el boceto real de la interfaz de la aplicación, estas se pueden dividir en más carpetas, unas para las pantallas principales y otras para las pantallas flash.

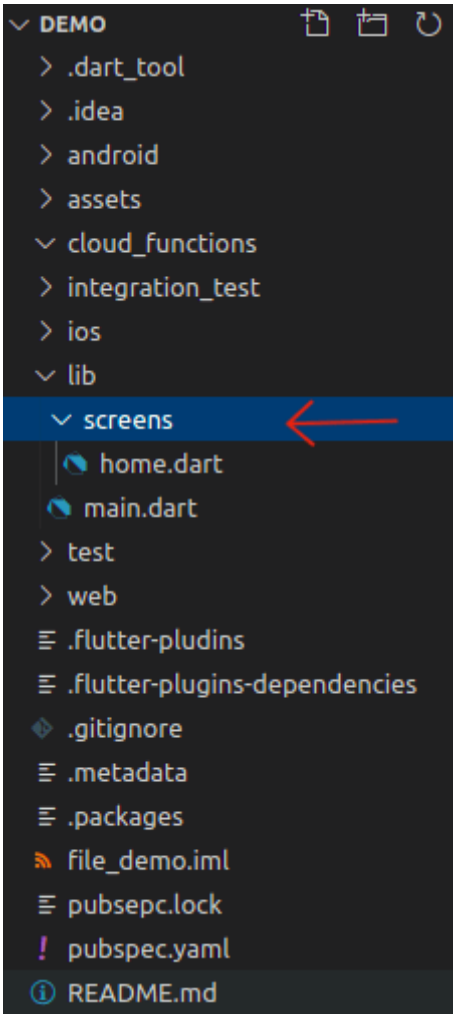



*Fig. 14: Pantallas*

4. **Proveedores**

   Este directorio contiene todas las interacciones que se realizarán fuera de la aplicación, ningún código dentro de este directorio interactúa con la nube o el servidor.

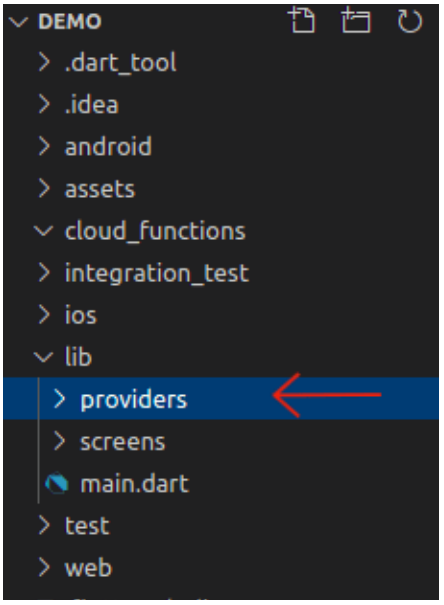



*Fig. 15: Proveedores*

### 5. Utilidades

Aquí encontraremos toda la lógica de la aplicación o la lógica empresarial de toda nuestra aplicación.

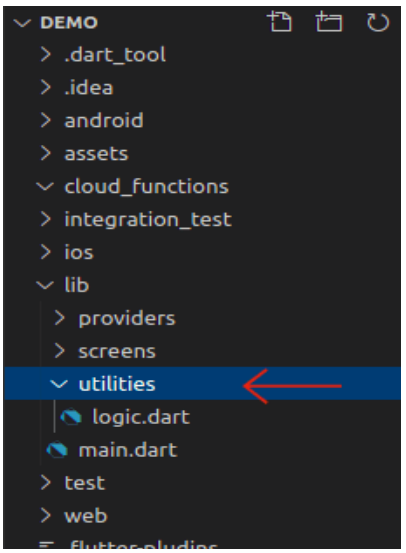



*Fig. 16: Utilidades*

6. **Widgets**

Por el nombre mismo vemos que esta carpeta contendrá los widgets estáticos o los widgets que se utilizan en la aplicación. Por ejemplo, los mosaicos de una red social, las listas de amigos entre otros.

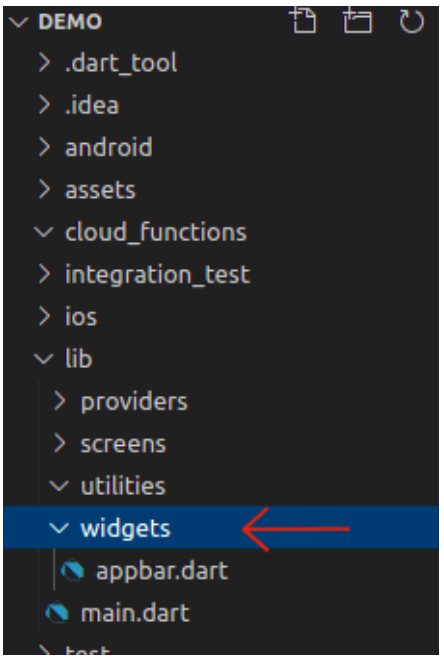



*Fig. 17:Widgets*

7. **Modelos**

Un modelo suele ser una colección de datos recuperados de un servidor, usuario o API externos. Se utilizan junto con los widgets para complementar la interfaz de una aplicación. Nuevamente, tomando el ejemplo de la aplicación meteorológica, el modelo o conjunto de datos podría ser el nombre de la ubicación, la temperatura en grados Celsius o la temperatura en Fahrenheit.

