



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

ANÁLISIS DEL KIT DE HERRAMIENTAS FLUTTER Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN

Alumno/a: Moreira da Silva, Adson Henrique

Tutor/a: Prof. D. Víctor Manuel Rivas Santos
Dpto.: Informática

Mayo, 2021



Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Víctor Manuel Rivas Santos, tutor del Trabajo Fin de Master titulado: Análisis del kit de herramientas Flutter y desarrollo de un prototipo de aplicación multiplataforma, que presenta Adson Henrique Moreira da Silva, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, 04 de mayo de 2021

El alumno:

El tutor:

Adson Henrique Moreira da Silva

Victor manuel Rivas Santos

Agradecimientos

Doy gracias a la Universidad de Jaén por haberme concedido la Beca de Atracción de Talento, con la cual me ha proporcionado más una forma de abrir una nueva jornada en mi vida profesional y personal. A todos los profesores de ese máster, que a pesar de lo ocurrido en esos años difíciles debido al covid-19 y la pandemia, continuaran fuertes e hicieron lo mejor que podían, sin dudas, para que nosotros estudiantes no resultásemos perjudicados.

Sin embargo, también tengo que agradecer a mi familia en Brasil, porque sin ellos nada de eso sería posible. Gracias a mi compañero de máster Álvaro, por su amistad durante el tiempo en que estuvimos juntos en las clases presenciales y telemáticas. Y también por toda la ayuda que me ha proporcionado con el idioma, con las asignaturas, hasta en algunas correcciones en ese documento.

No obstante, muchas gracias a mi amigo Marcos de Brasil, que, sin duda, fue una persona importante mientras estaba desarrollando ese documento. Por toda la colaboración hecha, todo el apoyo, soporte y por su incentivo.

“Una vida humilde y tranquila trae más felicidad que la persecución del éxito y la constante inquietud que implica”, (Albert Einstein)

Índice

1. INTRODUCCIÓN.....	10
1.1. Objetivos.....	11
2. EL CAMINO HASTA FLUTTER.....	12
2.1. La rápida ascensión de internet y móviles.....	12
2.2. Android y iOS dominan el mercado	15
2.3. El mercado de aplicaciones	16
2.4. El mercado de desarrollo de software móvil nativo	16
2.5. Herramientas para desarrollo móvil multiplataforma	17
2.5.1. Xamarin	18
2.5.2. PhoneGap	18
2.5.3. Titanium	18
2.5.4. React Native.....	18
2.6. Herramientas multiplataformas.	18
2.6.1. Aplicaciones móviles web	19
2.6.2. Aplicaciones web híbridadas.....	20
2.6.3. Aplicaciones interpretadas	21
2.6.4. Aplicaciones generadas por multiplataformas.....	22
2.7. El problema de herramientas multiplataformas	23
3. FLUTTER.....	24
3.1. ¿Para qué sirve Flutter?	24
3.2. Principales Partes de Flutter	24
3.2.1. Flutter <i>Engine</i>	25
3.2.2. <i>Foundation Library</i>	25
3.2.3. Dart.....	25
3.2.4. Widgets.....	26
3.3. Ventajas	27
3.3.1. <i>Hot Reload</i>	28
3.3.2. Multiplataforma de verdad	28
3.3.3. Dart.....	28
3.3.4. Widgets.....	28
3.3.5. Herramientas.....	28
3.4. Desventajas	29
3.4.1. Herramienta nueva.....	29
3.4.2. Mezcla de código	30

3.4.3.	Google	31
3.4.4.	Árbol de Widgets	31
3.4.5.	Tamaño de las aplicaciones	32
3.4.6.	Programación reactiva y gestión de estado	32
3.5.	¿Porque utilizar Flutter?	33
3.6.	¿Quién usa Flutter y por qué?	35
3.7.	¿Cuánto cuesta Flutter?	35
3.8.	¿Como aprender Flutter?	36
3.9.	¿Hay algún tipo de certificación para desarrolladores?	38
3.10.	¿Puede interactuar con otras tecnologías?	39
3.11.	Profundizando en Flutter	40
3.11.1.	Añadiendo Flutter a una aplicación ya existente	40
3.11.2.	<i>Null Safety</i>	43
3.11.3.	Animaciones y transiciones	45
3.11.4.	Categoría de Widgets	47
3.11.5.	El lenguaje Dart.....	50
3.12.	Flutter 2.0.....	52
4.	NEAT: CASO PRÁCTICO. DESARROLLO DE UN PROTÓTIPO DE APLICACIÓN MULTIPLATAFORMA UTILIZANDO FLUTTER.....	54
4.1.	Herramientas y tecnologías utilizadas	54
4.1.1.	Visual Studio Code	54
4.1.2.	Git.....	55
4.1.3.	Notion	55
4.1.4.	Whimsical	55
4.1.5.	AVD Manager.....	56
4.1.6.	SQFLITE.....	56
4.1.7.	Flutter Localizations	56
4.1.8.	Shared Preferences	56
4.1.9.	Intl	57
4.1.10.	Flutter Launcher Icons	57
4.1.11.	Flutter Calendar Carousel	57
4.1.12.	Flutter Slidable	57
4.1.13.	Scoped Model	58
4.1.14.	Figma.....	58
4.1.15.	Barcode Scan.....	58
4.1.16.	QR Flutter.....	58
4.2.	Metodología de desarrollo.....	59

4.2.1.	Definición de requisitos	60
4.2.2.	Casos de uso	62
4.3.	Implementación e Validación de la aplicación	63
4.3.1.	Estructura del árbol de widgets.....	63
4.3.2.	<i>OnBoarding</i>	66
4.3.3.	Prototipos del diseño de la aplicación	67
4.3.4.	Base de datos	68
4.3.5.	Estructura del proyecto	69
4.3.6.	La lógica por detrás de Scoped Model	72
4.3.7.	Informaciones adicionales	77
4.3.8.	Validación de la aplicación.....	77
5.	CONCLUSIONES Y LÍNEAS FUTURAS	83
5.1.	Conclusiones	83
5.2.	Líneas futuras	84
6.	BIBLIOGRAFIA.....	85

APÊNDICES

1. MANUAL PARA EL USUARIO	89
1.1. Onboarding.....	89
1.2. Citas (<i>Appointments</i>)	90
1.3. Contactos (<i>Contacts</i>)	94
1.4. Notas (<i>Notes</i>)	98
1.5. Tareas (<i>Tasks</i>).....	101
1.6. Enlaces (<i>Links</i>).....	104

Tabla de Ilustraciones

Ilustración 1. Sistemas operativos más usados en el mundo hasta 2017	13
Ilustración 2. Sistemas operativos más usados en el mundo hasta noviembre de 2020.....	14
Ilustración 3. Evolución de los móviles y de la internet	14
Ilustración 4. Requisiciones a páginas de internet a partir de dispositivos móviles	15
Ilustración 5. Ejemplos de aplicaciones móviles web	19
Ilustración 6. Ejemplo de aplicación web híbrida	20
Ilustración 7. Ejemplos de aplicaciones interpretadas	21
Ilustración 8. Ejemplos de aplicación hecha con Xamarin	22
Ilustración 9. Lenguajes de programación más relevantes en 2019 para desarrolladores	26
Ilustración 10. Representación de widgets	27
Ilustración 11. Crecimiento de la búsqueda de Flutter en Google (2018 - 2020)	29
Ilustración 12. Fragmento de ejemplo de la mezcla de código en Flutter	31
Ilustración 13. Demostración del paradigma reactivo de programación y el gerenciamento de estados	33
Ilustración 14. Modo de construcción de widget en React Native	34
Ilustración 15. Diferencia de widget en Android y iOS	34
Ilustración 16. Página oficial de instalación de Flutter	36
Ilustración 17. Documentación de Flutter para desarrolladores de otras plataformas	37
Ilustración 18. Contenidos más avanzados y detallados de Flutter y sus canales de comunicación oficial.....	38
Ilustración 19. Flutter paquetes.....	40
Ilustración 20. Añadiendo Flutter a una aplicación Android	41
Ilustración 21. Añadiendo Flutter a una aplicación iOS	42
Ilustración 22. Ejemplo de código (Null safety)	43
Ilustración 23. Ejemplo de código utilizando el operador <i>late</i>	44
Ilustración 24. Ejemplo de un <i>AnimatedContainer</i>	45
Ilustración 25. Ejemplo de un <i>AnimatedCrossFade</i>	46
Ilustración 26. Ejemplo de un <i>AnimatedDefaultTextStyle</i>	46
Ilustración 27. Categoría de widgets	47
Ilustración 28. Ejemplo de gestos utilizando un móvil.....	49
Ilustración 29. Java X Dart.....	50
Ilustración 30. Maneras de declaración de variables en Dart	51
Ilustración 31. Alcance léxico en Dart.....	52

Ilustración 32. Ejemplo del paquete Flutter Slidable	57
Ilustración 33. Proceso iterativo e incremental	60
Ilustración 34. Casos de uso general	63
Ilustración 35. Estructura del árbol de widgets de la aplicación	64
Ilustración 36. Ejemplo de móviles con <i>notch</i>	65
Ilustración 37. Pestañas de la aplicación	66
Ilustración 38. Uso de <i>Shared Preferences</i> en <i>OnBoarding</i>	66
Ilustración 39. Prototipos de diseño de la aplicación	68
Ilustración 40. Modelo de diseño de la base de datos	69
Ilustración 41. Estructura del proyecto	72
Ilustración 42. Implementación de Scoped Model con la entidad de Notas	74
Ilustración 43. Implementación de Scoped Model con la entidad de Enlaces.....	75
Ilustración 44. Ejemplo del widget <i>ScopedModel</i> en el árbol de widgets	76
Ilustración 45. Usuarios que ya tenían conocimientos previos en desarrollo móvil	78
Ilustración 46. Fluidez de la aplicación	79
Ilustración 47. Percepción al utilizar la aplicación	80
Ilustración 48. Sugerencias de mejoras para la aplicación	81
Ilustración 49. Percepción de diferencias entre Neat y otras aplicaciones	82

Apêndices -Tabla de Ilustraciones

Ilustración 1. Pantallas de onboarding	89
Ilustración 2. Pantallas tras pantallas de onboarding	90
Ilustración 3. Pasos mínimos necesarios para añadir una cita	91
Ilustración 4. Pasos para cambiar fecha y/o hora	92
Ilustración 5. Pasos para editar una cita	93
Ilustración 6. Pasos para borrar una cita	94
Ilustración 7. Primera pantalla al acceder a contactos	95
Ilustración 8. Pasos mínimos necesarios para la creación de un contacto	96
Ilustración 9. Pasos necesarios para la edición de un contacto	97
Ilustración 10. Pasos necesarios para borrar un contacto	98
Ilustración 11. Pantalla inicial de la pestaña notas	99
Ilustración 12. Pasos necesarios para añadir una nota	99
Ilustración 13. Pasos necesarios para editar una nota	100
Ilustración 14. Pasos necesarios para borrar una nota	101
Ilustración 15. Primera pantalla al acceder la pestaña de tareas	102
Ilustración 16. Pasos necesarios para creación de una tarea	102
Ilustración 17. Pasos necesarios para la edición de una tarea	103
Ilustración 18. Pasos necesarios para borrar una tarea	104
Ilustración 19. Pantallas de la pestaña Enlaces	105
Ilustración 20. Pasos necesarios para crear un enlace	106
Ilustración 21. Pasos necesarios para editar un enlace	106
Ilustración 22. Pasos necesarios para borrar un enlace	107

1. INTRODUCCIÓN

El uso de los *smartphones* crece a cada año y la tendencia es que siga creciendo. Según la investigación de la agencia británica *We are Social* [7], en el año de 2019 un total de 5.19 mil millones de personas ya utilizaban móviles, un aumento de 2,4% en relación al año de 2018. Y con toda esa gente utilizando móviles (independientemente del motivo por el que lo hagan: redes sociales, compras, *internet banking*, o juegos, por citar solo algunas opciones), el valor añadido que proporciona a las compañías hace que el interés de las mismas crezca cada vez más. Para si tener una idea, en 2019, según la misma investigación *We are Social* [7], las personas gastaron cerca de 120 mil millones de dólares en aplicaciones móviles.

No es de extrañar, por tanto, que las compañías empiecen a desarrollar sus propias aplicaciones para las más variadas plataformas, en busca de conseguir más clientes y atender los que ya tiene. Dando acceso a través del móvil al conjunto de servicios que ofrecen, las compañías consiguen que su negocio esté disponible en la palma de la mano del consumidor final.

El problema para las empresas entonces es la variedad de plataformas existentes; esto obliga a que las empresas tengan que desarrollar su aplicación no solo una vez, sino tantas veces como plataformas desea utilizar. Si desea que su aplicación esté en los móviles que ejecutan el sistema operativo Android, hay que desarrollar la aplicación usando el kit de herramientas (SDK) puestas disponibles por su creadora Google; si desea desarrollar su aplicación para móviles que ejecuten el sistema operativo iOS, hay que utilizar el SDK de Apple y así sucesivamente en función de la plataforma deseada.

El problema de múltiples desarrollos es el coste que eso proporciona para las empresas; porque múltiples plataformas, generalmente, significa múltiples equipos de trabajo, cada uno especializada en una plataforma. Con eso en mente, muchas compañías empiezan a buscar soluciones en que la aplicación no necesite de múltiples desarrollos para múltiples plataformas, sino más bien que una aplicación sea desarrollada una sola vez y que sea portable para todas las otras plataformas. Así

surgen herramientas como React Native¹, Xamarin², PhoneGap³ y muchas otras. Como no puede ser de otra forma, cada una de estas herramientas ofrecen sus ventajas y desventajas para sus usuarios.

En este contexto es en el que aparece Flutter, una herramienta que te permite escribir la aplicación una vez y distribuirla para múltiples plataformas. Flutter surge con el propósito de ser la herramienta que acabe con todas las desventajas encontradas por las otras herramientas del mismo seguimiento.

A partir de eso, en ese trabajo se encontrará una breve historia hasta el surgimiento de Flutter. Luego, un análisis del kit de herramientas ofrecido por Flutter para sus usuarios; y, por último, una descripción de la metodología utilizada para el desarrollo de una aplicación multiplataforma utilizando Flutter.

1.1. Objetivos

En el actual Trabajo Fin de Máster se estipularon los siguientes objetivos:

- Analisar el kit de herramientas de Flutter, incluyendo todo lo que ello implica, como por ejemplo el lenguaje de desarrollo Dart que es utilizado en Flutter.
- Enfatizar las ventajas y desventajas de Flutter en comparación con otras herramientas conocidas que tienen objetivos similares a Flutter.
- Desarrollar un prototipo de una aplicación utilizando el kit de herramientas, previamente analizado, de Flutter.

¹ <https://reactnative.dev/>

² <https://dotnet.microsoft.com/apps/xamarin>

³ <https://phonegap.com/>

2. EL CAMINO HASTA FLUTTER

En este capítulo se aborda una historia concisa de la rápida evolución de internet y de los dispositivos móviles para entender como llegamos a herramientas como Flutter.

2.1. La rápida ascensión de internet y móviles

Es importante mirar al pasado y constatar que los móviles, para alcanzar el estado que conocemos hoy, tuvieron una rápida evolución. Hoy en día, los móviles tienen muchas más funcionalidades comparadas con sus antecesores. Hoy no sirven apenas para llamadas, en verdad es que poca gente lo utiliza para eso, sino que sirve para muchas cosas como banca por internet, juegos, capturar fotos, estar conectado con amigos por redes sociales, mirar noticias en tiempo real, ver televisión, hacer compras en línea, escuchar música y muchas otras. Y esta rápida evolución se debe casi que íntegramente a la revolución de internet, con la burbuja de las puntocom (medio de los años 2000) y con el éxito de las redes sociales. [1]

Siguiendo con el crecimiento de internet, muchas innovaciones llegaron para los móviles en los mediados de los años 2000. Fabricantes de móviles empezaron a construir funcionalidades como reproductores de MP3, consola de juegos portátil, entre otros. El mercado de móviles que antes era dominado por Nokia, Sony Ericsson, LG; a principio de los años de 2007, junto con el aparecimiento del término *smartphone*, el dominio pertenencia a fabricantes como HTC, BlackBerry y Apple (con el primero iPhone).

Con el éxito de redes sociales como Instagram o Facebook, los móviles adquirieron más funcionalidades. Por ejemplo, Instagram - app para compartir fotos - lanzado en 2010, trae la preocupación de las fabricantes para producir mejores cameras para sus teléfonos. Cuestiones antiguas como modo de espera, tiempos de llamadas, fueran cambiadas por otras como sistemas operativos, calidad de las cameras, capacidad de la batería o descargar datos de forma más rápida [2].

Con el pasar de los años, con el mercado de móviles dominado por Apple con su sistema operativo iOS y sus productos, Huawei y Samsung con sus móviles optando por usar el sistema operativo Android desarrollado por Google; otras

fabricantes como BlackBerry, Microsoft fueran perdiendo el mercado de los móviles y consecuentemente de sistemas operativos móviles. [3]

Adicionalmente, en 2017 Android se vuelve el sistema operativo más utilizado en el mundo, superando a Windows (sistema operativo para ordenadores de mesa), conforme muestra la encuesta, de la empresa de estadísticas web *StatCounter*. [4]

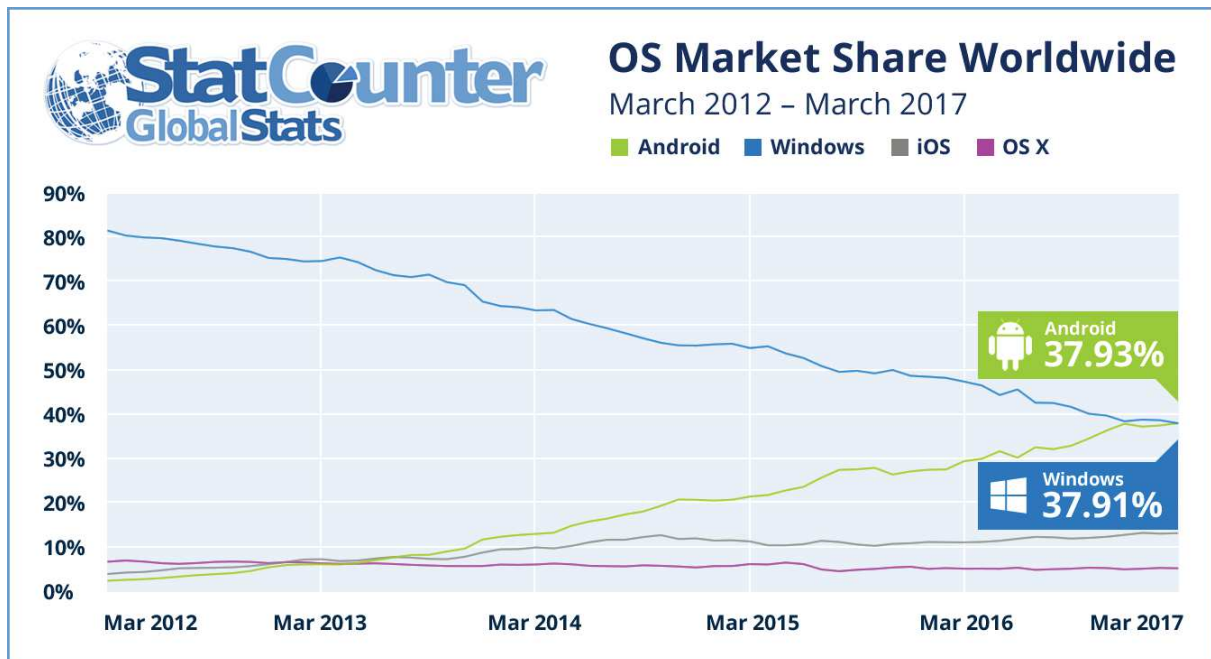


Ilustración 1. Sistemas operativos más usados en el mundo hasta 2017

Tomada de [4]

Según se evidencia en la ilustración 2, Android sigue superando a Windows en el sistema operativo más utilizado en el entorno global.

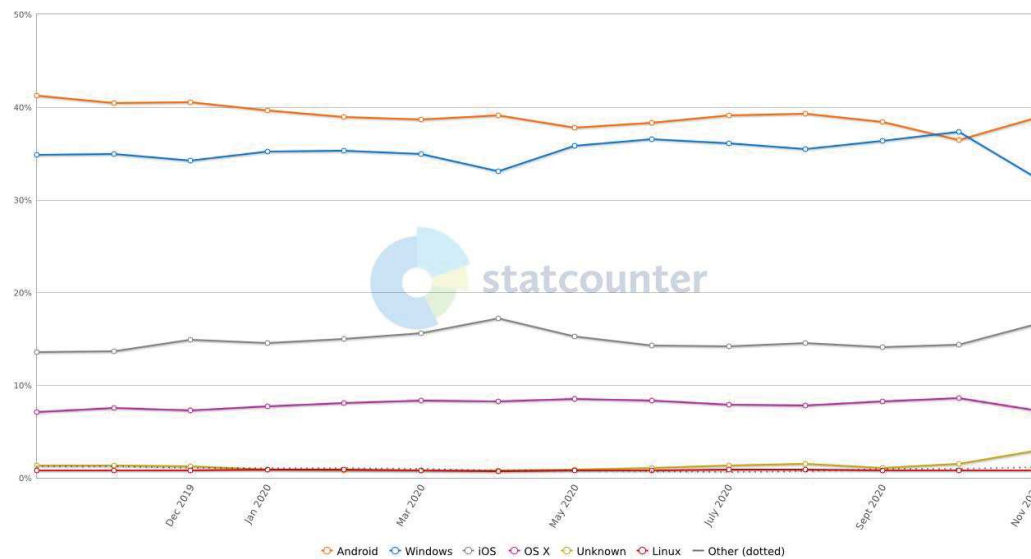


Ilustración 2. Sistemas operativos más usados en el mundo hasta noviembre de 2020

Tomada de [5]

La siguiente imagen (ilustración 3) muestra en resumen lo sucedido a lo largo del tiempo con la revolución de internet y la tecnología móvil.

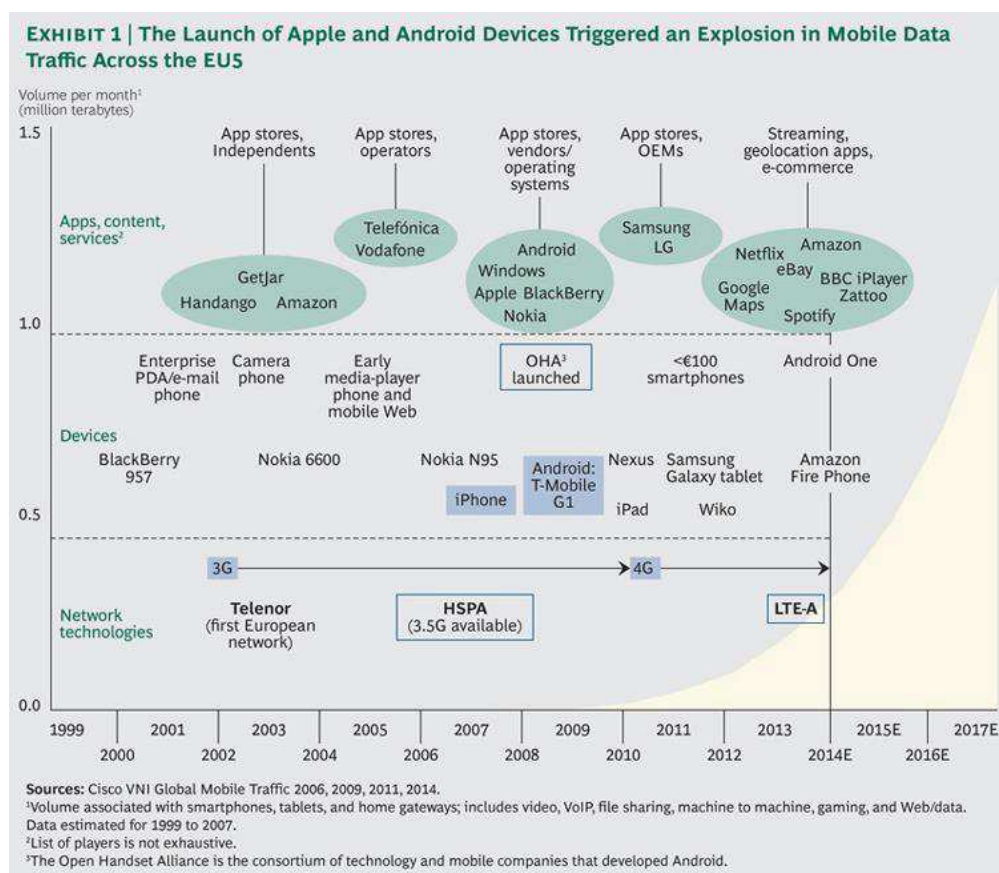


Ilustración 3. Evolución de los móviles y de la internet

Tomada de [6]

2.2. Android y iOS dominan el mercado

Con 5.19 mil millones de usuarios únicos en dispositivos móviles lo que representa el 67% de la población mundial y con 53% de las peticiones a páginas de internet hechas por móviles, según la investigación de *We are Social*; los *smartphones* están en una ascensión sin precedentes. [7]

Según la misma investigación, el sistema operativo Android es responsable del 74% de las solicitudes a páginas de internet originadas de dispositivos móviles, mientras iOS es responsable del 25% y el 1% restante es compartido con otros sistemas operativos. [7]

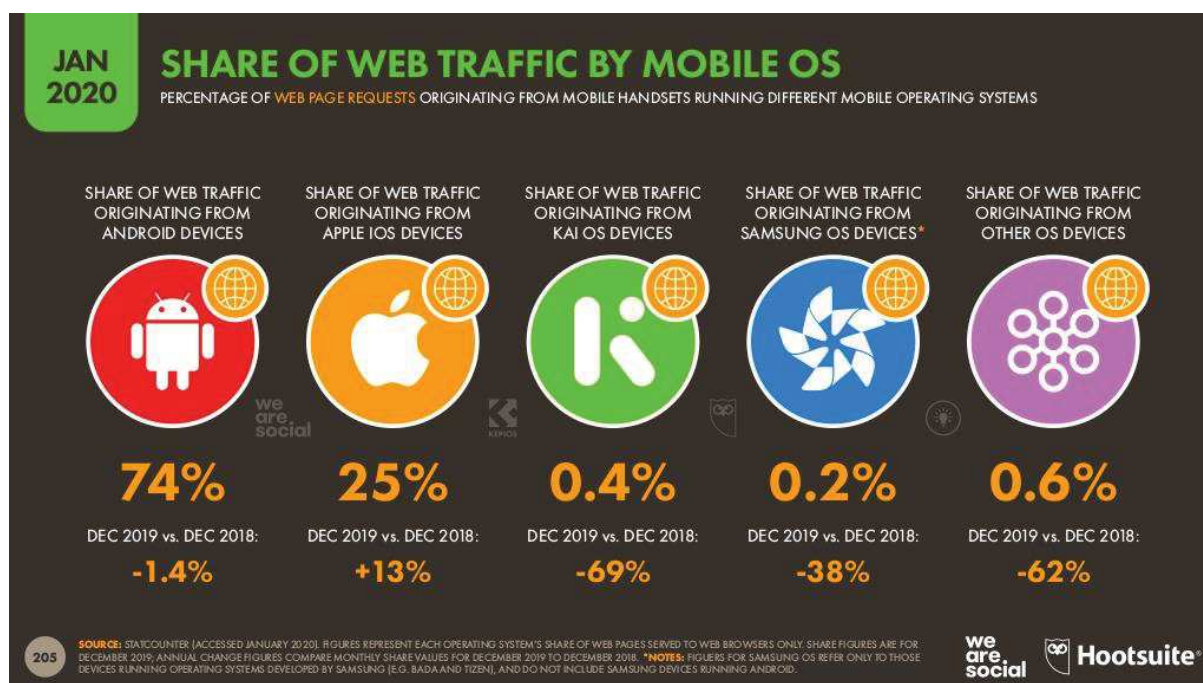


Ilustración 4. Requisiciones a páginas de internet a partir de dispositivos móviles

Tomada de [7]

Se puede evidenciar que Android y iOS dominando 99% el mercado. De un lado Android, de la gigante Google, sistema elegido por casi todas las fabricantes de móviles debido al código abierto - donde todas pueden descargar, modificar y redistribuir de forma gratuita. De otro lado iOS, de Apple, un sistema privado y que solo ella lo utiliza y lo modifica para un de sus principales productos, iPhone.

2.3. El mercado de aplicaciones

Las aplicaciones están presentes en la vida de todos los que tienen un móvil, sea para usar redes sociales, hacer compras, jugar, entretenimiento, escuchar canciones, mapas. Hoy en día es muy difícil que no haya alguna actividad en la cual ya no haya una aplicación que te permita hacerla a través de un móvil.

Según el informe anual de *We are Social*, pasamos en media 3 horas y 40 minutos por día usando el móvil y ese valor crece un 10% a cada año. De ese total de tiempo, pasamos el 91% de él en aplicaciones que descargamos o que vienen en el propio móvil. El otro 9% del tiempo lo pasamos usando los navegadores de internet presentes en el móvil. [7]

Para tener una idea, el número de descargas de aplicaciones en 2019 fue de 204 mil millones, ese número tiende a crecer un 6% al año. En el mismo año, los usuarios gastaron cerca de 120 mil millones de dólares en aplicaciones móviles y ese valor tiende a continuar creciendo a cada año, con una tasa del 20%.

Sabiendo esto, de todo el capital que los móviles están generando en los últimos años y que la tendencia es el crecimiento al largo de los años, la empresa que no tiene su negocio disponible online, sea por medio de aplicaciones, sea por medio de sitios web está perdiendo dinero y probablemente clientes.

2.4. El mercado de desarrollo de software móvil nativo

Hasta el surgimiento de Android y iOS el mercado de desarrollo de software móvil, en general, era pequeño y tímido. Los móviles, hasta entonces, no podían hacer grandes cosas, estando limitados a hacer llamadas, mensajes de texto, juegos y otras cosas más, pero con mucha limitación si los comparamos a los actuales.

Con el surgimiento de iOS y Android y debido a la rápida evolución de los *smartphones*, al gran valor agregado que los mismos traen para los usuarios, todo este mercado cambió. Ahora con esos sistemas operativos, con la popularización accesible de internet y de los móviles no hay justificación para que las empresas no desarrollaren sus propias aplicaciones y las ponga a disposición del cliente. Las empresas pueden elegir lanzar sus aplicaciones en una plataforma o en todas y las

que optan por lanzar en varias plataformas (sea cualquiera el motivo) tendrán que desarrollar la aplicación muchas veces, una para cada plataforma. Pero mirando la diferencia entre las plataformas y los sistemas operativos, esto no es lo común.

Enfocando más en las plataformas que dominan el mercado y dada la diferencia entre Android (que utilizaba inicialmente para desarrollo el lenguaje de programación Java) y iOS (inicialmente utilizando Objective-C), muchas empresas y desarrolladores optaban por lanzar sus aplicaciones solo en una plataforma, generalmente donde tenían la mayor parte de sus clientes. Entonces casos de apps como Instagram, que fue lanzado primeramente en la plataforma iOS y solo dos años después lanzada en Android, era muy común.

Desde entonces el mercado de desarrollo móvil ha pasado por grandes evoluciones, expansiones e inversiones [8]. Es difícil encontrar una aplicación que no esté disponible en la mayoría de las plataformas y eso no quiere decir que el desarrollo nativo individual para cada plataforma haya acabado. La verdad es que las herramientas para el desarrollo nativo de aplicaciones, o más conocidos como SDK's, puestas a disposición por sus creadores han mejorado mucho, posibilitando escribir una aplicación nativamente sin tener mucho esfuerzo como antes.

A parte del mejoramiento de los SDK's nativos, hubo también un gran aumento de técnicas y herramientas de desarrollo no nativos de multiplataformas que permiten escribir una aplicación una vez y usarla casi que da la misma forma en todas las plataformas, lo que ha posibilitado entonces desarrollos rápidos y concisos.

2.5. Herramientas para desarrollo móvil multiplataforma

Con el crecimiento y la evolución de herramientas que permitían desarrollar una aplicación una sola vez y usar esa misma única aplicación en todas o casi todas las plataformas; muchas técnicas, softwares y *frameworks* surgieron con el propósito de facilitar la vida de los desarrolladores y de las empresas. A continuación, se relacionan algunas.

2.5.1. Xamarin

Xamarin es una plataforma de aplicación de código abierto y gratuita de Microsoft para la construcción de aplicaciones iOS y Android modernos y de alto desempeño usando los lenguajes de programación C# y .NET. [9]

2.5.2. PhoneGap

Desarrollado por el equipo de Apache Cordova, PhoneGap es un *framework* de código abierto que fue discontinuado. Utiliza lenguajes de programación usadas en el desarrollo web (HTML, CSS y Javascript) para desarrollar aplicaciones iOS, Android y otras. [10]

2.5.3. Titanium

Es un ambiente de desarrollo que te permite crear aplicaciones iOS, Android. Mantenido por la empresa AppCelerator, utiliza lenguajes de programación usadas en el desarrollo web como PhoneGap, con una pequeña diferencia que Titanium añade una capa de abstracción más. [11]

2.5.4. React Native

React native es un *framework*, desarrollado por Facebook, para construcción de aplicaciones iOS y Android que utiliza el *framework* React – herramienta creada también por Facebook para creación de interfaces web. React native es gratuito y tiene el código abierto. Actualmente es una de las herramientas más utilizadas para el desarrollo multiplataforma. [12]

2.6. Herramientas multiplataformas.

Antes de empezar a entender el problema por detrás de esas herramientas que permiten el desarrollo rápido para que aplicaciones sean lanzadas en muchas plataformas, es importante entender la clasificación de las herramientas. Podemos clasificar todas esas herramientas citadas anteriormente y todas las otras que no fueran citadas en cuatro categorías: aplicaciones móviles web, aplicaciones web híbridas, aplicaciones interpretadas y aplicaciones generadas por multiplataformas. [13]

2.6.1. Aplicaciones móviles web

Esas aplicaciones son diseñadas para ser ejecutadas en un navegador web (*webview*) y son independientes de la plataforma en la cual se están ejecutando. Es decir, no requieren ninguna adaptación a medida que la plataforma cambia. Por otro lado, los tiempos de respuesta de esas aplicaciones son mayores si se comparan con aplicaciones construidas de forma nativa (usando SDK's proporcionados por las plataformas). Además, esas aplicaciones por razones de seguridad tienen permiso limitado para acceder a recursos nativos de un móvil, como la cámara o sensores.

PhoneGap es un ejemplo de esa categoría. Está diseñado con tecnologías web para ser ejecutado en una *webview* (aplicación que ejecuta en un navegador web), conforme el ejemplo en la ilustración 5.

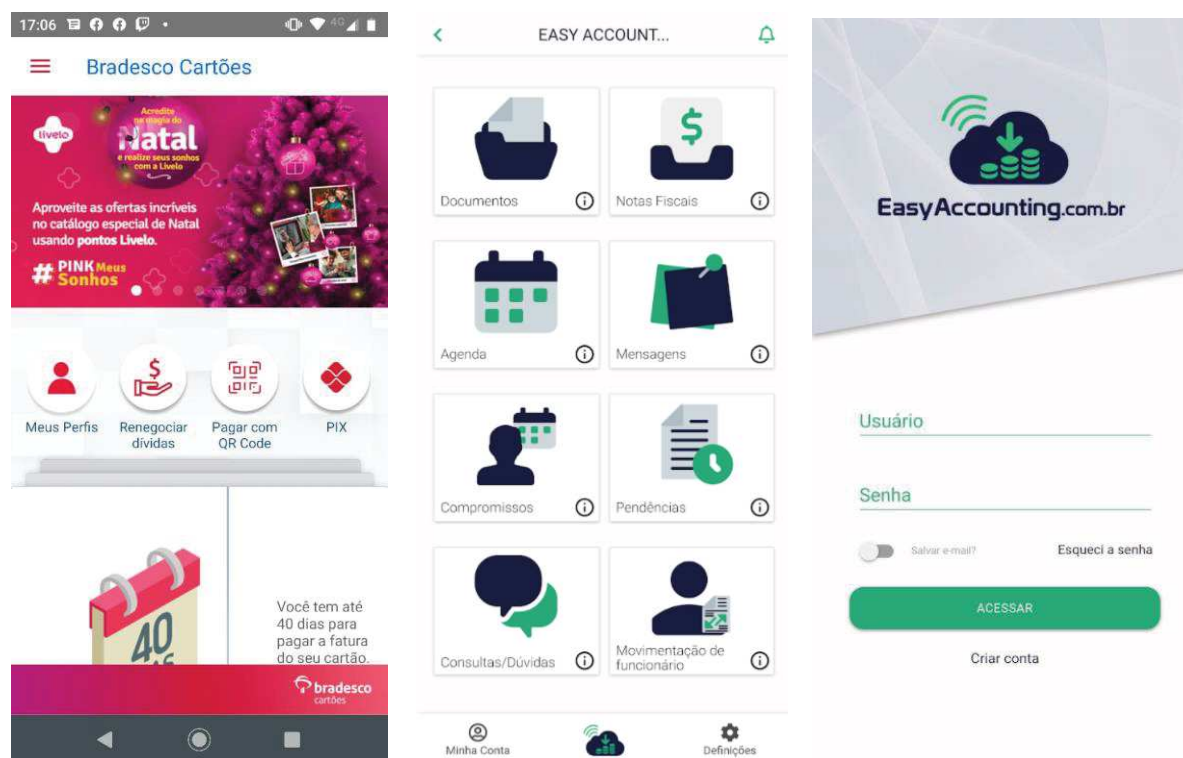


Ilustración 5. Ejemplos de aplicaciones móviles web

Elaboración propia

Por mucho que las apps mostradas en la ilustración 5 parezcan aplicaciones nativas, no lo son. Son aplicaciones brasileñas que son desarrolladas utilizando herramientas de desarrollo web dentro de una *webview*.