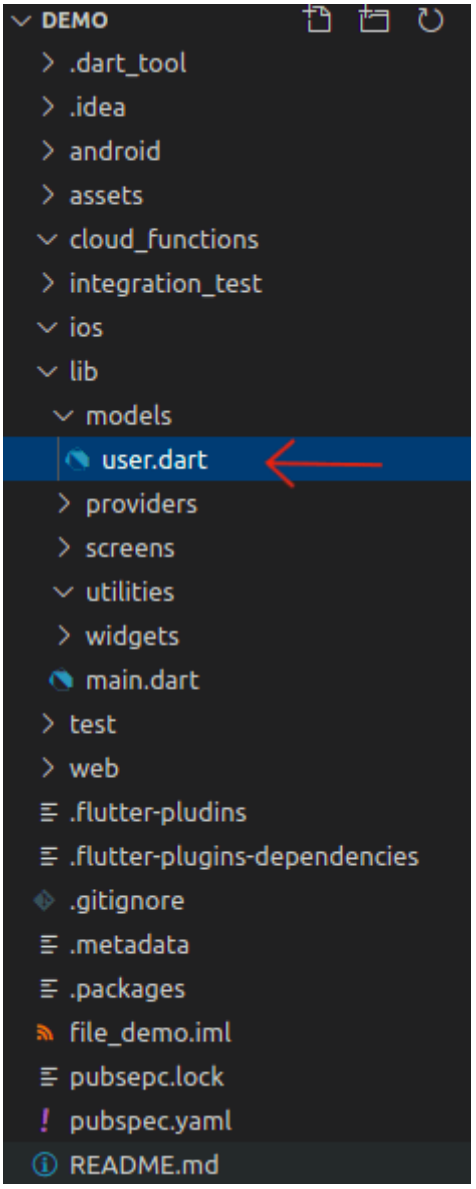



*Fig. 18: Modelos*

**Flutter vs Android (Java-Kotlin)**

Flutter el último SDK de desarrollo de aplicaciones multiplataforma, es un conjunto complemento de herramientas y soluciones para desarrollo móvil. La idea es usar el mismo código base para diferentes plataformas (Android y iOS). Con el desarrollo de aplicaciones en su lenguaje nativo cada vez es más caro el desarrollo.

Elegir entre Flutter y Android no es sencillo, ya que son varios factores los que influyen en esta decisión y una de las más claras es al realizar el análisis del proyecto y encontramos definido el alcance del proyecto. Android tiene una ventaja de antigüedad y con la llegada de su nuevo sucesor Kotlin ofreciendo una gama casi inagotable de soluciones incluso para problemas más complejos. El escribir una aplicación en Kotlin nos da la ventaja de tener acceso a las últimas mejoras del hardware de Android, su compatibilidad con varios dispositivos basados en Android.

Flutter tienta a los desarrolladores con la promesa de obtener dos aplicaciones nativas para Android y iOS, escribiendo la base del código en Dart para ambas interfaces de usuario para proyectos no tan complejos, especializado para desarrollar aplicaciones de tipo fetch-and-display.

Las aplicaciones nativas de Android desarrolladas con Kotlin ofrecen una experiencia de usuario de vanguardia y son una opción sin compromiso para la calidad y el rendimiento.

La aplicación multiplataforma Flutter será más que capaz de ofrecer una experiencia nativa de alta calidad al usuario. La ventaja más obvia de Flutter es su capacidad multiplataforma. Una aplicación Flutter terminada será indistinguible de una aplicación móvil 100% nativa.

En conclusión, Dart es más sencillo para el desarrollo de aplicaciones móviles mientras que el de Kotlin es más como una navaja suiza con sus nuevas herramientas para los desarrolladores siendo complejo para principiantes en el mundo del desarrollo de aplicaciones.

**Diferencias entre Flutter y Java**

Flutter es una herramienta de "Desarrollo móvil multiplataforma" y Java es un "lenguaje". Basándonos en estos dos factores, podemos diferenciar ambos. Vamos a entender la diferencia clave entre Flutter y Java.

*Tabla 5: Diferencias entre Flutter y Java*

| Número | Flutter | Java |
|---|---|---|
| 1 | Es una plataforma o herramienta para desarrollo de aplicaciones multiplataforma. | Es un lenguaje utilizado para desarrollar y diseñar aplicaciones multiplataforma. |
| 2 | Utiliza la programación de datos para escribir código | En sí mismo un lenguaje de programación para escribir código |
| 3 | Es de libre acceso | También está disponible gratuitamente en el mercado |
| 4 | Proporciona una interfaz de usuario eficaz y flexible. | La interfaz de usuario no es tan expresiva o efectiva |
| 5 | Proporciona diseño de materiales | No proporciona diseño de materiales |

*Tabla 6: Herramientas de Flutter y Java*

| Número | Flutter | Java |
|---|---|---|
| 1 | Firebase | Docker |
| 2 | Android SDK | Intellij IDEA |
| 3 | Dart | Spring Boot |

| | | |
|---|---|---|
| 4 | Socket.IO | Scala |
| 5 | Outbrain | Android SDK |
| 6 | Google AdMob | Spring |
| 7 | Agora | Eclipse |
| 8 | Razorpay | Datadog |

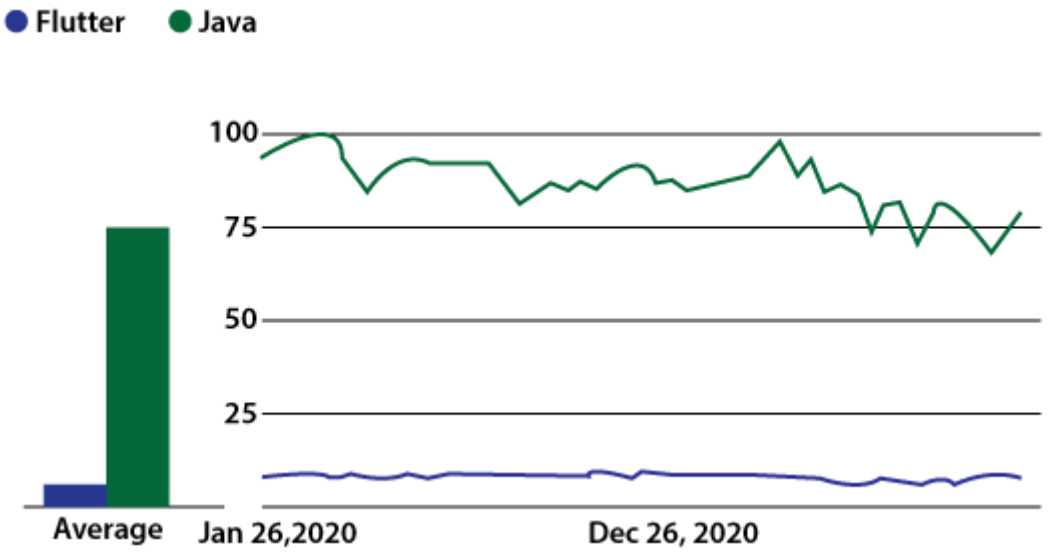



*Fig. 20: Uso de Flutter y Java*

**Flutter vs iOS (Swift)**

Swift es el lenguaje de codificación para el desarrollo de aplicaciones en iOS. Apple desarrolló Swift como un lenguaje con seguridad y dinámico. Una de las características más populares en el mundo del desarrollo es su evolución constante con nuevas tendencias

Swift proporciona a los desarrolladores de iOS herramientas eficientes, una amplia documentación y una comunidad global de Swift. Esto hace que el desarrollo de aplicaciones de iOS sea fácil y divertido.