2.6.2. Aplicaciones web híbridas

Las aplicaciones web híbridas utilizan el mismo enfoque de la categoría anterior, pero no se ejecutan en una *webview*. En su lugar, se ejecutan en un *webcontainer* del dispositivo, que tiene mayor acceso a recursos nativos del dispositivo por medio de una interfaz de programación de aplicación, también conocida como API.

Las aplicaciones web híbridas ofrecen grandes ventajas porque te permiten reutilizar todo el código para varias plataformas. Pero también ofrecen desventajas como ejecución lenta debido al *webcontainer* y la experiencia del usuario al usar la aplicación es comprometida al no utilizar todos los recursos nativos disponibles.

La ilustración 6 muestra un ejemplo de una aplicación híbrida.



Ilustración 6. Ejemplo de aplicación web híbrida

Elaboración propia

Como se puede observar en la imagen anterior, esta app usa el componente nativo del iPhone, *FaceID*, para autenticar el usuario. Esto permite una mejor integración del usuário con el aplicativo.

2.6.3. Aplicaciones interpretadas

Las aplicaciones interpretadas se generan a partir de un proyecto en que casi todo el código escrito es traducido para el código nativo de la plataforma. En otras palabras, todo el código fuente de la aplicación desarrollada es implantada en el dispositivo, donde es interpretada usando algún tipo de mecanismo (generalmente un SDK de la herramienta elegida).

La herramienta Titanium citada anteriormente encaja en esa categoría, pues cuando la aplicación es instalada en el dispositivo, todo el código es implantando en el dispositivo y es interpretado usando algún tipo de motor Javascript. En el caso de Titanium, Mozilla's Rhino es usado en Android y JavascriptCore en iOS. [11]

La ventaja de este tipo de aplicación es que es independiente de la plataforma, consigue llegar a un nivel muy parecido de una experiencia nativa. Mientras las desventajas son la total dependencia del entorno de desarrollo.

En la ilustración 7 se muestran capturas de pantalla de una aplicación interpretada; en términos de interfaz de la aplicación, es casi imposible notar la diferencia entre ella y una aplicación nativa.

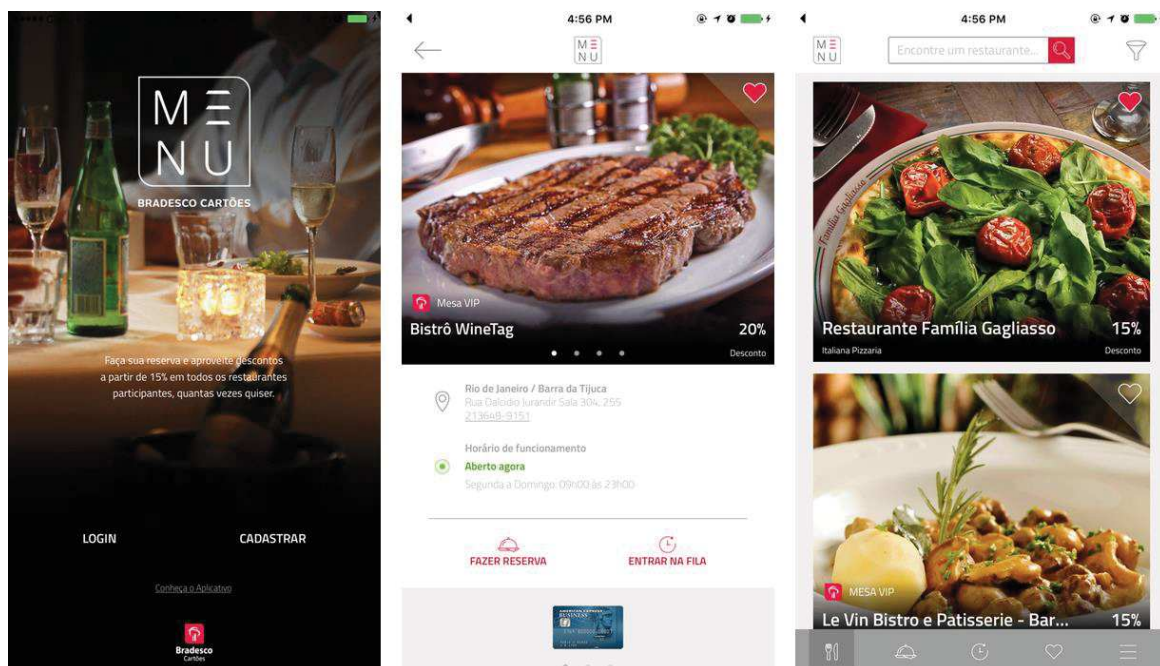


Ilustración 7. Ejemplos de aplicaciones interpretadas

Elaboración propia

2.6.4. Aplicaciones generadas por multiplataformas

Esa categoría es muy parecida con la anterior, la única diferencia es que en lugar del código ser interpretado, el código es compilado. Las aplicaciones son compiladas nativamente, creando una versión específica para cada plataforma de destino. Por ejemplo, antes de lanzar una aplicación, la herramienta hace una especie de “traducción” de la aplicación para la plataforma destino. Observe que no es el desarrollador que tiene ese trabajo, sino que la herramienta elegida. El desarrollador sigue desarrollando la aplicación una sola vez, pero compilando para las plataformas de destino.

La herramienta Xamarin es un ejemplo de esa categoría y te permite escribir el código una sola vez y compilar para las plataformas que esa herramienta soporta. Las ventajas de esa categoría son las mismas de la categoría anterior y las desventajas son cuanto al tamaño de la aplicación que tienden a ser mayores.

En la imagen abajo un ejemplo de una aplicación hecha con Xamarin.

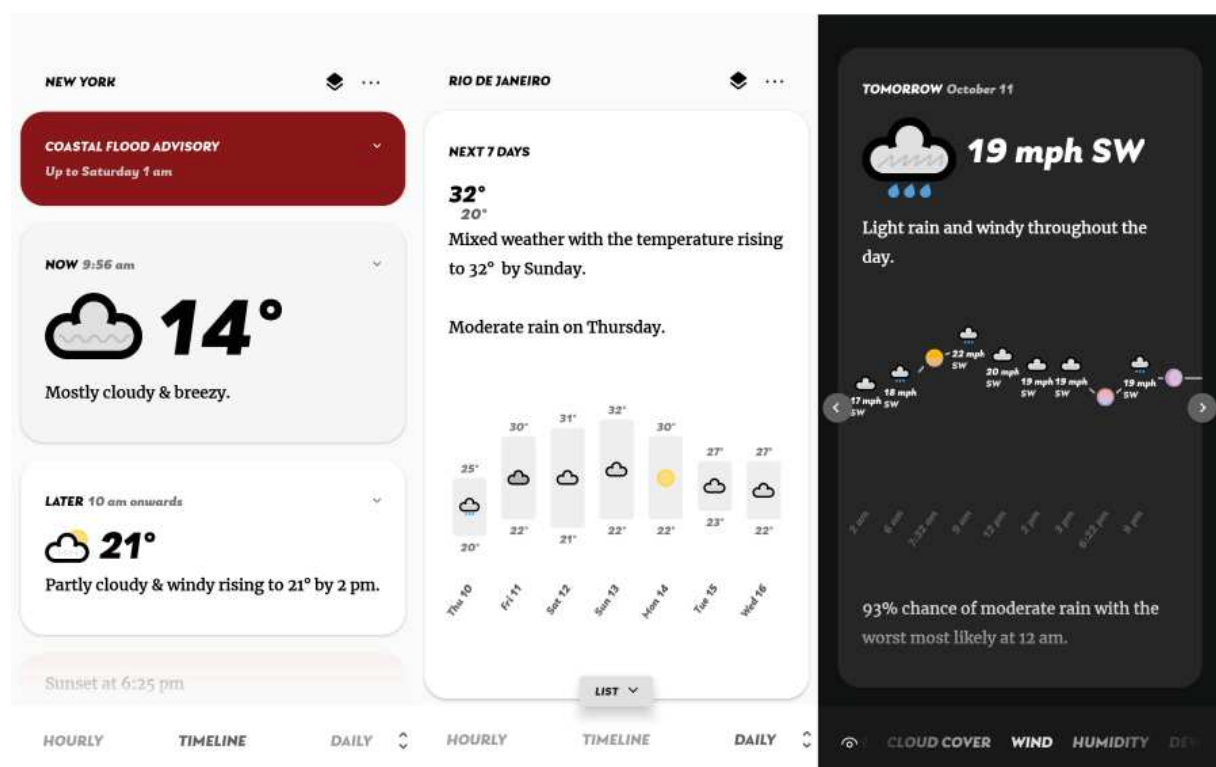


Ilustración 8. Ejemplos de aplicación hecha con Xamarin

Elaboración propia

2.7. El problema de herramientas multiplataformas

Hasta ahora hemos visto que la mayor parte de las desventajas de las herramientas son la pérdida de la experiencia nativa, la lentitud, el tamaño de las aplicaciones y la dependencia de entornos de desarrollo. Eses problemas deben ser llevados en cuenta ya que influyen directamente en el funcionamiento de la aplicación y también en la experiencia del usuario final, lo que puede dar lugar a una mala evaluación de la aplicación y consecuentemente la pérdida de clientes.

En contrapartida, las aplicaciones desarrolladas de forma nativa en gran parte no sufren de esos problemas. No obstante, el desarrollo de aplicaciones nativas exige algunos requisitos, como por ejemplo diferentes equipos para diferentes plataformas (debido a las diferencias entre plataformas, no es común que el mismo equipo trabaje en diferentes plataformas), diferentes bases de código para diferentes plataformas, equipos que deben contar con más personas. Todos estos requisitos influyen directamente en el coste de la construcción de la aplicación y en el tiempo para el desarrollo de esta.

Dentro de esta situación concreta que hemos descrito, nace Flutter - framework multiplataforma donde se desarrolla “una” vez, permitiendo agilidad, eficacia y un modelado de diseño único para cada aplicación, en donde sin importar la plataforma, esta se integrará para brindar la mejor experiencia de usuario posible - con el objetivo de agregar lo mejor de los dos mundos. En otras palabras, acabar o mitigar la mayor parte de los problemas de las herramientas multiplataformas y nativas.

3. FLUTTER

Flutter es una creación de la gran compañía Google. Según la propia compañía Flutter es el conjunto de herramientas de interfaz de usuario (UI) para crear hermosas aplicaciones compiladas de forma nativa para dispositivos móviles, web y de escritorio a partir de una única base de código. [14]

Flutter vio la luz en 2015 con el nombre de *Sky*. Al principio, por tratarse de una herramienta de Google, solo ejecutaba en dispositivos con el sistema operativo Android, pero después fue portado para iOS y hoy en día sigue cubriendo las dos plataformas que dominan este nicho de mercado.

Después de muchas versiones lanzadas desde el primer lanzamiento inicial en 2015, en diciembre de 2018 acontece el primer lanzamiento estable de Flutter, eso significa que la herramienta está lista para que la comunidad (desarrolladores, compañías, entre muchos otros) pueda empezar a utilizarla para crear y poner en marcha aplicaciones reales.

3.1. ¿Para qué sirve Flutter?

Flutter ha nacido con la intención de ser una herramienta que agilice el proceso de desarrollo de software; es decir, que, a través de una base de código, sea posible compilar para multiplataformas, sin tener la necesidad de tener una base de código para cada plataforma.

Pero Flutter surge también con la necesidad de acabar con los problemas comunes a estos tipos de herramientas. Y sabiendo de eso, uno de los principales objetivos de Flutter es ser capaz de renderizar interfaces a una constancia de 120 FPS. Todo esto posibilita una mayor fluidez en las aplicaciones, donde realmente hay una experiencia nativa y la lentitud no es perceptible.

3.2. Principales Partes de Flutter

Flutter está formado por cuatro partes principales. [8]

3.2.1. Flutter Engine

El motor de Flutter está en su mayor parte desarrollado en el lenguaje de programación C++. Su base de código utiliza el mecanismo gráfico Skia, que es una biblioteca gráfica de código abierto también escrita en C++ patrocinada y mantenida por Google, para hacer la renderización. [15]

3.2.2. Foundation Library

Flutter también provee una interfaz sobre los SDK's nativos (de ambas las plataformas – iOS y Android). Eso quiere decir que no es necesario preocuparse en cómo inicializar el aplicativo de cámara en iOS (programáticamente hablando) y cómo hacer eso también en Android, por ejemplo. Todo lo que tiene que saber es como llamar el aplicativo de cámara usando Flutter y entonces Flutter se encarga de hacer funcionar en la plataforma en la cual la aplicación se está ejecutando.

3.2.3. Dart

A diferencia de Android, Google ha elegido para Flutter el uso del lenguaje de desarrollo Dart; que también fue creada por Google en 2011 y tuvo su primer lanzamiento estable en noviembre de 2013, cerca de dos años antes del lanzamiento de Flutter.

Antes de Flutter, pocas personas conocían dicho lenguaje, pero debido a Flutter, Dart está evolucionando muy rápido desde su lanzamiento. La siguiente imagen muestra un gráfico sobre los lenguajes de programación que fueron más relevantes para los desarrolladores en 2019. [16]

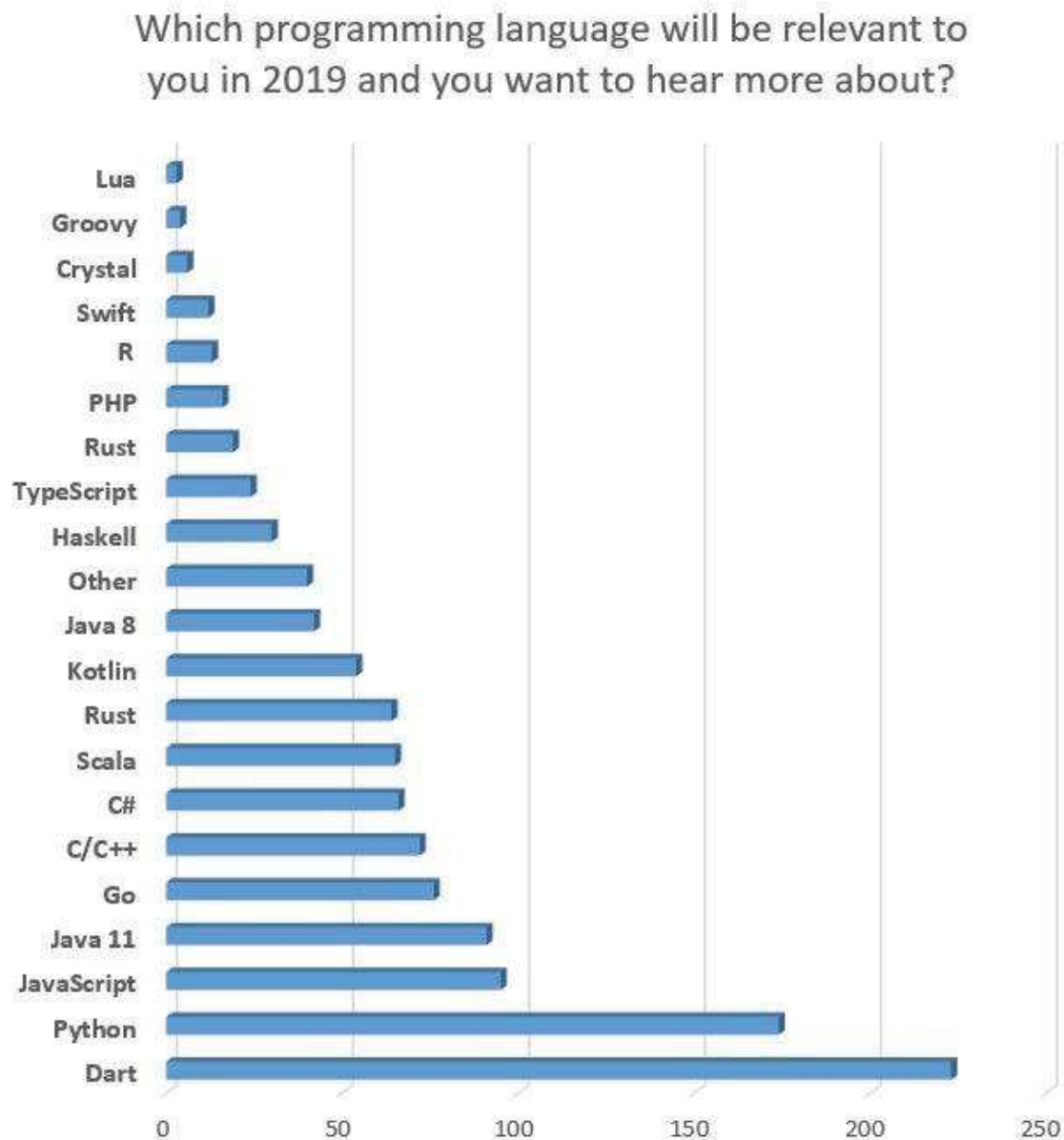


Ilustración 9. Lenguajes de programación más relevantes en 2019 para desarrolladores

Tomada de [16]

Es importante tener en cuenta que Dart no es solo un lenguaje de desarrollo móvil, sino también para desarrollo IoT (Internet de las Cosas) y aplicaciones web, entre otras.

3.2.4. Widgets

Por ultimo y quizás como aspecto más importante, hemos de hablar del concepto de **widgets**. En Flutter, todo es tratado como un widget y no es una exageración hablar así, porque en Flutter realmente todo es un widget.