La capacidad de Flutter para permitir el desarrollo de aplicaciones de iOS y Android desde un solo código fuente lo hace excepcionalmente útil. Está surgiendo una comunidad de usuarios de Flutter. Esto hará que el desarrollo de aplicaciones híbridas sea más dinámico.

**Comparación de las características:**

1. **Tiempos de desarrollo**

Con Swift, puede medir el tiempo de desarrollo de su aplicación iOS nativa utilizando la línea de comandos de Xcode o la compilación de Xcode. Cuando se ejecuta el comando de compilación desde el menú del producto, el tiempo de compilación se muestra en la pantalla. Para Swift, una construcción limpia generalmente toma alrededor de 13,33 segundos. Sin embargo, la compilación continua en Xcode tarda menos de un segundo.

Si usa Flutter, puede usar los mismos comandos para calcular el tiempo de desarrollo. Resulta que desarrollar aplicaciones de iOS en Flutter es relativamente más lento que en Swift. Sin embargo, la situación cambia una vez que se supera el umbral para la primera construcción limpia. Entonces Flutter es un poco más rápido.

2. **Incorporación de aplicaciones**

Swift le permite desarrollar aplicaciones iOS utilizando herramientas nativas de Apple. Se requiere Xcode como entorno de desarrollo integrado. Después de proporcionar la entrada, Xcode generará el código de muestra de iOS. Esta plantilla se puede utilizar para codificar su aplicación. Cuando se completa la programación de la aplicación iOS, aparece la pantalla del botón [Push Me]. En este punto, el desarrollo de la aplicación iOS está completo.

Para Flutter, necesitas usar los binarios Xcode y Flutter. También puede usar Android Studio, IntelliJ IDEA o cualquier otro editor de texto. Ejecute el comando Flutter Medical para verificar todos los reclamos. Luego, ejecute los comandos necesarios para crear su aplicación Flutter.

### 3. Recarga de aplicaciones

Para desarrollar aplicaciones móviles con Swift, debe cambiar los nombres de los nodos, los datos y la accesibilidad. También debe asegurarse de que el emulador o dispositivo refleje los cambios realizados. La función de recarga le ayuda a realizar cambios en tiempo real en su aplicación. Esto le permite acelerar el desarrollo de aplicaciones iOS y realizar evaluaciones precisas.

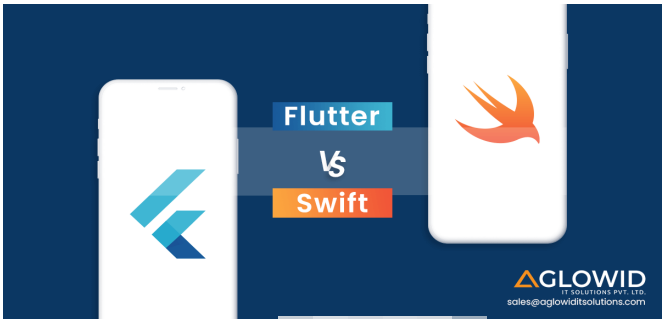
Flutter te da acceso a esta fantástica función llamada Hot Reload. Actualiza todos los cambios en su aplicación en segundos. Entonces, cuando recargues la aplicación, verás que Flutter es más rápido que Swift.

### 4. Pruebas y documentación de la aplicación

Las pruebas son significativamente diferentes para Swift y Flutter. Para las aplicaciones nativas de iOS con Swift, Apple ofrece XCTest. Esta es una plataforma muy eficiente para probar aplicaciones nativas. Esta herramienta proporciona una variedad de funciones de prueba, como pruebas de interfaz de usuario, pruebas de rendimiento y pruebas de integración. Apple actualiza regularmente sus herramientas de prueba como parte de su paquete de desarrollo de aplicaciones iOS.

Flutter también proporciona un poderoso marco de prueba como parte del kit de desarrollo. Flutter facilita a los desarrolladores la escritura de código limpio desde cero. Otras funciones de prueba hacen que el proceso y las funciones de prueba de la unidad

sean más eficientes. Además, Flutter también proporciona un proceso de documentación rápido para todos los proyectos de desarrollo de aplicaciones iOS. La función de documentación facilita a los desarrolladores de aplicaciones iOS la gestión del proceso




de desarrollo de la aplicación en cada etapa.

**¿Para qué se usa Flutter?**

Flutter contiene todo lo necesario para crear una interfaz de usuario, incluidos los widgets para Material Desing y Cupertino. Permite a los desarrolladores representar la interfaz de cada uno de los sistemas de manera más sencilla.

**¿Qué lenguaje de programación usa Flutter?**

El framework de Flutter, escrito en el lenguaje de programación Dart, tiene el motor Flutter, la biblioteca Foundation y widgets. El enfoque del desarrollo en Flutter difiere de otros por su escritura declarativa de la interfaz de usuario. Aquí, hay una necesidad de comenzar desde el final, lo que significa que antes de comenzar el desarrollo de algún elemento, el usuario debe tener en mente una imagen completa de qué tipo de interfaz de usuario será. Muchos desarrolladores distinguen esta escritura de interfaz de usuario como más clara, pero también causa ciertas dificultades para los desarrolladores al principio.


*Fig. 22:Estructura de Flutter*

### *3.12.5.1.* Flutter engine

Flutter sirve para crear interfaces de escritorio, web, móviles y hermosas a partir de una única base de código. Flutter cuenta con un código existente que es utilizado por desarrolladores y organizaciones de todo el mundo.

Flutter Engine es un repositorio de tiempo de ejecución portátil para alojar aplicaciones Flutter. Animación y gráficos, E / S de archivos y redes, compatibilidad con funciones, arquitectura de complementos, entorno de ejecución Dart y cadena de herramientas de construcción. La mayoría de los desarrolladores interactúan con Flutter a través de Flutter Framework. Proporciona un marco moderno y receptivo con un rico conjunto de utilidades, diseño y características en segundo plano. (Cardona, 2016)

### *3.12.5.2.* Foundation library

Las entidades definidas en esta biblioteca son las clases de utilidades de nivel más bajo y las funciones utilizadas por todas las demás capas del marco Flutter.