Un widget es un objeto de alto nivel que se utiliza para describir cualquier parte de la aplicación. Por ejemplo, pueden ser (pero no se limita a) elementos de la interfaz de usuario como botones; datos (un tema, configuraciones); diseño (alineación, relleno). [17]

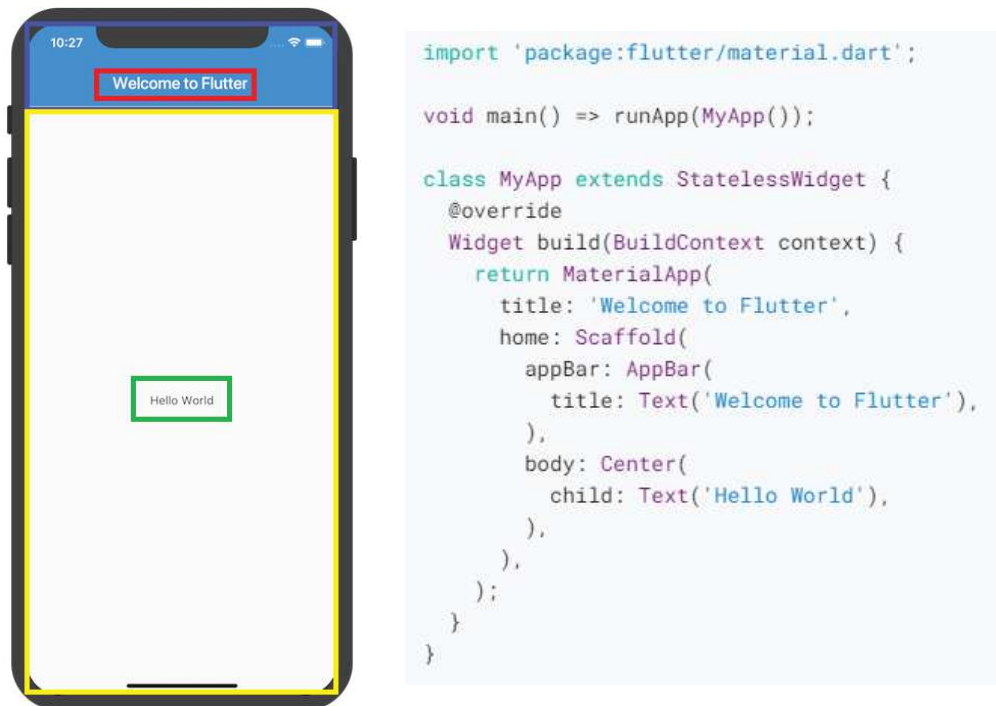


Ilustración 10. Representación de widgets

Elaboración propia

En la ilustración 10, en el móvil a la izquierda cada rectángulo formado por un color es un widget diferente. Al lado derecho, el código que representa cada widget destacado en la izquierda.

En Flutter, algunos widgets pueden tener hijos; algunos pueden tener un solo hijo, mientras otros pueden tener más de uno y esos hijos pueden tener otros hijos y así sucesivamente; el resultado es una enorme jerarquía de widgets, también conocido como árbol de widgets.

3.3. Ventajas

Por ser una herramienta nueva, hasta el momento en que se escribe este documento, Flutter tiene en sus componentes muchas ventajas que son tenidas en cuenta por desarrolladores de software. Algunas de ellas, según Zammetti (2019) [8], son:

3.3.1. Hot Reload

Esa ventaja te permite, según Google [18], experimentar y crear interfaces de usuario, agregar funciones y corregir errores de forma rápida y sencilla; esa funcionalidad inyecta archivos de códigos recién actualizados en la máquina virtual de DART en ejecución. Así Flutter reconstruye automáticamente el árbol de widgets sin la necesidad de reiniciar la aplicación a cada modificación.

3.3.2. Multiplataforma de verdad

Según Google, sus aplicaciones Flutter se ejecutarán correctamente en ambas las plataformas, con el mínimo de esfuerzo. Gracias a Flutter es posible proveer dos grupos de widgets, *Material Design* y *Cupertino* para Android y iOS respectivamente; de esa forma, el desarrollador puede crear aplicaciones con experiencia nativa para cada sistema plataforma distinta.

3.3.3. Dart

Debido a que Dart no tiene una curva de aprendizaje tan larga como Java, muchos desarrolladores y no desarrolladores pueden empezar con Flutter sin grandes esfuerzos, dado que Dart es similar a lenguajes como Javascript, Java y Objective-C.

Es importante decir también que en la propia documentación de Google sobre el lenguaje Dart, hay un apartado completo de introducción de Dart para desarrolladores Java y también muchas otras herramientas cuyo objetivo es el facilitar el proceso de aprendizaje. [19]

3.3.4. Widgets

Además de que Flutter permite crear widgets personalizados, también tiene un rico conjunto de widgets ya creados disponibles, tanto para iOS como para Android, lo que hace que esa sea una de sus principales ventajas. Sin contar que ofrece muchos más widgets, sin la necesidad de API de terceros, como sí ocurre con React native, por ejemplo.

3.3.5. Herramientas

Flutter permite empezar con el entorno desarrollo de forma muy rápida y fácil, porque en Flutter no hay mucha dependencia de dicho entorno de desarrollo.

Por lo tanto, si un desarrollador quiere empezar con un ambiente más avanzado con la utilización de otras herramientas, Flutter te permite usar muchas de esas herramientas.

3.4. Desventajas

Así como cualquier herramienta, Flutter tiene sus puntos negativos que necesitan ser evaluados por quién desea utilizar ese *framework* y así juzgar si Flutter es realmente la mejor opción.

3.4.1. Herramienta nueva

Por tratarse de una herramienta nueva, hay muchas dudas cuanto a su credibilidad. Por más que pertenezca a una gran compañía como Google, lo que le aporta un cierto nivel de seguridad; eso no es suficiente.

Herramientas nuevas traen por sí solas muchas dudas e inseguridad, no solo en lo relativo a la empresa que las ha creado, sino también por tener muchos errores iniciales, o no tener disponible una gran comunidad. Y en Flutter no es diferente, por más que el interés en Flutter ha crecido en los últimos años desde su lanzamiento, como puede ser observado en la siguiente imagen, que fue obtenida de una búsqueda en Google Trends. [20]

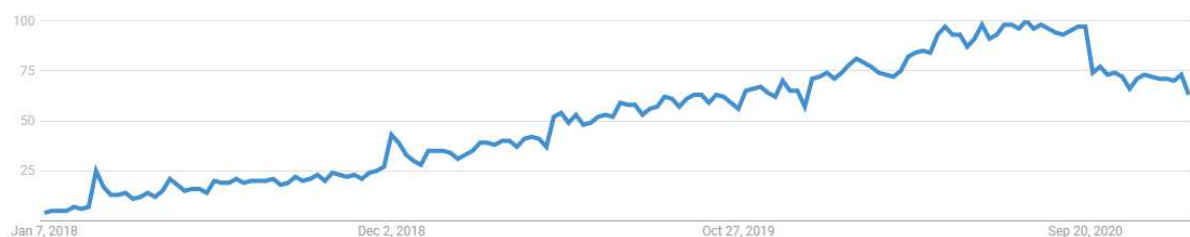


Ilustración 11. Crecimiento de la búsqueda de Flutter en Google (2018 - 2020)

Tomada de [20]

En la web oficial de Flutter⁴, hay una información que afirma que Flutter ofrece “performance nativa y una interfaz de usuario expresiva y flexible”. No obstante, la comunidad de desarrolladores no está totalmente de acuerdo con esa afirmación, ya que en las *issues* del repositorio de Flutter en github⁵, hay más de 8 mil cuestiones

⁴ <https://flutter.dev/>

⁵ <https://github.com/flutter/flutter/issues>

abiertas, y algunas de ellas retratan exactamente lo contrario de lo que dice el sitio oficial de Flutter. Al parecer, en el momento de la redacción de esa memoria de TFM, Flutter se estaría ejecutando sensiblemente mejor en la plataforma Android. Adicionalmente, según algunas *issues* y muchos usuarios en la comunidad Flutter en Reddit⁶, Flutter ha dejado de ser seguro para aplicaciones en producción, debido a un problema que Flutter viene enfrentando con animaciones en la plataforma iOS, también conocido como *iOS jank animations*. Según la comunidad el problema fue descubierto en agosto de 2020 y hasta el momento no hay solución oficial por parte del equipo de Flutter de Google.

Lo que parece entonces es que Flutter no cuenta con un equipo de trabajo con muchas personas, ya que hay más de 8 mil cuestiones abiertas en su repositorio oficial y aún quedan muchos errores por solucionar⁷.

3.4.2. Mezcla de código

A diferencia de otros ambientes de desarrollo como, por ejemplo, desarrollo web, en Flutter hay mezcla de código de lógica e interfaz. Para ejemplificar mejor esa desventaja, en desarrollo web, por ejemplo, en la mayoría de los casos la interfaz está separada de la parte lógica. HTML, CSS y JS se usan para la interfaz y algún otro lenguaje de desarrollo para la parte lógica (PHP, JS, Java, entre otras).

En Flutter todo se hace con Dart. Entonces es común tener el código de lógica junto con el código de interfaz. Claro que para muchos desarrolladores eso no llega a ser una desventaja, es una cuestión de costumbre con la herramienta.

La ilustración 12 muestra un ejemplo de código en Flutter, donde hay una mezcla de código de la parte lógica (rectángulo rojo) y código de interfaz (rectángulo verde).
[21]

⁶ <https://www.reddit.com/r/FlutterDev/>

⁷ Sobre la resolución de problemas en Flutter, ver apartado 3.12. Flutter 2.0

```
class MyButton extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return GestureDetector(  
      onTap: () {  
        print('MyButton was tapped!');  
      },  
      child: Container(  
        height: 36.0,  
        padding: const EdgeInsets.all(8.0),  
        margin: const EdgeInsets.symmetric(horizontal: 8.0),  
        decoration: BoxDecoration(  
          borderRadius: BorderRadius.circular(5.0),  
          color: Colors.lightGreen[500],  
        ),  
        child: Center(  
          child: Text('Engage'),  
        ),  
      ),  
    );  
  }  
}
```

Ilustración 12. Fragmento de ejemplo de la mezcla de código en Flutter

Tomada de [21]

Oficialmente, Google no ha lanzado ninguna forma o actualización que solucione dicho “problema”, hasta el momento. Pero la comunidad ha creado sus propias soluciones; la más común es la creación de funciones para las partes lógicas en archivos distintos y entonces solo hacen la llamada de esas funciones.

3.4.3. Google

Hay personas que consideran el hecho de que Flutter pertenezca a Google una ventaja y hay sentido en eso. Pero para Zammetti (2019) [8] eso hace todo menos confortable, ya que Google controla muchas cosas en la internet hoy en día. Como se ha mencionado anteriormente, Flutter, Dart y hasta Skia fueron creadas por Google o son mantenidas por él. Entonces con todo ese control Google puede dictar normas y discontinuar la herramienta cuando quiera, como ya ha pasado antes con otras tecnologías.

3.4.4. Árbol de Widgets

Los widgets en si son una maravilla, pero cuando se trata de la unión de ellos, la jerarquía que generan entre sí tiende a ser un terror para muchos desarrolladores.

La profundidad que puede llegar una jerarquía de árbol de widgets en Flutter, muchas veces es mucho mayor que en HTML, por ejemplo, lo que provoca cierto temor a la hora de hacer desarrollo en Flutter. No obstante, la comunidad de Flutter ya tiene soluciones para estos problemas, como herramientas para identificación y comentarios en el código para que el desarrollador no se pierda dentro de la jerarquía de widgets.

3.4.5. Tamaño de las aplicaciones

Las aplicaciones desarrolladas con Flutter tienden a ser un poco mayores que las aplicaciones desarrolladas con SDK's nativos. Eso se debe a que las aplicaciones necesitan incluir en su estructura el motor de Flutter (*Flutter Engine*), el framework de Flutter, librerías de soporte y algunos recursos más.

3.4.6. Programación reactiva y gestión de estado

Flutter sigue el paradigma de programación reactiva, así como React native. Pero a diferencia de React Native, en Flutter cada widget tiene un requisito casi obligatorio, que es el método *build*.

El método *build* retorna una representación visual que incorpora el estado actual del widget. Veamos un ejemplo: partamos de la siguiente imagen (ver ilustración 13), donde tenemos un texto en el centro de la página que cuenta el número de veces que el botón fue presionado. El texto no es más que un widget que tiene como estado actual el número cero indicando que el botón aún no fue presionado. Pero a medida que el botón es presionado, este widget de texto empieza a reaccionar al clic en el botón. El método *build* entonces es llamado para reconstruir nuevamente el widget de texto para que muestre el nuevo valor. [22]

Ese paradigma puede ser muy atractivo tanto para Flutter como para otras herramientas como React native, pero algunas veces puede convertir cosas fáciles en más difíciles de lo que deberían ser. Adicionalmente al tratamiento de los eventos, ha de considerarse el tema de la gestión del estado para la cual no existe una forma concreta, sino que existen diversas formas de manipulación del estado de la aplicación y de los widgets y cada forma tiene sus ventajas y desventajas.

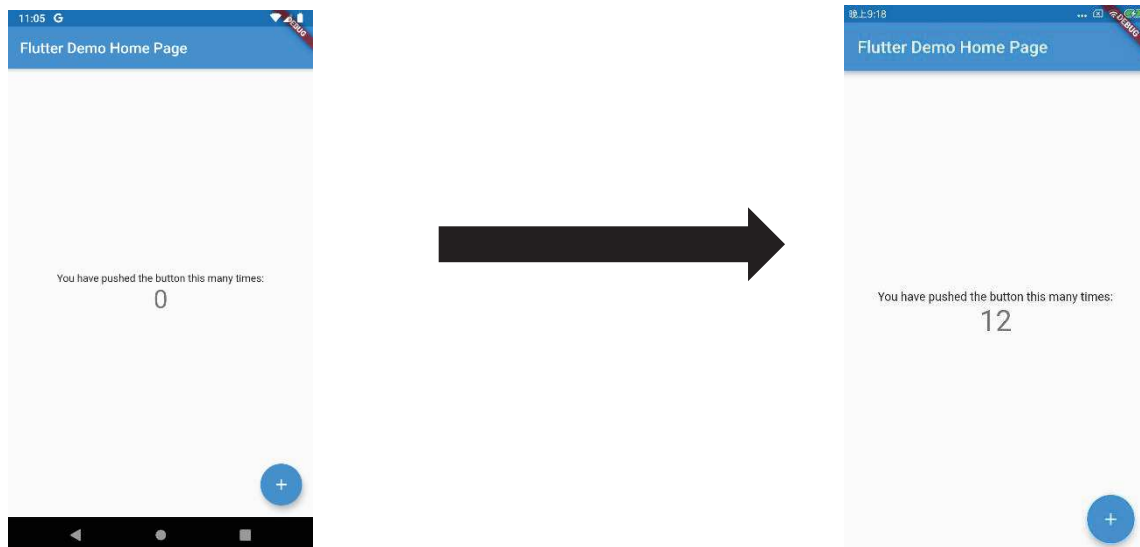


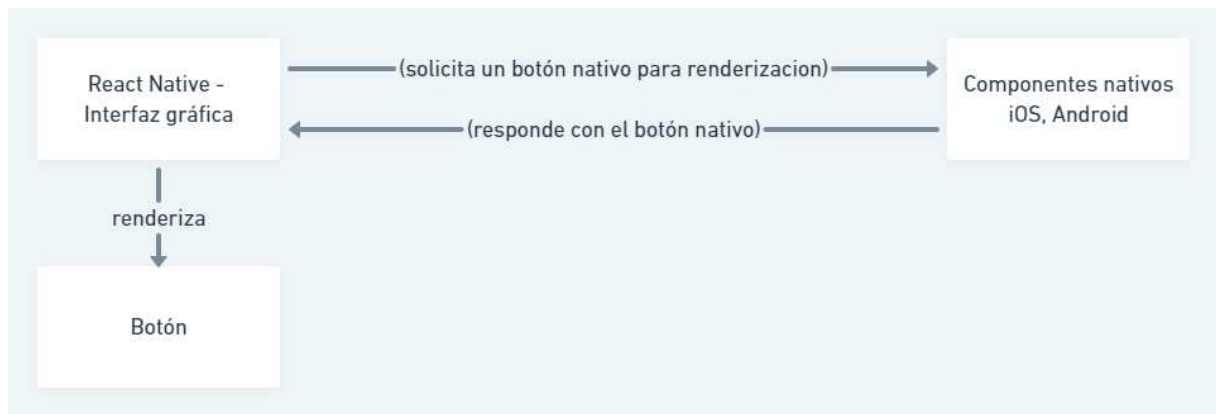
Ilustración 13. Demostración del paradigma reactivo de programación y el gerenciamento de estados

Elaboración propia

3.5. ¿Porque utilizar Flutter?

Flutter, en comparación con la mayor parte de las otras herramientas disponibles, renderiza sus propios componentes de interfaz y no usa componentes nativos, como la mayoría de las herramientas hace. Por ejemplo, cuando un desarrollador escribe un código para dibujar un botón en la pantalla, Flutter no hace una llamada al sistema operativo para renderizar el botón, sino que Flutter renderiza dicho botón por sí solo.

La siguiente imagen muestra el modo en que React native hace la renderización de un componente de botón; debido a esta forma de trabajar (no empleada solamente por React Native), a medida que la aplicación crece, acaba volviéndose más lenta. En este sentido, Flutter tiene una gran ventaja sobre esas herramientas.

**Ilustración 14. Modo de construcción de widget en React Native**

Elaboración propia

Debido a su propia forma de renderizar componentes, Flutter puede proyectar sus propios widgets. En otras palabras, Flutter no está vinculado al sistema operativo en lo concerniente a los widgets. Eso permite a Flutter ofrecer dos kits de conjuntos de widgets: *Material Design* widgets (Android) y *Cupertino* widgets (iOS), lo que brinda una experiencia nativa.

Es importante decir también que Flutter no impone restricciones en cuanto a renderización de widgets. En otras palabras, si el desarrollador quiere renderizar un widget que es común en Android, en iOS; lo puede hacer. Igualmente, puede renderizar un widget común en iOS en un dispositivo Android. Claro que influye en una cuestión de experiencia nativa de ambos sistemas operativos, ya que no es común para una persona que tiene un teléfono con Android tener widgets similares al de iOS.

**Ilustración 15. Diferencia de widget en Android y iOS**

Elaboración propia

La ilustración 15 es un ejemplo de la diferencia de un widget entre los dos sistemas operativos.

3.6. ¿Quién usa Flutter y por qué?

Con la rápida ascensión de Flutter en los últimos años desde su lanzamiento y debido a sus puntos positivos en comparación con otras tecnologías existentes del mismo ramo, Flutter ha atraído la atención de grandes empresas, visto que es mucho mejor tener una sola base de código y la compilación de esta para multiplataformas, que mantener varias bases de códigos, una para cada plataforma distinta.