### *3.12.5.3.* Desing-specific widgets

En Flutter dispone de una extensa variedad de plugins para la creación de widgets ya sea para iOs, Android, Reac etc. Los widgets deben ser visibles, relevantes y personalizados. Esto significa que debe ser conciso y tener la intención del usuario principal. Debe resumir las características clave de su aplicación y aclarar su diseño. Los widgets pueden admitir diferentes tamaños. Elija el que mejor funcione para su aplicación y sus usuarios.

Los widgets están construidos utilizando el último marco inspirado en React. La idea principal es crear una interfaz a partir de un widget. Los widgets describen el aspecto de una vista en función de su configuración y estado actual. Cuando cambia el estado del widget, la utilidad reconstruye su descripción. Esto, a diferencia de la descripción anterior, determina los cambios mínimos necesarios para que el árbol de procesamiento subyacente se mueva de ese estado al siguiente. (Clown. M, 2019)

```dart
import 'package:flutter/material.dart';

 void main() => runApp(HelloWorldApp());

class HelloWorldApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {

    return MaterialApp(
       //El nombre de nuestra aplicación se determina con "title"
       title: 'Aplicación Hello World',


      home: Scaffold(
        appBar: AppBar(
          title: Text('Hola'),
        ),
        body: Center(
          // imprime "Hello World" en la pantalla.
          child: Text('Hello World'),

        ),
      ),
    );
  }
}
```

*Fig. 23: Ejemplo de código*

#### *3.12.5.4.* Ventajas

- Alto rendimiento.
- Curva de aprendizaje suave.
- Ayuda a crear excelentes diseños para aplicaciones móviles de Android e iOS.
- Reducción de gastos.
- Reducción de tiempos de desarrollo.
- Poderosa Comunidad.

#### *3.12.5.5.* Desventajas

- Falta de compatibilidad con el administrador de contraseñas
- Un conjunto limitado de herramientas y bibliotecas
- Limitación del tamaño de las aplicaciones
- No permite el uso de widgets no nativos de Flutter

### 3.12.6. Plataforma Firebase

**¿Qué es Firebase?**

Firebase es un base de datos en tiempo real referenciada al crecimiento de aplicaciones móviles y el manejo de información a través de ella. La mayoría de bases de datos necesitan una conexión a través de HTTP para obtener datos. Cuando una aplicación se conecta a Firebase lo hace a través de un websockets. Websocket es mucho más rápido que HTTP ya que con una sola conexión de socket es suficiente.

Todos los datos se sincronizan automáticamente a través de ese único websocket tan rápido como la red de su cliente los pueda transportar.

**Principales características de Firebase**

**Desarrollo:** Este contiene los servicios precisos para desarrollar proyectos de aplicaciones web o móviles. Algunas tareas se dejan a Firebase, por lo que pueden ayudar a acelerar el proceso, mientras que otras pueden optimizar varios aspectos para obtener la calidad que necesita.

**Analítica:** El análisis es necesario para tomar decisiones vinculadas a las estrategias de marketing asociadas con su proyecto. Con Firebase Analytics, puede controlar diferentes métricas y conseguir diferentes métricas de forma gratuita desde un solo panel.

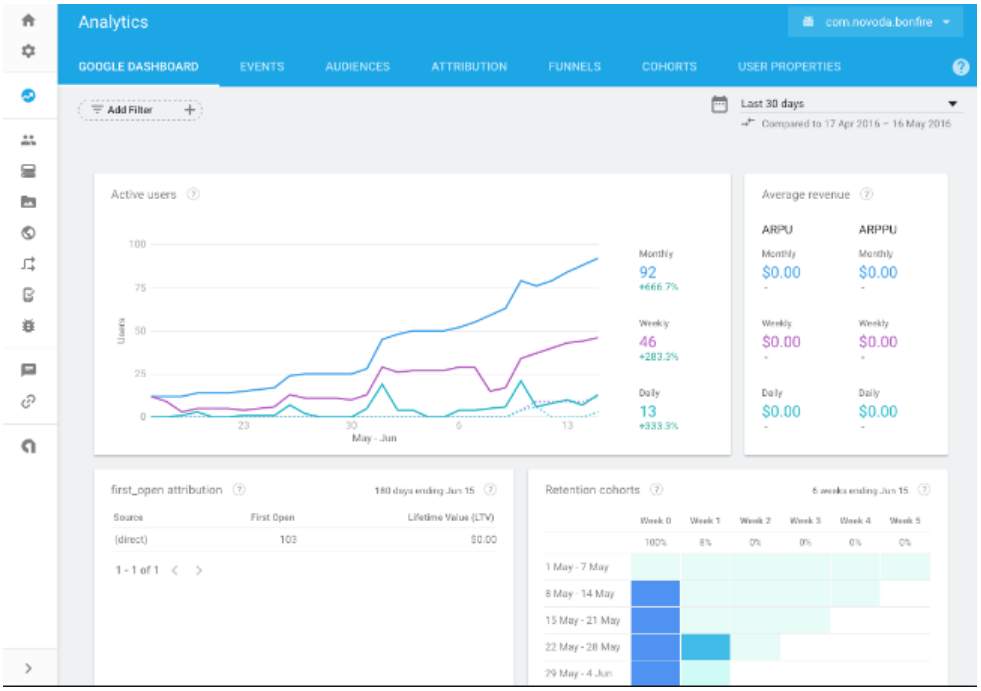



*Fig. 24: Panel de Análisis*

**Monetización**: Monetizar con Firebase es el tercer paso a considerar. En este caso, su búsqueda de ingresos está vinculada a anuncios que puede colocar en su aplicación para asegurarse de que sus usuarios reciban anuncios relevantes según el segmento proporcionado para su campaña.


*Fig. 25: Plataforma de Firebase*

### *3.12.6.1.* Firebase cloud messaging

Firebase Cloud Messaging (FCM) (anteriormente Google Cloud Messaging (GCM)) es un servicio gratuito en la nube proporcionado por Google que permite a los desarrolladores de aplicaciones enviar notificaciones y mensajes a los usuarios en múltiples plataformas, como Android, iOS y aplicaciones web. (Wieruch.R, 2021)

### *3.12.6.2.* Firebase auth

Firebase tiene un sistema de autenticación por correo electrónico, también siendo compatible por otros medios de autenticación como Facebook, Twitter, etc.