Multiplataformas no quiere decir solamente iOS y Android, sino también aplicaciones web y aplicaciones de escritorio. Aunque Flutter ha evolucionado mucho más en relación al mundo móvil, sigue teniendo muchos progresos en las otras ramas. Así que, en un futuro próximo, dependiendo de cómo esta tecnología avance, podremos tener una sola base de código y aprovechar esa misma base para hacer aplicaciones para todas las plataformas, con pequeñas modificaciones de una para otra.

Teniendo en cuenta el coste, es mucho más ventajoso para las compañías mantener una sola base de código; por ejemplo, para una empresa mantener una base de código para cada plataforma implica muy probablemente tener más de uno equipo de desarrollo, ya que las tecnologías que cada plataforma utiliza, en general, son distintas.

Sabiendo de eso y con todas ventajas que Flutter ha proporcionado desde su lanzamiento, empresas como BMW, NuBank, Ebay, Tencent, Google, Alibaba Group, New York Times entre otras muchas están interesadas en utilizar Flutter. Por otro lado, aplicaciones como Google Ads, Insight Timer, Stadia, Baidu, NuBank, Philips Hue, Ebay ya están hechos con Flutter. [23]

3.7. ¿Cuánto cuesta Flutter?

Hasta el momento, Flutter sigue siendo una herramienta de código abierto lo que significa que cualquier persona puede usarla y/o modificarla sin tener que pagar nada. Como se ha dicho anteriormente y siguiendo el ritmo del mercado de nuevas tecnologías, su creadora Google tiene la costumbre de ofrecer muchas tecnologías desarrolladas por ella sin la necesidad de pagar, hasta cierto punto.

Es ventajoso para Google que Flutter siga siendo gratis, pues así la comunidad tiende a crecer mucho y evolucionar cada vez más esta tecnología, convirtiéndola así la mayor compañía en este mercado también.

3.8. ¿Como aprender Flutter?

Al igual que con la mayoría de las nuevas tecnologías que surgen, Google no lo ha hecho de forma diferente con Flutter. En el sitio web oficial de Flutter (<https://flutter.dev/>), Google ofrece una extensa documentación sobre la herramienta. Es importante recordar que aprender Flutter implica también en aprender el lenguaje Dart, ya que es el lenguaje usado por Google en Flutter. Dart también ofrece un sitio web oficial (<https://dart.dev/>) con toda la documentación necesaria para aprender este lenguaje.

Empezando con la instalación, la documentación de Flutter cubre gran parte, sino todo, lo que hay en Flutter. En el mismo apartado, hay una parte dedicada exclusivamente a enseñar cómo configurar el ambiente de desarrollo y un tutorial de cómo escribir su primera aplicación en Flutter.

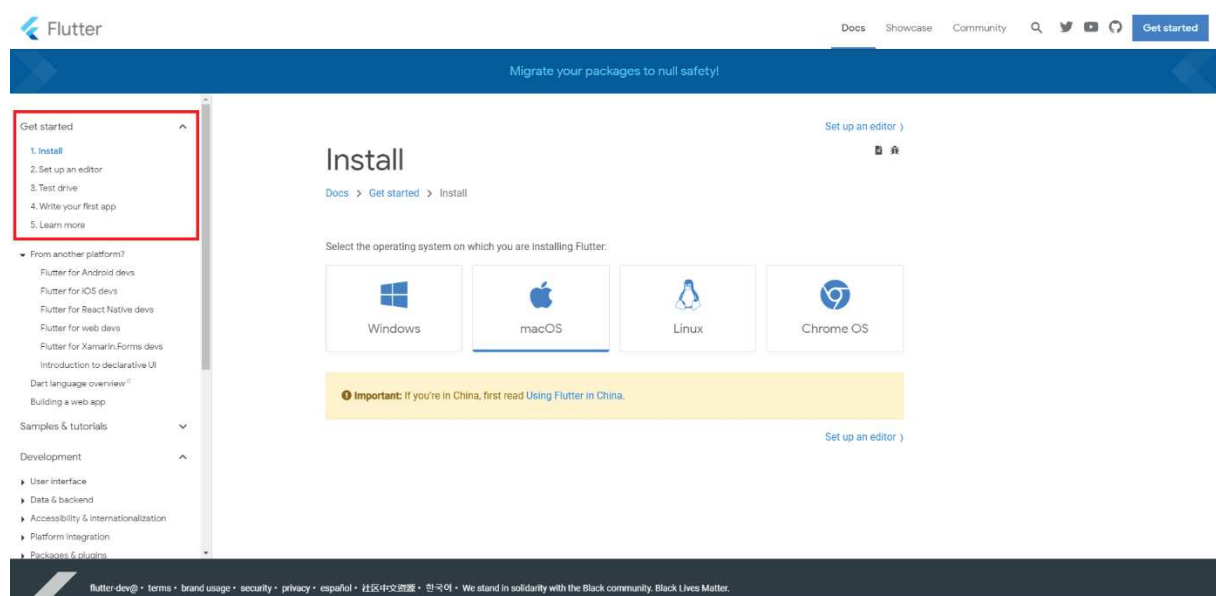


Ilustración 16. Página oficial de instalación de Flutter

Tomada de [24]

En la ilustración 16, marcado con el rectángulo rojo, se observan todas las etapas iniciales para quién desea empezar con Flutter.

Pero claro, si ya tiene experiencia con desarrollo de software con alguna otra tecnología, las cosas deben ser más fáciles. Se puede encontrar en el mismo sitio web un apartado solo para desarrolladores que ya tienen experiencia con otras tecnologías. Este apartado ayuda a desarrolladores a entender Flutter partiendo de tecnologías como Android (nativo), iOS (nativo), React native, tecnologías web o Xamarin. En otras palabras, si conoces el funcionamiento de alguna de esas tecnologías, la tendencia es que el entendimiento de Flutter sea más fácil.

En el mismo apartado hay una visión general sobre el lenguaje Dart y también sobre cómo utilizar Flutter y Dart para la construcción de una aplicación web.

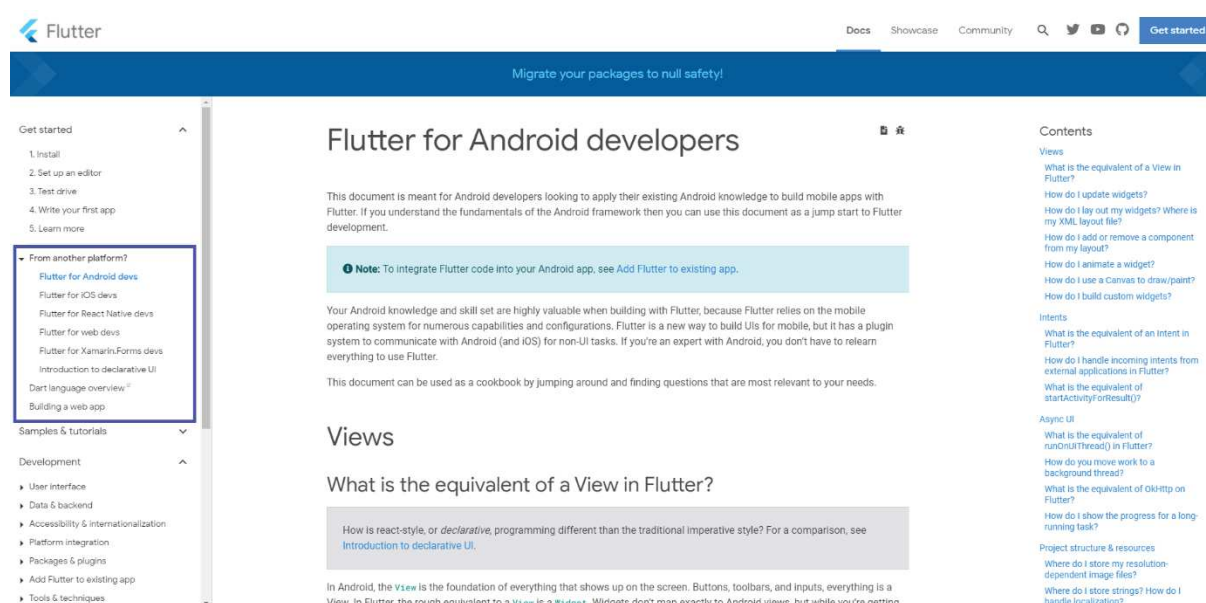


Ilustración 17. Documentación de Flutter para desarrolladores de otras plataformas

Tomada de [25]

A continuación, hay otro apartado en que se recogen muchas aplicaciones hechas con Flutter para que el interesado pueda observar cómo es una aplicación en tiempo real. Además, también hay tutoriales y enlaces para códigos de terceros.

Más adelante, ya se puede notar una parte más descriptiva y avanzada. En otras palabras, cada tecnología o asunto necesario para desarrollo en Flutter está expuesto detallado e individualmente. Por ejemplo, si quieres saber cómo es el funcionamiento de datos en una aplicación Flutter, ahí hay apartados para ese tema y muchos otros.

La documentación de Flutter, engloba también la parte de pruebas en aplicaciones, cómo depurar y también cómo desplegar su aplicación en las tiendas de aplicaciones oficiales, Google Play (Android) y App Store (iOS).

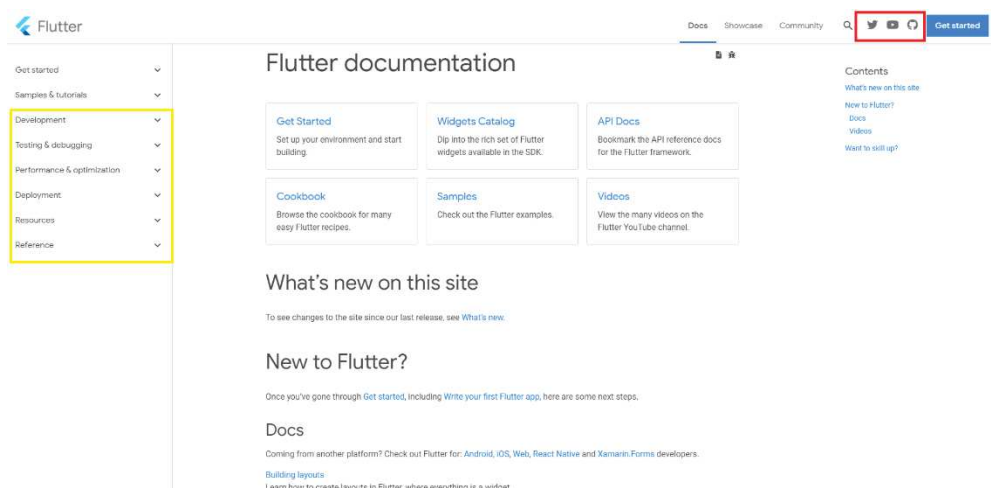


Ilustración 18. Contenidos más avanzados y detallados de Flutter y sus canales de comunicación oficial

Tomada de [26]

La forma de aprender es una elección personal, porque hay muchas formas de aprender Flutter. Actualmente ya hay muchos cursos que se ofrecen en muchas plataformas; en YouTube ya es posible encontrar muchos tutoriales sobre la herramienta; además de los canales oficiales de la propia herramienta, en el cual hay disponibilidad de muchas más informaciones oficiales.

3.9. ¿Hay algún tipo de certificación para desarrolladores?

Todos desarrolladores tienen algún tipo de habilidad en lo cual se destacan. Sin embargo, todo desarrollador debe tener una forma de probar que realmente es un experto en determinada área. Las certificaciones existen justamente para hacer este papel: probar que determinada persona es experta en alguna área.

Las Certificaciones son credenciales que validan, reconocen el conocimiento y/o experiencia de una persona. Generalmente, las certificaciones son emitidas o generadas por empresas de gran prestigio en el mercado, por empresas reguladoras o por la propia empresa que ha creado tal tecnología [27]. Por ejemplo, Google es una empresa que ofrece muchas certificaciones, ya que es la creadora de muchos productos y mantenedora de muchos otros, sin contar que es una gran compañía y muy reconocida en el mercado de tecnología. En el sitio web de certificaciones para desarrolladores, se puede acceder a todos los tipos de certificaciones ofrecidos por Google. [28]

Oficialmente Flutter, hasta el momento de la creación de este documento, no tiene ningún tipo de certificación. Es decir, su creadora Google aún no ha puesto certificaciones disponibles para la comunidad sobre esa herramienta [28]. Aunque no haya certificaciones por parte de Google, hay otras empresas que están ofertando esa certificación, como es el caso de la empresa ATC.

ATC ofrece además de la certificación en Flutter, muchas otras certificaciones como por ejemplo en el desarrollo Android. Es importante recordar que esa es una empresa independiente y que no están afiliados a Google, así que su modelo de certificaciones no ha sido verificado por Google que es la creadora de esas tecnologías. [29]

3.10. ¿Puede interactuar con otras tecnologías?

Flutter, así como otras tecnologías, tiene buena interacción con otras tecnologías. De hecho, la verdad es que cualquier tecnología que no interactúa con otras tecnologías tienden a ser menos elegidas por la comunidad, ya que hoy en día con todo ese avance tecnológico, es difícil que una tecnología se mantenga sola, sin la necesidad de interacciones.

Debido a la gran creciente comunidad de Flutter, esa herramienta ya cuenta con muchos paquetes desarrollados y compartidos por desarrolladores del ecosistema Flutter y Dart. Los paquetes no son nada más que códigos de otros desarrolladores, empresas que ya son compatibles con Flutter, que te permiten interactuar y, conectar con otras tecnologías de manera mucho más fácil.

Google también pone a la disposición de la comunidad un sitio web con todos los paquetes disponibles, tanto para Dart como para Flutter. En la ilustración 19 se puede observar, por ejemplo, que hay muchos paquetes ya listos para conectar Flutter con PostgreSQL, que es una base de datos relacional. Destacado con el rectángulo amarillo está el campo de busca, donde puede buscar el nombre del paquete que tiene interés; en naranja esta la tecnología que está buscando el paquete, siendo posible elegir entre Dart, Flutter y *Any* (cualquiera). Más abajo, en rojo es, un ejemplo del retorno de la búsqueda que devuelve un listado de todo lo que fue encontrado en base a la búsqueda. Cada elemento de la lista muestra información como nombre del

paquete, descripción, versión, fecha de la última actualización, persona o compañía creadora, número de “me gusta”, puntuación y popularidad del paquete [30].

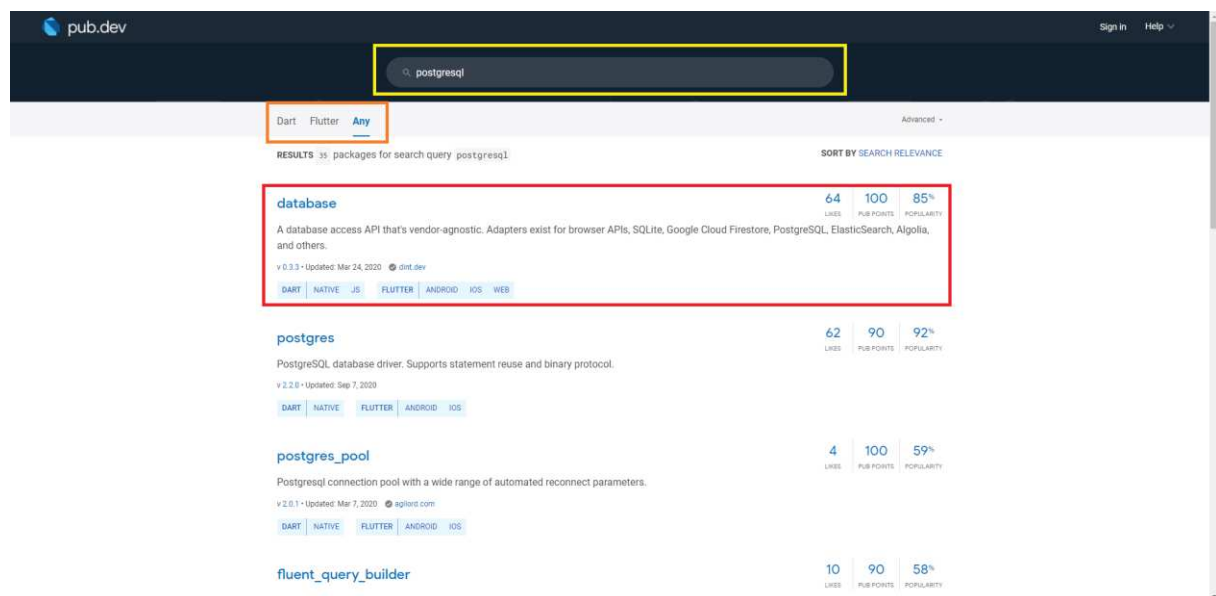


Ilustración 19. Flutter paquetes

Tomada de [30]

Además de los paquetes, Flutter también cuenta con muchos *plugins* que te permiten hacer lo mismo que los paquetes, solo que de forma más directa e integrada al ambiente de desarrollo.

3.11. Profundizando en Flutter

Este apartado cubrirá partes más avanzadas de Flutter. Como por ejemplo nuevas funcionalidades que llegaron en las nuevas versiones, curiosidades, explicaciones más detalladas sobre los principales widgets usados en el desarrollo de la aplicación que se describe en el próximo capítulo y Dart.

3.11.1. Añadiendo Flutter a una aplicación ya existente

Para muchas empresas y hasta para desarrolladores, que ya tienen aplicaciones hechas, no es ventajoso reescribir la aplicación desde cero en Flutter de una sola vez. Teniendo eso en cuenta y todas las ventajas que Flutter ofrece, Google ha pensado en eso y ha lanzado una funcionalidad que permite que Flutter sea integrado, poco a poco a una aplicación ya existente, a través de módulos o librerías.

Oficialmente, esta funcionalidad llamada *add-to-app* y disponible a partir de la versión v1.12 de Flutter, solo soporta aplicaciones hechas en Android y iOS; es decir, si tienes una aplicación hecha con React native y deseas reescribir la misma de forma gradual con Flutter, no es posible.

Según Google, teniendo en cuenta un conocimiento previo en Flutter, es posible utilizar Flutter en aplicaciones hechas para Android o iOS en pocos pasos, pero siempre con algunas limitaciones: [31]

- Ejecutar múltiples instancias de Flutter o usarlo para renderización parcial de pantallas pueden tener comportamiento inesperado;
- Usar Flutter como *background* aún no está disponible;
- Usar múltiples librerías Flutter en una aplicación no es compatible;
- Algunos *plugins* de Flutter pueden tener comportamiento inesperado;
- A partir de la versión v1.17, esa funcionalidad solo soporta aplicaciones Android desarrolladas con AndroidX (versión del SDK del Android).

Las siguientes imágenes muestran de manera rápida cómo añadir Flutter a aplicaciones (Android y iOS respectivamente) y como es el comportamiento de estas.

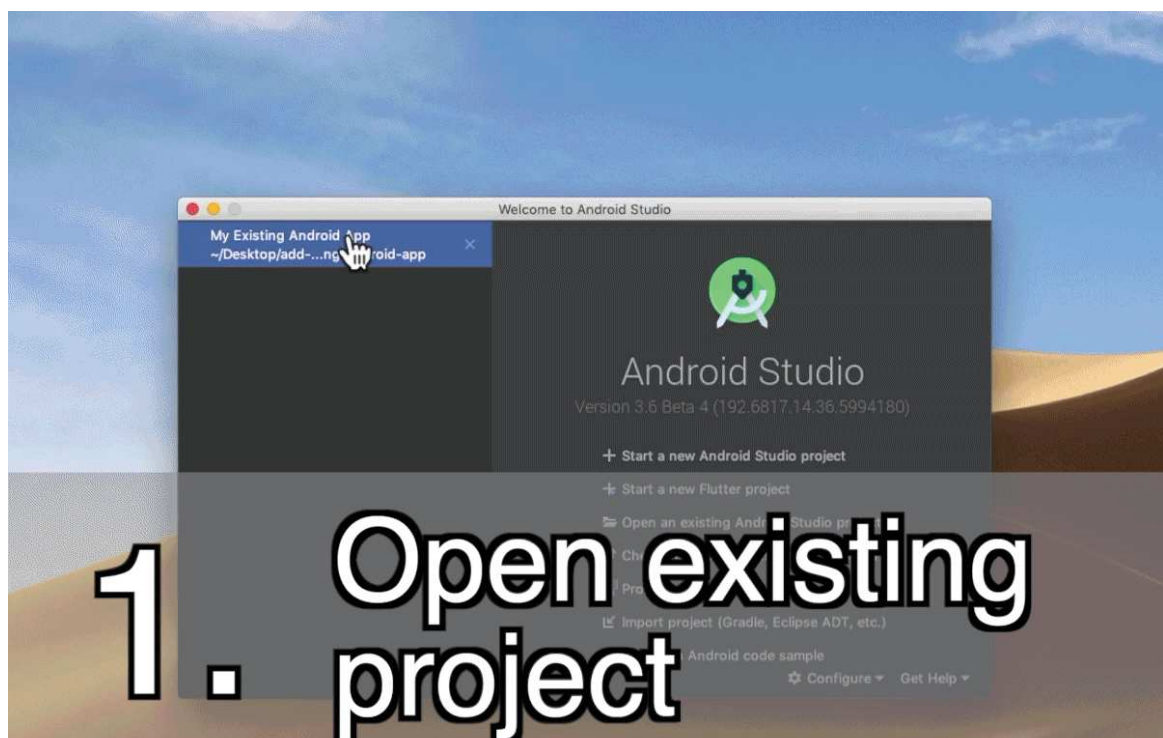


Ilustración 20. Añadiendo Flutter a una aplicación Android

Tomada de [31]



Ilustración 21. Añadiendo Flutter a una aplicación iOS

Tomada de [31]

La empresa Nubank, mayor banco digital independiente del mundo, que es una *fintech* brasileña que cuenta con más de 20 millones de clientes en todo Brasil tiene un caso de ejemplo del uso parcial de Flutter [32]. Según la propia compañía la opción por Flutter lleva en cuenta las ventajas que esa herramienta proporciona y que la aplicación que muchos brasileños utilizan cuenta con partes desarrolladas nativamente, React native y Flutter. De esa forma, las nuevas funcionalidades serán desarrolladas en Flutter, por lo que a medida que la aplicación crezca, se espera que Flutter se convierta en la mayor parte de la base de código. [33]

También, según la compañía, las aplicaciones siguen funcionando sin problemas, usando tanto partes en Flutter como partes con código nativo, debido a esa funcionalidad de Google, siendo además transparente para los usuarios. De hecho, estos últimos tienen la sensación de estar utilizando una aplicación hecha completamente de forma nativa. [33]

3.11.2. *Null Safety*

Flutter a partir de la versión v1.24 ha introducido una nueva funcionalidad llamada *Null Safety*. Es una funcionalidad que tiene el objetivo de ayudar a que tanto desarrolladores expertos como desarrolladores noveles eviten errores mientras desarrollan una aplicación. [34]

Para quién ya tiene algún conocimiento en lenguajes de programación como Kotlin, Typescript o C#, esa funcionalidad no debe ser ninguna novedad, ya que en estos lenguajes es una funcionalidad común. Sin embargo, para quien ya ha tenido una experiencia con algún error como *NullPointerException* (muy común en el lenguaje Java) mientras desarrolla o ejecuta alguna aplicación, esta funcionalidad puede ser muy útil.

Null Safety tiene el objetivo de crear dos tipos distintos: los que aceptan valores NULL y los que no aceptan valores NULL; con ello se puede representar de dos formas distintas un valor que antes era representado solo por un tipo. En otras palabras, en lenguajes de programación como Java, por ejemplo, las cadenas de texto son atribuidas al tipo *String* sin importar si la cadena de texto es nula o no. En Flutter y Dart siguiendo el mismo ejemplo, con esa funcionalidad integrada, hay dos tipos: *String?* y *String*, la primera opción acepta valores nulos mientras la segunda no. Entonces lo que tenemos es una aceptación explícita de valores nulos (así como en Java implícitamente), pero con algunos principios:

- Por defecto, el tipo siempre es no nulo: a menos que esté explícito en la declaración del tipo, de la variable, que ella acepta el valor nulo. En la ilustración 22 hay un ejemplo de los tipos, donde la variable *nombre* del tipo *String*, por defecto no acepta valores nulos, mientras la variable *apellidos* explícitamente acepta valores nulos.

A screenshot of code on a dark background. The first line is `String nombre = "Adson Henrique";` with 'String' in blue, 'nombre' in white, '=' in white, 'Adson Henrique' in yellow, and ';' in white. The second line is `String? apellidos = null;` with 'String?' in blue, 'apellidos' in white, '=' in white, 'null' in light blue, and ';' in white.

Ilustración 22. Ejemplo de código (Null safety)

Elaboración propia

- Adopción incremental: se puede elegir qué quiere migrar y cuándo migrar. Y lo puede hacer incrementalmente, no hace falta que se haga todo a la vez y para eso Dart tiene en su documentación muchas instrucciones de cómo hacerlo.
- Totalmente fiable: la seguridad de *null safety* es sólida. Si una variable es creada determinando que no puede tener un valor nulo, dicha variable nunca tendrá un valor nulo.

Además de introducir el operador `?`, como se ve en la imagen anterior, Dart y Flutter introducen también el operador *late*. A diferencia del signo de interrogación, que tiene el objetivo de explícitamente decir que determinada variable acepta valores nulos, el operador *late* te permite crear una variable sin la necesidad de atribución de ningún valor a ella.

En la ilustración 23, hay un ejemplo del uso del operador *late*, donde naturalmente el compilador de Dart no permitiría que la variable *numeroHabitantes* fuese creada sin atribución de ningún valor, sin el uso de *late*. Con el uso de ese operador, se puede atribuir el valor para la variable después de la creación. [35]

```
1 class ejemploTFM {  
2   late int numeroHabitantes;  
3  
4   ejemploTFM() {  
5     numeroHabitantes = calculaNumeroHabitantes();  
6   }  
7  
8   numeroHabitantes = numeroHabitantes + 500;  
9 }
```

Ilustración 23. Ejemplo de código utilizando el operador *late*

Elaboración propia

Otro importante punto para tener en cuenta con ese operador es que en tiempo de ejecución la variable se inicializa de forma perezosa (*lazy initialize*). Por ejemplo, teniendo en cuenta la imagen arriba, la variable *numeroHabitantes* recibe el resultado de la función *calculaNumeroHabitantes*. Con el operador *late*, Dart o Flutter retrasa la ejecución de la función hasta que la variable sea usada por primera vez, lo cual en la imagen ocurre en la línea 8.

3.11.3. Animaciones y transiciones

Las animaciones y transiciones son un importante recurso utilizado hoy en día en el mundo de los móviles. Los usuarios esperan cada vez más que las aplicaciones tengan movimientos visuales y que esos movimientos sean fluidos; lo que agrega mucho en la experiencia del usuario. [8]

Flutter ofrece widgets de animaciones y transiciones que ayudan al desarrollador a crear múltiples animaciones e interacciones con el usuario final. Así incluye desde widgets para animaciones sencillas hasta widgets más avanzados que te permite hacer animaciones más complejas que necesitan de cálculos matemáticos. En seguida una explicación breve de alguno de los más comunes:

- *AnimatedContainer* – es un widget sencillo que te permite crear animaciones sencillas. Este widget cambia sus valores gradualmente de manera automática; lo único que hay que hacer es informar los valores iniciales, los valores finales y el tiempo de la animación. En la siguiente imagen hay un ejemplo del funcionamiento de dicho widget de animación.



Ilustración 24. Ejemplo de un *AnimatedContainer*

Tomada de [36]

- *AnimatedCrossFade* – es un widget sencillo que te permite hacer un *cross-fade* entre dos elementos. Un *cross-fade* es cuando un componente se desaparece (*fade-out*) mientras otro aparece (*fade-in*) en el mismo lugar. En la siguiente tenemos un ejemplo.

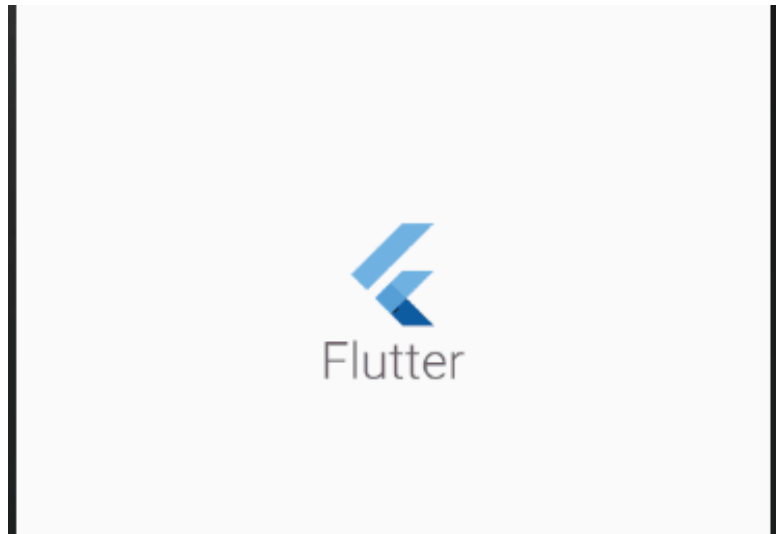


Ilustración 25. Ejemplo de un *AnimatedCrossFade*

Tomada de [37]

- *AnimatedDefaultTextStyle* – es un widget para animaciones de texto. Tiene el funcionamiento muy parecido a *AnimatedContainer* y *AnimatedCrossFade*. En la siguiente imagen tiene un ejemplo sencillo de cómo funciona.

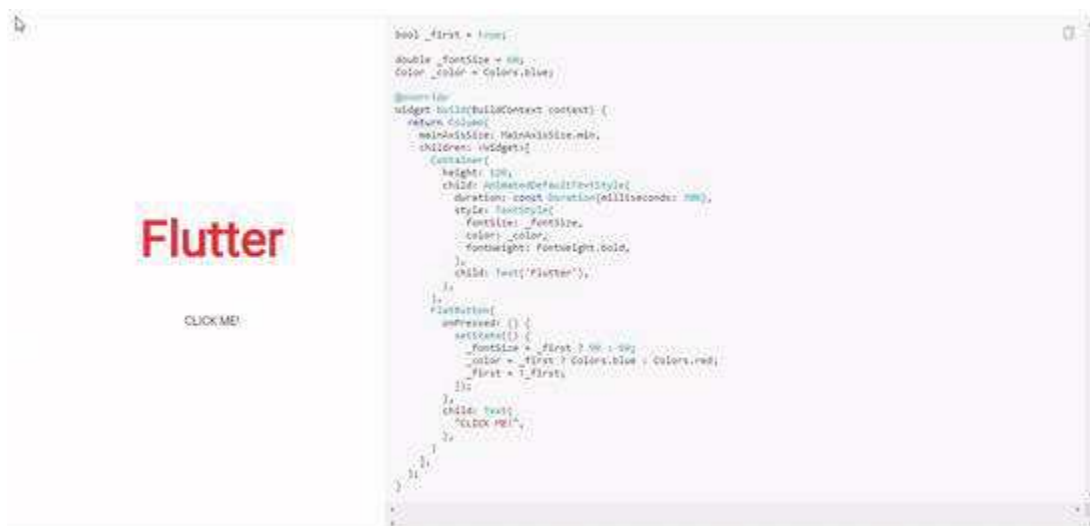


Ilustración 26. Ejemplo de un *AnimatedDefaultTextStyle*

Tomada de [38]

3.11.4. Categoría de Widgets

Como se ha dicho anteriormente, en Flutter todo es widget. Y hasta el momento de la redacción de ese documento, hay más de 150 tipos distintos teniendo en cuenta *Cupertino* widgets (iOS) y *Material Design* widgets (Android). Cada uno con su particularidad, con su objetivo. [39]

El propio Google, en su página web oficial sobre Flutter, ha organizado todos esos widgets por categoría, para que se quede más fácil la búsqueda y también porque hay widgets que hacen cosas muy similares a otros. A ejemplo, en la ilustración 27, podemos mirar la categoría de widgets texto, en la cual hay 3 widgets: *DefaultTextStyle*, *RichText* y *Text*. Podemos observar también que en cada widget contiene una imagen que debería representarlo, el nombre y una breve descripción de lo que hace.

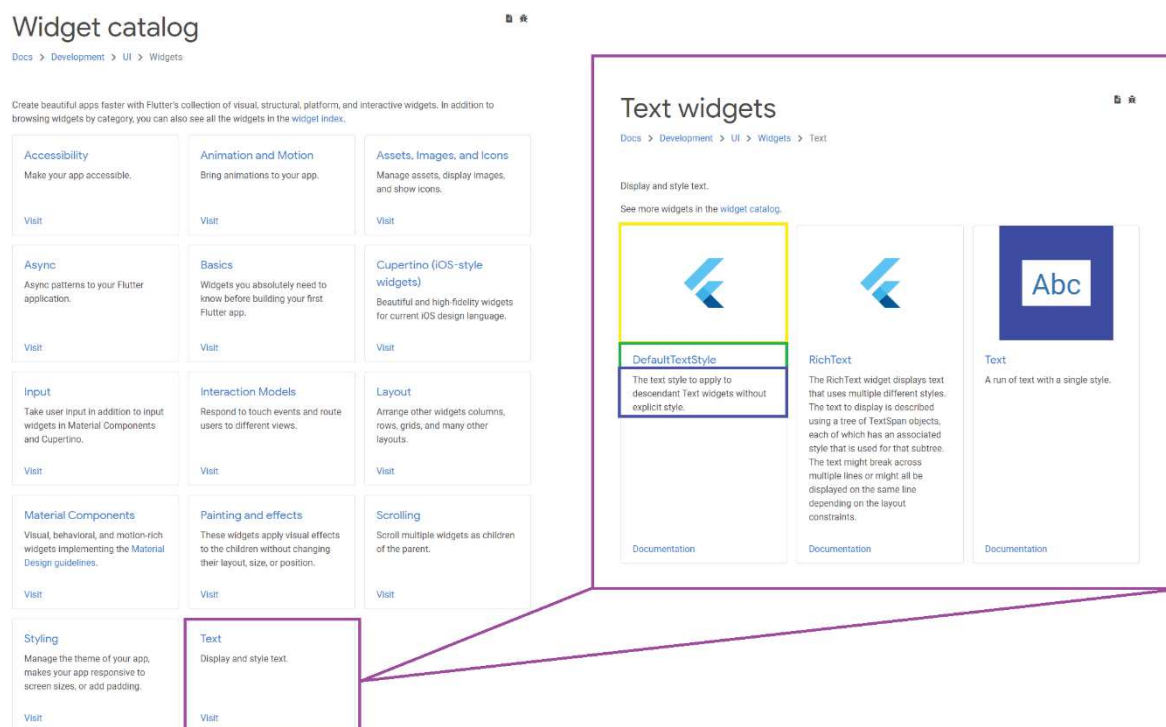


Ilustración 27. Categoría de widgets

Tomada de [40]

En seguida una breve explicación de las categorías de widgets:

- *Layout* – los widgets que se encuadran en esta categoría tienen por objetivo la disposición, dirección, y tamaño en el espacio de los widgets. En otras palabras, son responsables por la organización visual de la aplicación. Los widgets más comunes y utilizados en esta categoría son *Column*, *Row*, *Container*, *Padding*, *Center* y *Align*.
- *Material Components widgets* - como el propio nombre indica, esa categoría incluye los widgets creados para ser utilizados en Android (o al menos debería). En esa categoría encontramos widgets muy importantes como *MaterialApp* y *Scaffold*. Mientras *MaterialApp* es responsable de renderizar componentes en la pantalla principal (la primera en aparecer cuando se abre la app), *Scaffold* es responsable de proporcionar el diseño básico de la aplicación utilizando otros widgets de esa misma categoría como por ejemplo *AppBar*, *Drawer*, *BottomNavigationBar*, o *FloatingActionButton*.
- *Cupertino widgets* – de forma similar a la categoría anterior, en esta encontramos los principales widgets para iOS. *CupertinoPageScaffold* es la versión de *Scaffold* para iOS y tiene las mismas responsabilidades, pero ahora aplicadas a iOS, utilizando widgets como *CupertinoNavigationBar*, *CupertinoButton* entre otros.
- *Input* – en esa categoría están los widgets que proporcionan al usuario una manera de insertar datos en la aplicación, sea por medio de formularios utilizando el widget *Form* o con un simple campo de entrada de texto utilizando el widget *FormField*.
- *Styling* – aquí se encuentran todos los widgets responsables de la parte de estilización visual de la aplicación. Un widget muy importante en esta categoría es *Theme*, porque a través de él es posible cambiar todo el temario de la aplicación, como colores, fuentes y muchas otras cosas (cuando se empieza a desarrollar una aplicación en Flutter, ya viene con un tema por defecto de Flutter).
- *Async* – estos widgets no son visuales, pero tienen el objetivo de ayudar en la aplicación de manera directa. En ese caso con metodologías de

desarrollo asíncronas con el uso de widgets como *FutureBuilder* y *StreamBuilder*.