Firebase auth se consolida directamente con Firebase database, por lo que puede usarlo para controlar el acceso de sus datos.

#### *3.12.6.3.* Real time database

Firebase suministra una base de datos organizada de back-end en tiempo real. Esto facilita a los desarrolladores de aplicaciones una API que les permite sincronizar la información de la aplicación y almacenarla en la nube. Una de las características más útiles es que si un usuario no puede establecer una conexión estable. Firebase SK usa un caché local en el dispositivo, por lo que no se pierde información de ningún tipo y el usuario pierde la conexión a Internet. Cuando regrese a la conexión, sus datos se sincronizan automáticamente con sus datos locales. (Windmill.E, 2020)

#### *3.12.6.4.* Firebase storage

Firebase storage proporciona una forma sencilla de almacenar archivos binarios, la mayoría de veces imágenes entre otros, pero en si podría ser cualquier cosa la que se almacenaría en Google Cloud Storage.

#### *3.12.6.5.* Firebase cloud firestore

Cloud Firestore es una base de datos NoSQL que presenta diversas cualidades. Esta base de datos puede guardar diferentes tipos como: cadena de texto, tipos numéricos, coordenadas, arrays, booleanos entre muchos más que facilitan la manipulación de la información. (Wieruch.R, 2021)

#### *3.12.6.6.* Ventajas

Estos son los diez beneficios principales de usar Firebase.

- Datos en tiempo real
- Almacenamiento de archivos respaldado por Google Cloud Store
- Tratar los datos como flujo para crear aplicaciones altamente escalables

#### *3.12.6.7.* Desventajas

- Capacidad de consulta limitadas debido al modelo de flujo de datos.
- Los modelos de datos relacionales tradicionales no son aplicables a NoSql
- No requiere instalación local

### 3.12.7. Comparativa con la producción de apps en su lenguaje nativo

**Desarrollo Nativo**

Este es el desarrollo de una aplicación cuyo código base está escrito para ejecutarse en un sistema operativo en particular (iOS o Android o cualquier otro sistema operativo). Por ejemplo, Instagram se ve prácticamente igual en ambos sistemas operativos, pero está representado por dos aplicaciones nativas con diferentes bases de código. La aplicación está especialmente creada para usuarios de destino en ambos sistemas operativos.

**Ventajas**

- Rapidez
- Pueden acceder a funciones nativas de un teléfono

Existen importantes beneficios al utilizar el desarrollo nativo. Tiene acceso a todas las funciones de su teléfono, incluido el GPS, la cámara y el altavoz. Además, puede usar fácilmente las notificaciones automáticas para permitir que los propietarios de aplicaciones sean más productivos para sus usuarios.

**Desventajas**

- Gastos elevados
- Desarrollo que consume recursos

Las aplicaciones nativas de iOS no se pueden ejecutar en Android. Lo contrario también es cierto. Por lo tanto, para aprovechar al máximo su producto, necesita dos bases de código separadas. Esto significa que tienes que gastar mucho dinero porque tienes que contratar dos equipos. Se necesita mucho tiempo para desarrollar y entregar una aplicación. También complica el soporte más adelante, ya que necesita solucionar dos aplicaciones nativas en lugar de solo una.

**Desarrollo multiplataforma**

Este estilo de desarrollo de aplicaciones nos permite que un producto se ejecute en múltiples sistemas operativos móviles y esté escrito en un solo lenguaje de programación. Cuando el código de su aplicación esté listo, conviértalo a la API nativa de iOS o Android a través de middleware.

**Ventajas**

- Periodos cortos de desarrollo
- Código reutilizable
- Reducción de gastos

Este tipo de desarrollo móvil le permite no perder la oportunidad de capturar una parte significativa del mercado móvil en aplicaciones que se adaptan a los principales sistemas operativos. Puede ahorrar dinero porque solo necesita el código base para ejecutar su aplicación en diferentes sistemas operativos. El desarrollo multiplataforma le brinda una ventaja cuando se trata de crear aplicaciones atractivas para usuarios de Android, iOS y otros sistemas operativos de forma gratuita.

**Desventajas**

- Limitación a funciones nativas del teléfono
- Experiencia de usuario no tan buena

Durante el desarrollo multiplataforma, las capacidades de todos los dispositivos son diferentes, lo que limita las capacidades de los conjuntos de funciones de un teléfono. Como resultado, cada dispositivo tiene sus propias características y plataformas únicas, lo que puede generar problemas en la experiencia del usuario. Por lo tanto, las aplicaciones móviles multiplataforma son menos competitivas que las aplicaciones nativas. (Hernández.M, 2019)

### 3.12.8. Minimización de tiempos de producción

La reducción de los tiempos de desarrollo de aplicaciones multiplataforma está muy recalcada en el que el desarrollo va centrado en una sola aplicación sin importar el sistema ya sea Android, iOS o aplicaciones web.

Las aplicaciones híbridas se trabajan en su único código base para todos los sistemas, por ello, con los mismos recursos, el tiempo de desarrollo de este tipo de aplicaciones podría ser más corto. En ocasiones nos podemos encontrar con apps que requieran una diferenciación de acuerdo a la plataforma. En este caso se requiere un doble desarrollo, pero siempre ser un subconjunto del mismo proyecto así evitando el desarrollo de la misma aplicación para otras plataformas.

La inclusión de nuevas funcionalidades en aplicaciones híbridas es mucho más corta en comparación a las nativas de mantener nuevas funcionalidades en diferentes códigos base, por lo que en aplicaciones híbridas podemos producir nuevas funcionalidades de forma más rápida.

Igualmente, al realizar cambios en el diseño las aplicaciones híbridas cuentan con mayor facilidad y flexibilidad a la hora de modificar elementos de la UI (motion, vistas, controles personalizados, etc.)

En la siguiente imagen podemos visualizar los costos de desarrollo de acuerdo a cada región del mundo:

| Costo de desarrollo apps móviles por hora | | | |
|---|---|---|---|
| Región | iOS USD$/hora | Android USD$/hora | Híbrido USD$/hora |
| Norteamérica | $150 | $168 | $174 |
| Australia | $110 | $110 | $121 |
| Europa | $70 | $70 | $77 |
| América Latina | $43 | $34 | $42 |
| India | $30 | $26 | $38 |
| Indonesia | $11 | $12 | $12 |

*Fig. 26: Costos de desarrollo apps móviles por hora*

**¿Cuánto cuesta crear una app móvil a nivel mundial?**

Según estudios realizados los costos por el desarrollo de aplicaciones móviles ya sean nativas o híbridas es cambiante de acuerdo la zona del mundo y esto se debe a los recursos del país a disposición de los programadores ya sean de una empresa o freelance.