- *Accessibility* – es una categoría que engloba importantes widgets, ya que la accesibilidad es un tema importante en los días de hoy. Widgets como *Semantics* permite que herramientas de accesibilidad o, motores de búsqueda determinen cual es el objetivo de la aplicación.
- *Assets, images and icon* – como el propio nombre implica, en esa categoría tenemos todos los widgets (*Icons*, *Image*, *RawImage*, *AssetBundle*, etc.) responsables de poner archivos, imágenes y iconos en la aplicación.
- *Interaction model* – los widgets de esa categoría son los responsables de proporcionar una interacción más efectiva del usuario con la aplicación. En otras palabras, el objetivo es que al utilizar uno de los widgets de esa categoría el usuario tenga una experiencia más inmersiva, nativa mientras usa la aplicación. Widgets como *GestureDetector* intentan detectar gestos del usuario para interactuar con la aplicación.

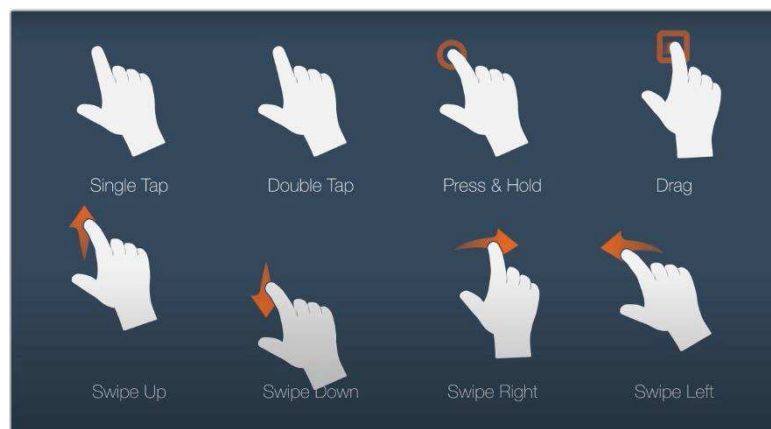


Ilustración 28. Ejemplo de gestos utilizando un móvil

Tomada de [41]

- *Scrolling* – widgets que son capaces de hacer que sus hijos (otros widgets) se desplacen por la pantalla. Es decir, permiten que haya más contenido en una misma pantalla a través del desplazamiento de contenido horizontal o verticalmente, por medio de gestos o barras de desplazamiento. Por ejemplo, el widget *ListView* permite crear una lista con los datos que desees.

3.11.5. El lenguaje Dart

Como se comentó anteriormente Dart, tiene muchas cosas en común con otros lenguajes de programación modernas, como por ejemplo Typescript. De igual manera, también tiene algunas particularidades, lo que hace con que Dart se destaque entre las demás.

A diferencia de lenguajes comunes como Java, donde hay diferencia entre tipos de datos primitivos (números, texto) y los que no lo son, en Dart todo es tratado como un objeto; incluso los tipos primitivos como números, textos, funciones y hasta NULL son objetos. El hecho de que todo sea tratado como objeto en Dart, hace que sea un lenguaje totalmente orientado a objetos, lo que significa que utiliza el patrón de desarrollo POO (Programación Orientada a Objetos).

Otra cosa importante que debemos mencionar es que, en Dart, a diferencia de Java, por ejemplo, no es necesario especificar la visibilidad de elementos dentro de una clase con las palabras *public*, *private* y *protected*. En Dart, todo es público a menos que lo ocultes implícitamente poniendo la señal de guion bajo (_). En la ilustración 29 es posible ver la diferencia entre la declaración de visibilidad entre los lenguajes Java (rectángulo naranja) y Dart (rectángulo verde).

La ilustración muestra dos fragmentos de código comparados. El código de Java (encuadrado en naranja) muestra la declaración explícita de la visibilidad: `public class Persona {` en la línea 1 y `private Name() {` en la línea 3. El código de Dart (encuadrado en verde) muestra la declaración sin especificar visibilidad: `class Persona {` en la línea 8 y `Name() {` en la línea 9. Ambos fragmentos continúan con `// TODO` y se cierran con `}` en las líneas 5 y 6 para Java, y 11 y 12 para Dart.

Ilustración 29. Java X Dart

Elaboración propia

En Dart hay muchas maneras de declarar una variable, cada una con su particularidad. A pesar de que Dart es un lenguaje fuertemente tipado, no hace falta

que la declaración de tipos sea explícita porque Dart (al igual que otros lenguajes como Python) tiene inferencia de tipos, lo que permite omitir el tipo de variable al declararla. La ilustración 30, muestra algunas maneras de declarar una variable en Dart: concretamente, en la línea 2 se realiza la inferencia de tipo usando simplemente la palabra clave *var* (así como en Javascript y otros lenguajes) Dart infiere el tipo debido al valor que la variable recibe (*texto = String*); en la línea 3 declarando la variable directamente con el tipo *String*; en la línea 5 y 6 usando la palabra clave *dynamic* que te permite cambiar el tipo de la variable en tiempo de ejecución, en ese caso del tipo texto (*String*) para número (*Int*) y en la línea 8 usando la palabra clave *Object* (dado que, como se ha dicho anteriormente en Dart todo es un objeto, entonces esa es una manera de declarar una variable).

```
1
2  var name = "Adson";
3  String apellido = "Moreira da Silva";
4
5  dynamic f = "Universidad de Jaén";
6  f = 55;
7
8  Object n = "Flutter";
9
```

Ilustración 30. Maneras de declaración de variables en Dart

Elaboración propia

Las funciones en Dart también presentan algunas diferencias si comparadas con otros lenguajes de programación. Dart es considerado un lenguaje de alcance léxico (*lexically scoped*) lo que significa que el alcance de una variable es determinado estáticamente, por la forma en que el código está escrito. En la ilustración 31 hay un ejemplo de cómo eso funciona en la práctica, donde debido a la forma en que está escrito el código, la variable *topLevel* tiene su alcance hasta la última función *nestedFunction*. [42]

```
bool topLevel = true;

void main() {
  var insideMain = true;

  void myFunction() {
    var insideFunction = true;

    void nestedFunction() {
      var insideNestedFunction = true;

      assert(topLevel);
      assert(insideMain);
      assert(insideFunction);
      assert(insideNestedFunction);
    }
  }
}
```

Ilustración 31. Alcance léxico en Dart

Tomada de [42]

Gracias a que Dart no tiene una gran curva de aprendizaje, cualquier persona que tenga interés en ese lenguaje pueden aprenderlo muy fácilmente. Claro que en Dart hay muchas otras particularidades y curiosidades además de las mencionadas aquí, pero en su documentación oficial puede encontrar de todo que hay en ese lenguaje, hasta el modo en que *Null safety*, es tratado en Dart.

Google, para que las personas interesadas en aprender, no necesiten instalar todo el SDK de Dart antes de probar el lenguaje, ha construido un sitio web llamado *DartPad* (<https://dartpad.dev>). La web hace una emulación de como el SDK de Dart trabaja, lo que permite hacer pruebas con el lenguaje. Claro que en *DartPad* no se puede hacer todo que el lenguaje propone, porque hay algunas limitaciones en esa web.

3.12. Flutter 2.0

En 3 de marzo de 2021, Google anunció el lanzamiento de Flutter 2.0 y junto con ese anuncio, una serie de novedades. Esa nueva versión trae más compatibilidad

entre multiplataformas además de que agrega más plataformas como Windows, macOS y Linux.

La primera novedad más importante anunciada es que en esa nueva versión estable, corrigieron muchos errores y crearon nuevos widgets tanto para iOS como para Android. El asunto de los errores (muchas *issues* en el repositorio de Github de Flutter) era uno de los más esperados, ya que la comunidad requería eso por mucho tiempo. No obstante, hay mucho trabajo por hacer a este respecto.

Junto con esa versión 2.0, Dart también ha ganado una nueva versión: la 2.12. En esta nueva versión, Dart utiliza Null Safety por defecto. Entonces a partir de esa versión, tanto de Flutter como de Dart, el compilador de Dart validará el código para que sea totalmente compatible con Null Safety. Y eso implica que las aplicaciones que estén en las versiones antiguas y deseen migrar para la nueva, tendrán que validar el Null Safety. De igual modo, los paquetes y *plugins* en Flutter también tienen que agregar esa nueva funcionalidad. Según la documentación de Flutter sobre la nueva versión [45], más de mil paquetes fueron publicados con la validación de Null Safety.

Otra importante novedad de esta versión es que Flutter web (la versión de Flutter para desarrollo web), ha pasado de beta a versión estable. Eso quiere decir que, a partir de ahora, Flutter lleva la capacidad de reutilización de códigos (antes solo en dispositivos móviles) para la plataforma web y que también la comunidad ya puede utilizar Flutter web para aplicaciones en producción.

Adicionalmente Google ha anunciado Flutter para aplicaciones de escritorio. Es decir, la utilización de Flutter para creación de aplicaciones para plataformas como macOS, Windows y Linux. Además de eso, Canonical (la compañía por detrás del sistema operativo Ubuntu) ha anunciado que Flutter será desde ahora la opción predeterminada para futuras aplicaciones de escritorio y móviles creadas por la compañía. En realidad, estas variantes de Flutter están aún sus versiones iniciales (*beta snapshot*) pero se espera que la versión estable esté lista para fin de ese año (2021) según la propia Google.

4. NEAT: CASO PRÁCTICO. DESARROLLO DE UN PROTÓTIPO DE APLICACIÓN MULTIPLATAFORMA UTILIZANDO FLUTTER

La app desarrollada para este TFM, Neat surge con el objetivo de ser una aplicación sencilla pero útil, que permite mostrar y experimentar algunas de las diversas herramientas de Flutter además de ejemplificar el funcionamiento del mismo en un caso práctico. Todo ello, no obstante, sin perder el propósito de crear una aplicación que sea útil.

Neat es una aplicación que permite que el usuario almacene datos de cita, notas, tareas y contactos localmente en su móvil. Funciona como una agenda para el usuario, de modo que pueda almacenar todas estas informaciones en la misma app.

Es importante decir que Neat, así como muchas otras aplicaciones existentes tiene mucho que mejorar, en muchos requisitos. Hay ideas de muchas otras funcionalidades, que serán citadas posteriormente, para incorporarse en la aplicación que llevarán la app para un próximo nivel, siendo mucho más útil para el usuario.

4.1. Herramientas y tecnologías utilizadas

A continuación, se mostrarán las tecnologías utilizadas en el desarrollo y en la construcción de la aplicación, así como una descripción y por qué fue utilizado.

4.1.1. Visual Studio Code

El editor de código Visual Studio Code o simplemente VSCode, fue elegido para el desarrollo de la aplicación. En la documentación de Flutter⁸, VSCode es citado como uno de los muchos editores de código posibles para el desarrollo con Dart y/o Flutter. También podría ser elegido Android Studio, que es un ambiente de desarrollo completo e integrado (IDE) y no solo un editor de código como VSCode. La opción por VSCode se debe casi que exclusivamente por ser una aplicación más leve (en comparación con Android Studio), más sencilla y que no exige un ordenador potente para ejecutarlo.

⁸ <https://flutter.dev/docs/get-started/editor?tab=vscode>

Adicionalmente, para el desarrollo de Flutter y/o Dart en VSCode, hay que instalar la extensión de Flutter proporcionadas por Google en la tienda de extensiones de VSCode.

4.1.2. Git

Git fue la opción elegida para mantener toda la documentación y código, generados por este trabajo, versionados. Es decir, fue el *software* de control de versión elegido para gerenciar todos los archivos generados por este trabajo. Fue elegido por su popularidad y también por la facilidad que proporciona sea utilizando una aplicación con interfaz gráfica o línea de comando.

Adicionalmente todos los archivos generados están alojados en un repositorio⁹ en Github, que es un proveedor creado para alojar código y/o documentos y que mantiene el control de versión usando justamente Git.

4.1.3. Notion

Notion es una aplicación que permite la creación de cosas como notas, calendarios, cuadros de *Kanban*, base de datos, dentro de muchas otras. En ese trabajo, Notion fue elegido para mantener todas las informaciones a respecto del desarrollo de este trabajo en un solo lugar. Es decir, la página¹⁰ de Notion de ese documento, se encuentra informaciones como: los sitios usados para búsqueda de información, notas, calendarios con las horas trabajadas en cada parte que compone ese documento, instrucciones sobre la aplicación Neat, los requisitos, historias de usuario y muchas otras.

4.1.4. Whimsical

Whimsical es una aplicación que permite la creación de documentos, flujogramas, notas adhesivas, mapas mentales y *wireframes*. Por ser una aplicación sencilla fue elegida para la creación de flujogramas de la planificación de este documento, de la estructura de la interfaz gráfica de la aplicación, de la estructura de la base de datos y casos de uso.

⁹ <https://github.com/AdSoNaTuRaL/tfm-uja-flutter>

¹⁰ <https://www.notion.so/TFM-Trabajo-Fin-de-M-ster-a01256bf528b4eed968583d49163fe8b>

4.1.5. AVD Manager

AVD Manager es una aplicación de Android Studio que permite crear un dispositivo Android (móvil) virtualmente para poder hacer pruebas sin tener la necesidad de usar un dispositivo físico. Fue elegido debido a su buena integración con las herramientas de Flutter.

4.1.6. SQFLITE

Como el almacenamiento de datos es un requisito para esa aplicación, se ha optado por utilizar el *plugin* sqflite de Flutter, que utiliza la base de datos de SQLite. El uso de SQLite se debe a sencillez que ofrece esa herramienta desde el uso sencillo de funciones y subrutinas, a la creación de una base de datos con una configuración previa mínima. Además, al ser una base de datos local, permite que cada usuario mantenga su información en su propio móvil.

4.1.7. Flutter Localizations

Flutter localizations es un paquete que viene junto con el SDK de Flutter, y cuya finalidad es gestionar la internacionalización de la aplicación. Es decir, si la aplicación tiene la necesidad de ser entendida en muchos idiomas, ese es un paquete que te ayuda hacer la “traducción” de la aplicación de manera sencilla y escalable.

Para Neat, ese paquete fue elegido por su facilidad y por venir integrado en el SDK de Flutter, además de que Flutter proporciona una documentación muy rica para ese paquete. Neat está traducido a tres idiomas: portugués, inglés y español, siendo el inglés el idioma predeterminado.

4.1.8. Shared Preferences

Shared Preferences es un *plugin* de Flutter que permite el almacenamiento de datos sencillos específico de cada plataforma (en iOS utiliza NSUserDefaults y en Android utiliza SharedPreferences). Fue elegido debido a su facilidad de utilización para el almacenamiento de una variable que controla si el usuario está utilizando la aplicación por primera vez o no. En caso de que esté usando por primera vez, la aplicación muestra al usuario las pantallas de *onboarding* que le enseña brevemente el funcionamiento básico de la app.

4.1.9. Intl

Intl es un paquete de Flutter que permite, así como el Flutter Localizations, gestionar internacionalización. En el caso de Neat, fue elegido debido a su facilidad para manejar formatos de fecha y hora, ya que la aplicación tiene muchos casos de uso en que la fecha y la hora son utilizados.

4.1.10. Flutter Launcher Icons

Flutter Launcher Icons es un paquete que permite cambiar los iconos de lanzador de la aplicación, sin tener la necesidad de hacer eso individualmente para cada plataforma. Fue elegido debido a su facilidad de uso y por contener una documentación completa proporcionada por el equipo de Flutter.

4.1.11. Flutter Calendar Carousel

Flutter Calendar Carousel también es un paquete de Flutter que permite la creación de visualización de un calendario que puede ser deslizado horizontalmente. Fue elegido por ser altamente personalizable y sencillo. En Neat, ese paquete fue utilizado en la parte de almacenamiento de citas, donde era necesario una vista de calendario.

4.1.12. Flutter Slidable

Al igual que alguno de los anteriores, Flutter Slidable es un paquete de Flutter que permite acciones de deslizar en listados. En la ilustración 32 se recoge una demostración de cómo funciona el paquete. Fue elegido por ser altamente personalizable y por ser sencillo para trabajar.



Ilustración 32. Ejemplo del paquete Flutter Slidable

Tomada de [43]

4.1.13. Scoped Model

Scoped Model es un paquete que, a diferencia de los anteriores, no trae ninguna forma de representación visual. Es un paquete que permite gestionar el estado de la aplicación en estado global, compartiendo datos de los widgets padres para sus descendientes. Fue elegido por que la forma en que trabaja con la gestión de los datos entre widgets padres e hijos se aplica perfectamente para el modelo de la aplicación. En el apartado de implementación, se puede encontrar información más detallada a respecto de esa herramienta, así como una explicación de su funcionamiento.

4.1.14. Figma

Figma es una aplicación que permite la edición de gráficos vectoriales y también una herramienta para hacer prototipos. Actualmente es una de las herramientas más utilizadas para hacer prototipos ya que cuenta con una versión gratis que es muy útil y versátil. También cuenta con una versión de pago que adicionalmente te permite hacer más cosas.

En ese trabajo fue elegido por ser una herramienta muy popular y por ofrecer muchas facilidades en la hora de prototipar el diseño de la aplicación Neat.

4.1.15. Barcode Scan

Barcode Scan es un *plugin* de Flutter que permite que el móvil tenga compatibilidad con el escaneo de códigos de barras, en Android y iOS. Fue elegido por tener una documentación bien descriptiva y por ser una herramienta fácil de utilizar.

4.1.16. QR Flutter

QR Flutter es un paquete de Flutter que permite la creación y lectura de códigos QR de forma sencilla y rápida. Fue elegida por ser una herramienta con gran popularidad en la tienda de Flutter y por tener una documentación descriptiva. En el caso de la aplicación, esa herramienta se utiliza para la lectura de códigos QR.

4.2. Metodología de desarrollo

Al elegir la metodología utilizada para el desarrollo de la aplicación se tuvieron en cuenta diversos factores, de los cuales los principales fueron: los plazos, los objetivos y el equipo.

Dado que el equipo estaba formado por una sola persona y que la experiencia con Flutter era inexistente, los requisitos no fueron estables, sino que cambiaron a lo largo del proyecto varias veces. Igualmente, otros requisitos se añadieron paulatinamente, haciendo con que surgiesen problemas que alteraron la planificación establecida. Uno punto positivo a respecto de no tener una fecha límite predeterminada, fueron los plazos flexibles que permitieron la organización de ese proyecto.