**Por tipo de profesional**

El tipo de profesional va en discusión ya que podemos ver que una empresa de desarrollo cuenta con costos más elevados por la utilización de inmuebles aun así brindándonos una gran ventaja sobre los tiempos de desarrollo gracias a que habrá más personal realizándose. El caso contrario sería contratar un desarrollador freelance a menor costo ya que ellos no necesitan de una oficina para trabajar.

**Por localidad**

Los valores de desarrollo de aplicaciones móviles varían de acuerdo al tamaño del proyecto como también de la localidad en la que se solicita el trabajo.


*Fig. 27: Costos de desarrollo de apps móviles en España*

Los costos al contratar un freelance o a una empresa fluctúan alrededor de en un 50% ya que uno es dependiente de sus instalaciones para lograr a cabo el pedido mediante el otro no depende de qué lugar del mundo se encuentre solo dependiendo de la conexión a internet estable. (Yeeply, 2019)

Los costos también pueden variar en el caso del desarrollo de aplicaciones nativas o multiplataforma como lo muestra la siguiente imagen.


*Fig. 28: Costos de apps multiplataforma y nativas en España*

La medida en la que vemos el alza de los precios ya sea por producir un app multiplataforma o nativa de acuerdo al personal que solicitamos contratar y la fiabilidad del mismo.

En resumen, el desarrollo de aplicaciones nativas es mayor en comparación al desarrollo de apps multiplataforma teniendo en cuenta el grupo de desarrolladores que se van a contratar ya sea un freelance o una empresa de desarrollo. (Yeeply, 2019)

### 3.12.9. Comparativa de arquitecturas

Uno de los puntos más importantes al elegir un lenguaje de programación es el tener en claro si la plataforma va dirigida a un solo sistema o es multiplataforma ya que la arquitectura es la misma pero los tiempos de desarrollo varían demasiado.

Como lo vemos en la arquitectura de iOS en la figura 29 y como también podemos ver en la arquitectura de Android en la figura 6 donde vemos que la

arquitectura de ambos sistemas no se diferencias en gran parte al ser divididas en capas y dirigidas a los desarrolladores para que trabajen en capas superiores del sistema.

Es muy importante saber la arquitectura de cada uno de los sistemas a los que vamos a dirigir nuestro desarrollo ya que las aplicaciones nativas pueden acceder a funcionalidades de cada dispositivo donde su sistema esté basado en la máquina virtual en la cual se programa.

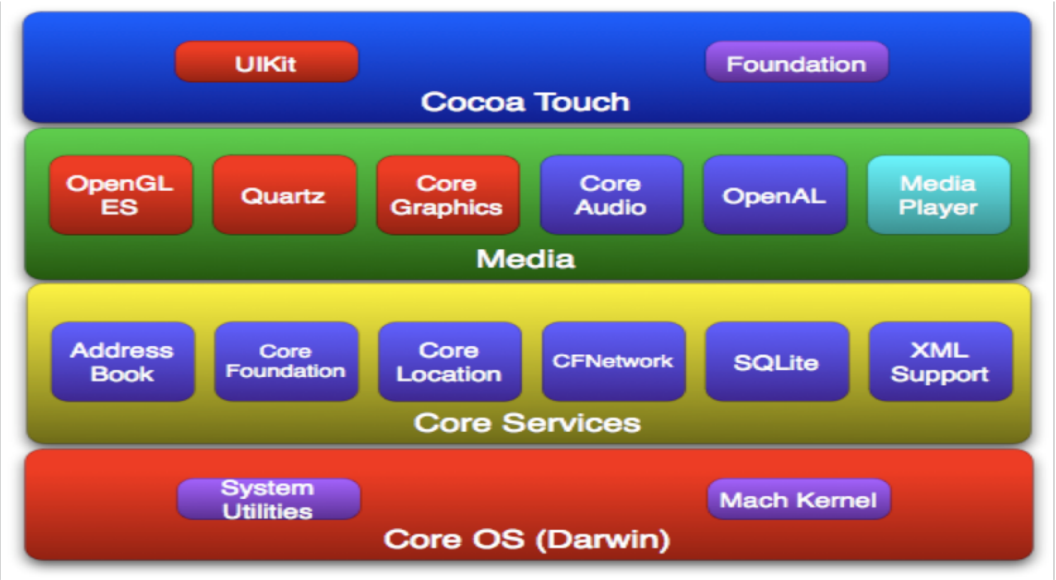



*Fig. 29: Arquitectura de apps móviles.*

# 4. Conclusiones y recomendaciones

## 4.1.Conclusiones

En conclusión, el uso de herramientas como Flutter y Firebase nos ayudan a tener un mejor desempeño en el desarrollo de aplicaciones móviles sin tener en cuenta las limitaciones de las mismas, hoy en día el desarrollo de aplicaciones multiplataforma está tomando fuerza al existir tantos medios por los cuales se puede llegar a los usuarios. Más del 50% de personas que han utilizado Flutter sienten que es una de las herramientas más completas en el mercado para desarrollar aplicaciones multiplataforma ofreciéndonos una gama extensa de herramientas que ayuda a desarrollar interfaces de usuario de manera más sencilla.

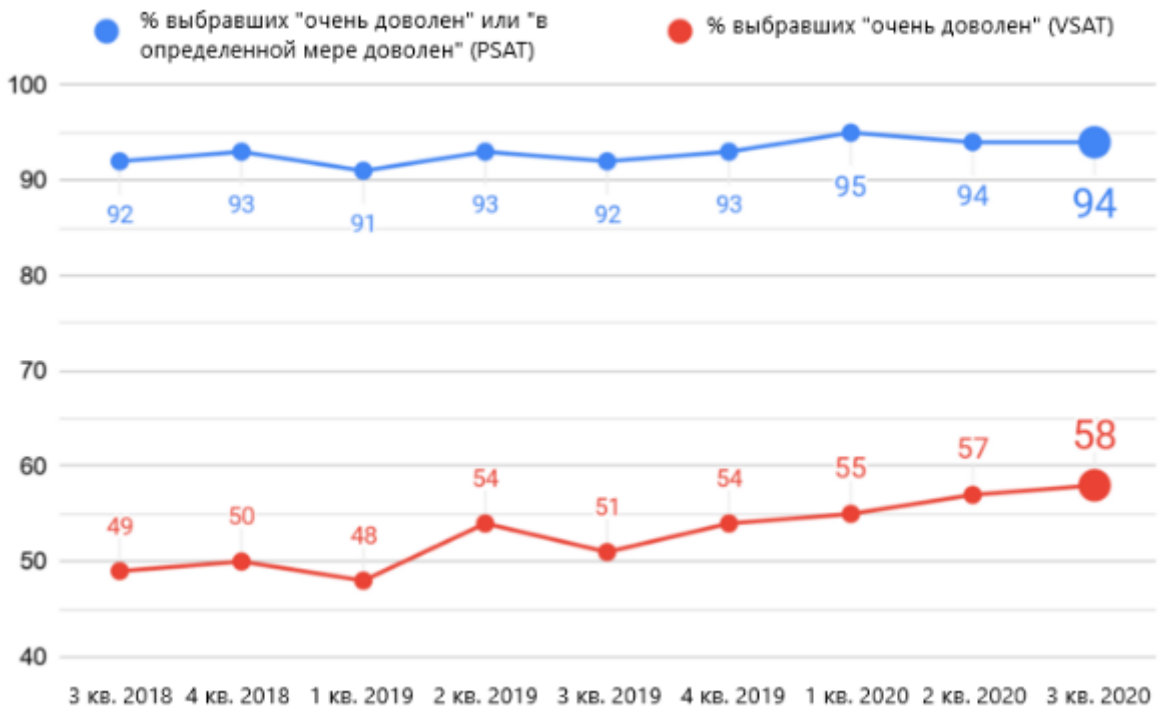



*Fig. 30: Satisfacción de los usuarios que usaron Flutter*

Como también el uso de widgets en las interfaces de usuario en los cuales se puede presentar de manera concreta la información hacia el usuario final. Lo que no vemos en el desarrollo de aplicaciones en su lenguaje nativo como tal.


*Fig. 31:Widgets*

## 4.2.Recomendaciones

Se recomienda a los desarrolladores que ingresan al mundo multiplataforma, tener un análisis previo del mercado al que se va dirigido, ya que todo esto puede influenciar en la decisión de salir con una aplicación multiplataforma o nativa. El hacer el desarrollo multiplataforma tiene como gran ventaja la cual es tomar como mercado dos de los grandes sistemas operativos dominantes en el mundo entre otros y así generando ganancias al hacerlo. Como también la limitación de no poder ser de gran tamaño o de no poder acceder a funciones específicas de cada dispositivo.

# 5. Anexos

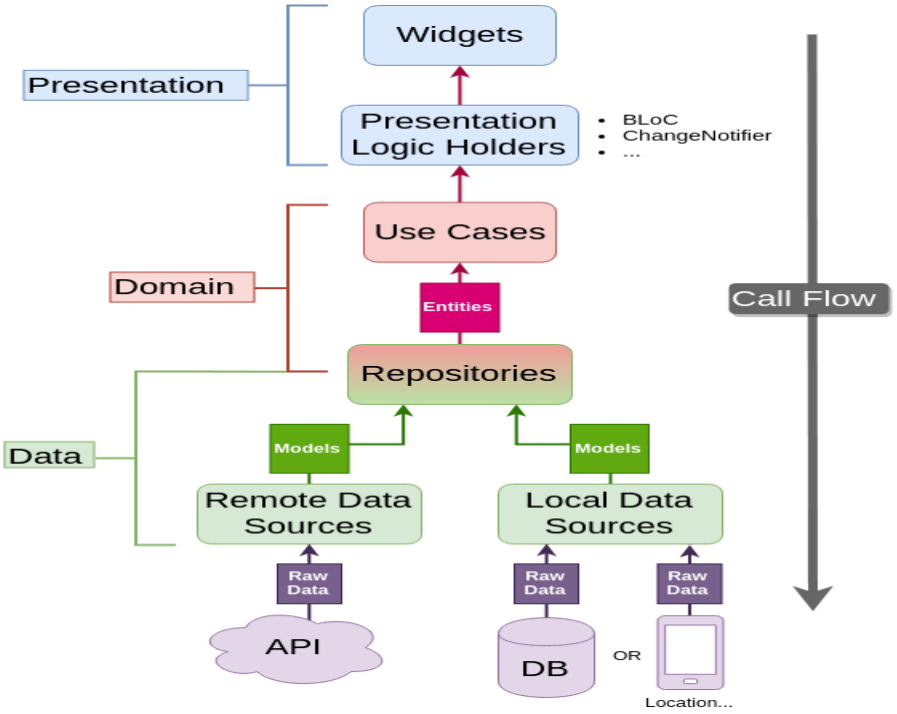



*Fig. 32: Arquitectura de Flutter*

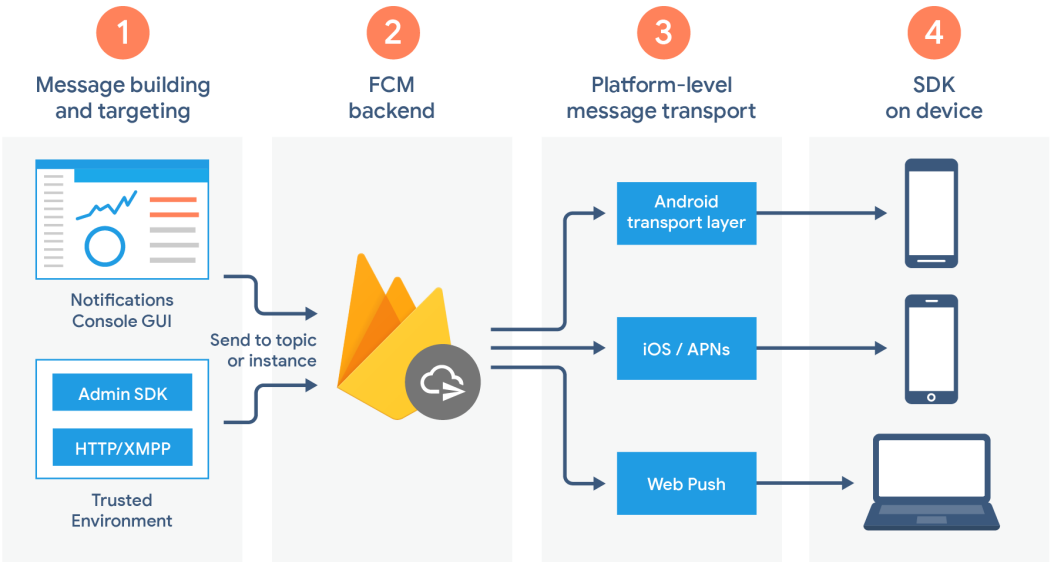



*Fig. 33: Arquitectura de FMC*

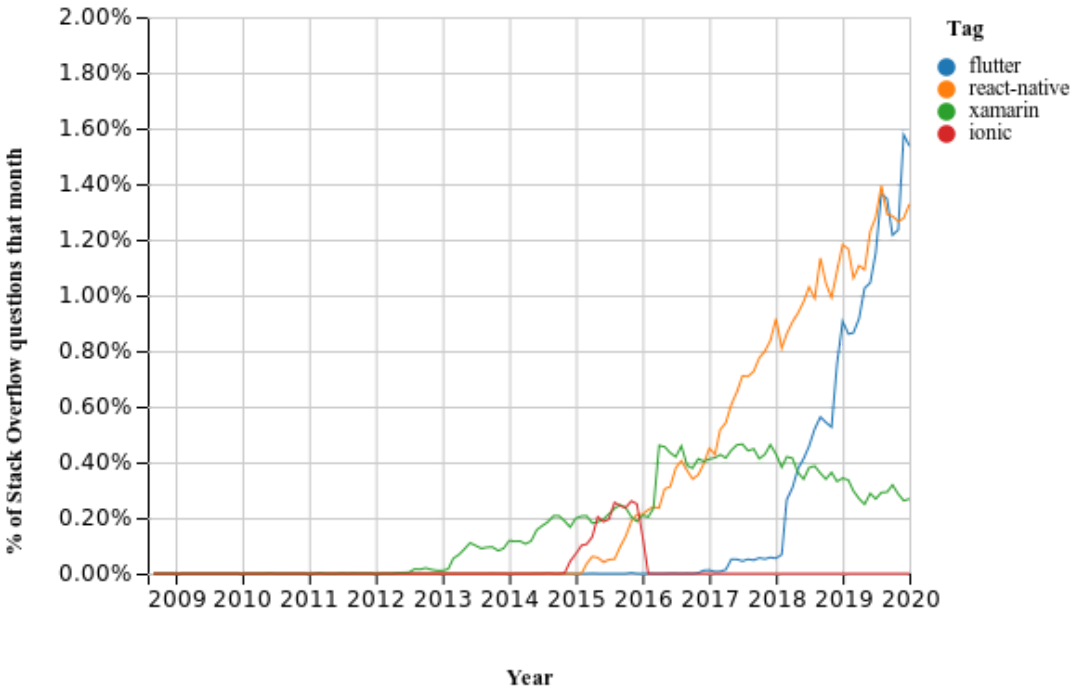



*Fig. 34:Crecimiento de Flutter desde el 2009*

# 6. Relación del trabajo desarrollado con los estudios cursados

En base a los cursos realizados hemos deducido que el aprendizaje de nuevas tecnologías que puedan abarcar varios entornos, es de mucha ayuda al momento de dirigirnos al mercado de desarrollo móvil, ya sea por la amplia gama de dispositivos que se puede dirigir o por los sistemas que se pueden abarcar, así teniendo en cuenta el desarrollo de aplicaciones móviles en su lenguaje nativo donde vemos que nos dan más funcionalidades gracias al acceso de las características internas de cada dispositivo pero con costos mayores de desarrollo.

La creación de aplicaciones multiplataformas está tomando el mercado mundial como la aplicación Apptree una plataforma de uso profesional y comercial por parte de McDonalds, Wayfair y Fermilab Stanford, que ofrece a todas las compañías que la usan al acelerar sus negocios y al resolver problemas comunes. Así viendo la potencia que tiene el desarrollo de aplicaciones basadas en Flutter acompañado de Firebase.

A lo largo de la carrera todos los conocimientos adquiridos han hecho pensar que el desarrollo nativo está bien, en condiciones en las que las aplicaciones deban realizar ciertas funcionalidades en dicho sistema, esto lo podemos realizar cosas como: desarrollar aplicaciones multiplataforma, abarcar varios dispositivos y sistemas al brindar la misma funcionalidad reduciendo el rendimiento de la aplicación en un 1%.

# 7. Referencias bibliográficas


1. Ionos, M. p. (09 de 10 de 2020). *Introducción al framework multiplataforma* . Obtenido de digital guide ionos: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-flutter/

2. *Arquitectura de la plataforma*. (07 de 05 de 2020). Obtenido de developers: https://developer.android.com/guide/platform?hl=es-419

3. CARDONA, M. P. (14 de 10 de 2016). *Firebase, qué es y para qué sirve la plataforma de Google*. Obtenido de iebschool: https://www.iebschool.com/blog/firebase-que-es-para-que-sirve-la-plataforma-desarroladores-google-seo-sem/

4. Clow, M. (2019). *Learn Google Flutter Fast: 65 Examples Apps*. Reino Unido: Independently

5. *Cuánto cuesta crear una app*. (01 de 06 de 2019). Obtenido de Yeeply: https://www.yeeply.com/blog/cuanto-cuesta-crear-una-app/

6. HERNÁNDEZ, M. (20 de 05 de 2019). *Apps Nativas vs Multiplataforma*. Obtenido de Pixzelle Studio: https://www.pixzelle.mx/blog/apps-nativas-vs-multiplataforma

7. Miguel, M. (10 de 06 de 2021). *Historial de versiones de Android*. Obtenido de wikipedia: https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android

8. *Qué es una arquitectura de aplicaciones*. (14 de 06 de 2020). Obtenido de Redhat: https://www.redhat.com/es/topics/cloud-native-apps/what-is-an-application-architecture.

9. Windmill, E. (2020). *Flutter in Action.* EE. UU: Manning Publications

10. Wieruch, R. (2021). *The Road to Firebase: Your journey to master Firebase in JavaScript.* EE.UU. Independently