Por dichas razones, fue elegido la metodología SCRUM que tiene puntos clave que son óptimos para factores de esa naturaleza. Por ejemplo, el desarrollo iterativo e incremental fue adoptado en este proyecto ya que el “cliente” ha podido hacer el uso de software con las características primordiales desde el principio (MVP). Además, al haber iteraciones y entregas el cliente puede proporcionar *feedback* y así ayudar a mejorar el producto a cada iteración.

Con ese enfoque incremental e iterativo, el proyecto fue dividido en pequeñas partes (iteración, sprint) y a cada pequeña parte, cada iteración, fueron aplicadas las etapas de análisis, diseño, implementación y pruebas.

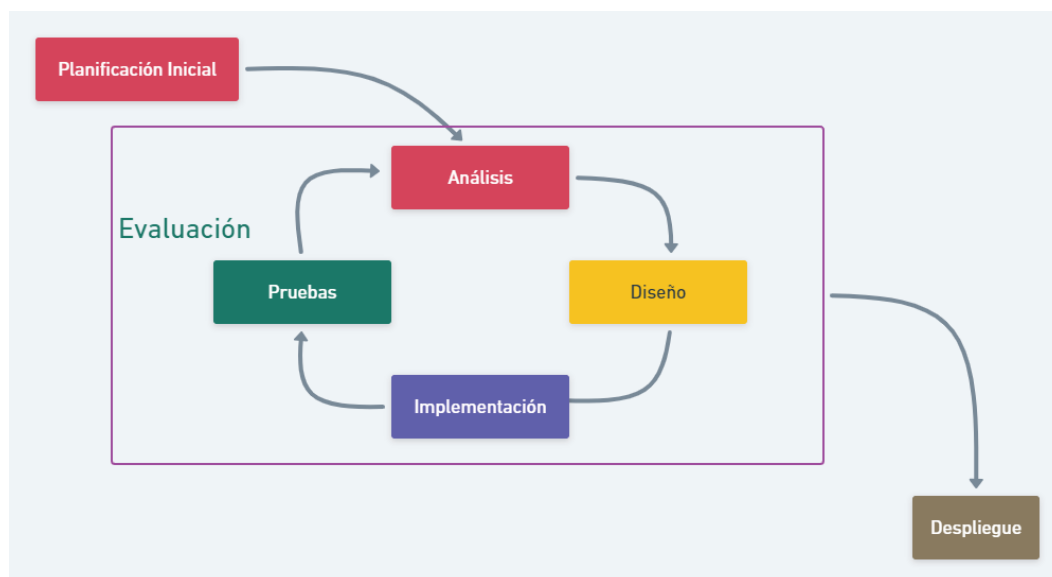


Ilustración 33. Proceso iterativo e incremental

Elaboración propia

Por cada sprint, cuya duración es de 1 semana, hay una evolución del producto (requisitos funcionales – RF o requisitos no funcionales RNF) a partir de las conclusiones de las iteraciones anteriores, sea añadiendo nuevos requisitos o mejorando los que ya fueran completados.

4.2.1. Definición de requisitos

A la continuación se establece los requisitos para el producto Neat, además de contener las historias de usuario para cada requisito, en cual sprint fue desarrollado, una medida de prioridad y el estado en que el requisito se encuentra.

CRUD Tareas		RF1
Como usuario, quiero ser capaz de poder crear, editar, listar y borrar tareas en la aplicación para poder realizar el seguimiento de mis tareas.		
Prioridad: Alta	Estado: Hecho	Sprint: 2

CRUD Notes		RF2
Como usuario, quiero ser capaz de poder crear, editar, listar y borrar notas en la aplicación para poder realizar el seguimiento de mis notas.		
Prioridad: Alta	Estado: Hecho	Sprint: 1

CRUD Contactos		RF3
Como usuario, quiero ser capaz de poder crear, editar, listar y borrar contactos en la aplicación para poder realizar el seguimiento de mis contactos.		
Prioridad: Alta	Estado: Hecho	Sprint: 3

CRUD Citas		RF4
Como usuario, quiero ser capaz de poder crear, editar, listar y borrar citas en la aplicación para poder realizar el seguimiento de mis citas.		
Prioridad: Alta	Estado: Hecho	Sprint: 4

Autenticación de usuario		RF5
Como usuario, quiero ser capaz de poder autenticarme en la aplicación para poder almacenar mis informaciones con seguridad, de la misma forma que quiero poder salir de la aplicación y la misma no mantener mis datos guardados localmente.		
Prioridad: Baja	Estado: Por hacer	Sprint: -

CRUD Enlaces		RF6
Como usuario, quiero ser capaz de poder crear, editar, listar y borrar enlaces en la aplicación para poder realizar el seguimiento de mis enlaces.		
Prioridad: Medio	Estado: Hecho	Sprint: 7

Creación de <i>onboarding</i>		RNF1
La aplicación debe ser capaz de ofrecer pantallas de <i>onboarding</i> para que así el usuario pueda aprender lo que la aplicación ofrece y como puede operarla.		
Prioridad: Medio	Estado: Hecho	Sprint: 5

Creación de pantalla de bienvenida		RNF2
La aplicación debe ser capaz de ofrecer una pantalla de bienvenida para el usuario para que así, mientras el usuario haga cambios de aplicaciones o cualquier el motivo, la aplicación pueda exhibir una pantalla amigable.		
Prioridad: Medio	Estado: Hecho	Sprint: 5

Internacionalización		RNF3
La aplicación debe ser capaz de tener soporte al menos en 3 lenguajes: inglés, español y portugués. El lenguaje por defecto debe ser el inglés.		
Prioridad: Medio	Estado: Hecho	Sprint: 6

Integración con servicios del móvil		RNF4
La aplicación debe ser capaz de integrarse con otras aplicaciones del móvil, para que así haya mayor interactividad de la aplicación. Por ejemplo, interacción con las notificaciones del móvil.		
Prioridad: Baja	Estado: Por hacer	Sprint: -

4.2.2. Casos de uso

Según los requisitos especificados y anteriormente mencionados, se puede identificar cuatro funcionalidades imprescindibles para el funcionamiento inicial de la aplicación de los cual, el usuario interactúa directamente: CRUD Notas, CRUD Contactos, CRUD Citas, CRUD Tareas y CRUD Enlaces.

En la ilustración 34 podemos mirar el diagrama de caso de uso general del proyecto, que contiene los cuatro principales requisitos.

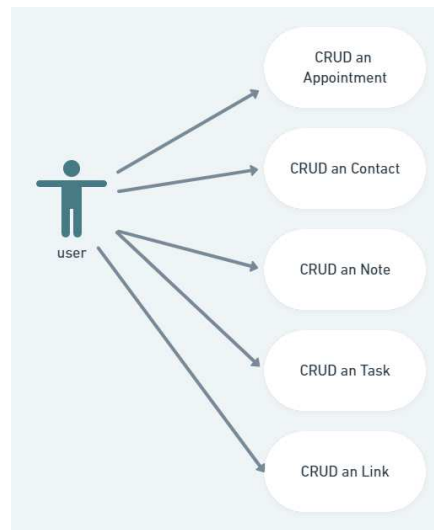


Ilustración 34. Casos de uso general

Elaboración propia

4.3. Implementación e Validación de la aplicación

Este apartado cubrirá las partes más importantes referentes a la implementación de la aplicación utilizando Flutter y las otras tecnologías mencionadas anteriormente. Es decir, las partes que exigieron más cuidado al ser desarrollado, patrones de diseño entre otras.

4.3.1. Estructura del árbol de widgets

Como se ha indicado en los apartados anteriores, todo en Flutter es widget y como el árbol de widget, algunas veces, puede crecer en exceso, haciendo con que la organización y separación del código sea comprometida. Dicho eso, antes de empezar el desarrollo de la aplicación decidí hacer un breve borrador de como sería la estructura del árbol de widgets de toda la aplicación que puede verse en la ilustración 35.

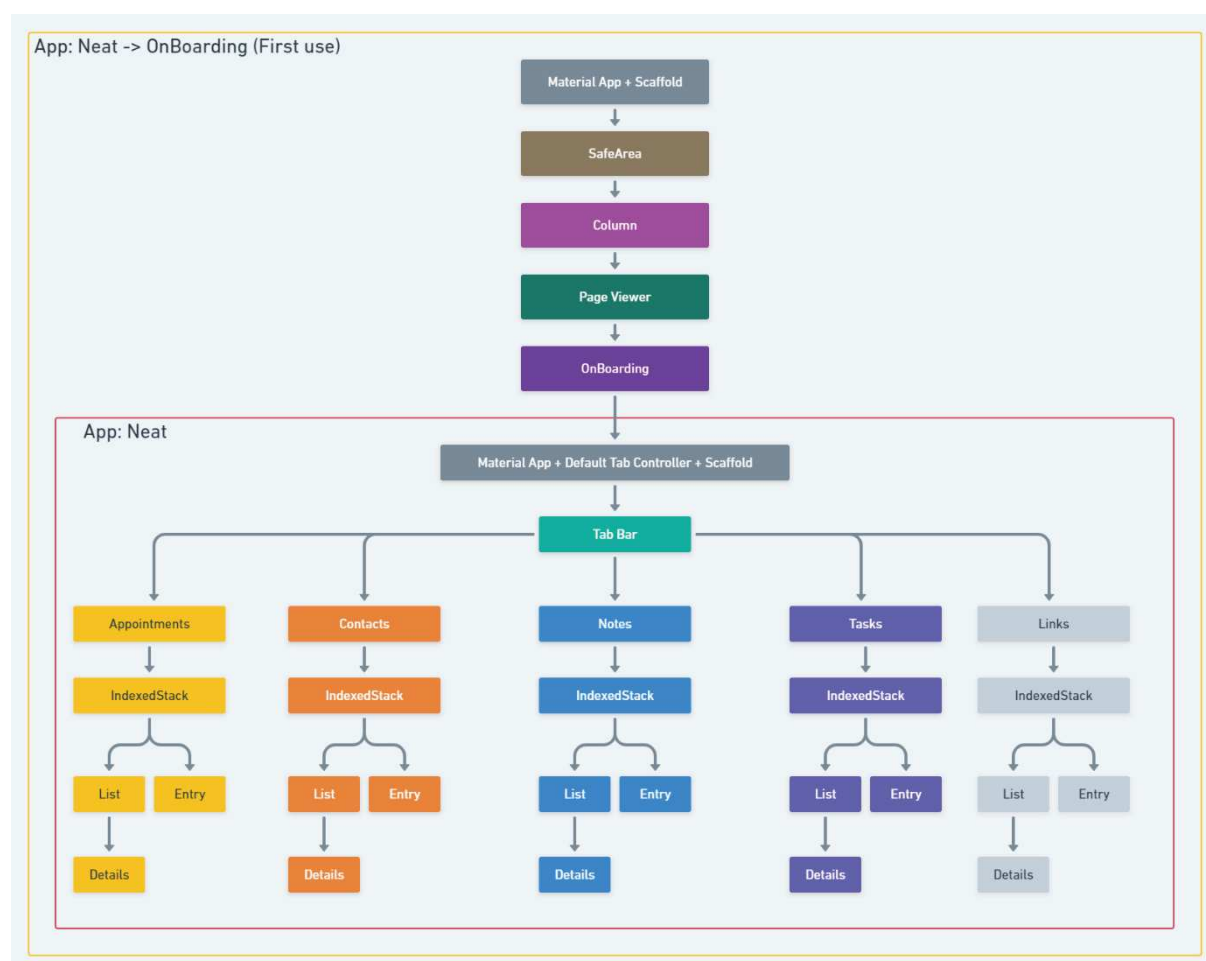


Ilustración 35. Estructura del árbol de widgets de la aplicación

Elaboración propia

La idea de la ilustración no es mostrar todos los widgets presentes en la aplicación, sino intentar demostrar cómo la estructura de la parte visual está montada para que haya un cierto patrón en el desarrollo. En la ilustración 35, en la parte más exterior (rectángulo amarillo) están los widgets que componen toda la parte de *onboarding* de la aplicación que solo es exhibida en la primera vez que el usuario accede a la app. Esta primera parte está formada por widgets como *MaterialApp* y *Scaffold* que generan el dibujo inicial de la aplicación; a continuación tenemos un widget llamado de *SafeArea* que sirve para que Flutter adapte la aplicación en móviles que contienen *notch*, que es muy presente en los móviles más actuales. En la ilustración 36, un ejemplo de móviles que contienen *notch*.

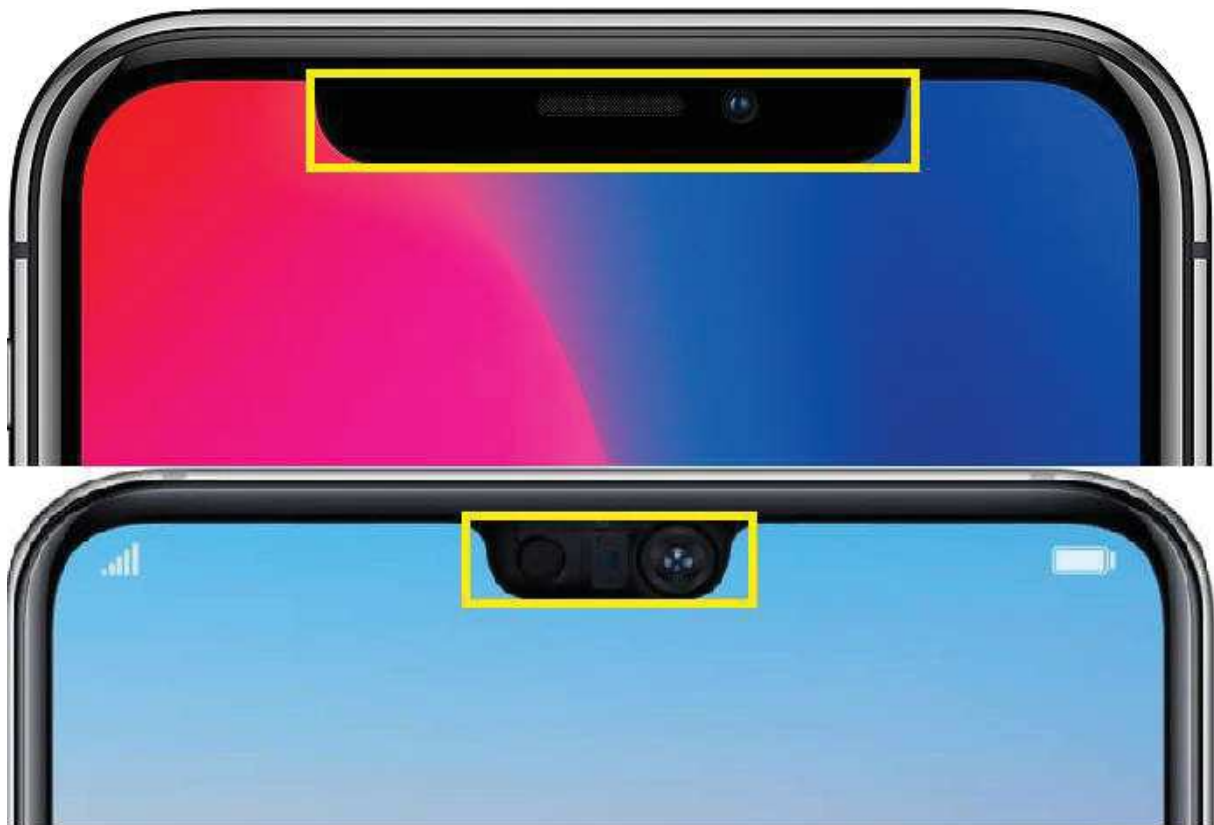


Ilustración 36. Ejemplo de móviles con notch

Elaboración propia

Tras el widget de *SafeArea*, tenemos un widget de *PageView* que hace una exhibición de páginas en una única visualización; eso permite que el usuario pueda deslizar la pantalla de derecha para la izquierda y/o viceversa, sin la necesidad de crear una página para cada visualización. Y por último el widget de *OnBoarding*, que es un widget personalizado, o sea, que fue creado por el autor de este trabajo.

Después del widget de *OnBoarding* los próximos widgets (rectángulo rojo) son los widgets principales de la aplicación propiamente dicha. De hecho, el widget padre también está compuesto por un *MaterialApp* y *Scaffold*, con la diferencia de que ahora hay un nuevo widget llamado *DefaultTabController*, que hace el control de las pestañas de la aplicación que puede ser visto en la ilustración 37.

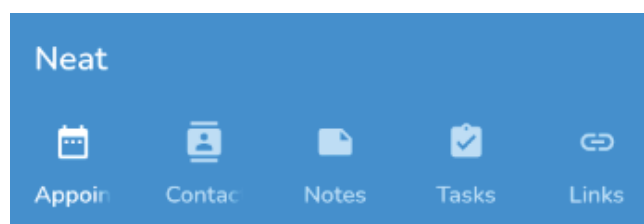


Ilustración 37. Pestañas de la aplicación

Elaboración propia

Más abajo, siguiendo la ilustración 35, encontramos el widget *TabBar* que controla, por ejemplo, los iconos mostrados, el título de cada pestaña, entre otras cosas. Y desde ahí pasamos entonces para cada widget individualmente: Citas (*Appointments*), Contactos (*Contacts*), Notas (*Notes*), Tareas (*Tasks*) y Enlaces (*Links*). Cada uno de estos widgets está compuesto por un widget *IndexedStack*, que a su vez está formado por un *List* (donde son listados los datos creados por el usuario) y un *Entry* (donde el usuario inserta los datos); finalmente cada *List* contiene más widgets que fueron resumidos en la ilustración como *Details*.

4.3.2. OnBoarding

Como se ha dicho anteriormente, la aplicación solo muestra el *onboarding* en la primera utilización. Eso es posible gracias al paquete, mencionado anteriormente, *Shared Preferences*, que permite grabar las preferencias del usuario localmente de forma sencilla. En la ilustración 38, mostramos el código que hace esa validación.

```
10  int firstTime = 0;
11  Run | Debug
12  Future<void> main() async {
13    WidgetsFlutterBinding.ensureInitialized();
14    SharedPreferences preferences = await SharedPreferences.getInstance();
15    firstTime = preferences.getInt('onBoardingAlreadyBeenSeen');
16    await preferences.setInt("onBoardingAlreadyBeenSeen", 1);
17    runApp(Neat());
18  }
55  home: firstTime != 1 ? OnBoarding() : Home(),
```

Ilustración 38. Uso de *Shared Preferences* en *OnBoarding*

Elaboración propia

Inicialmente, en la línea 10 de la ilustración 38, se encuentra la inicialización de la variable *firstTime* con el valor cero. A continuación, dentro de la función *main*, en la línea 13, podemos ver el uso de *Shared Preferences*, en lo cual la aplicación espera que este paquete obtenga una instancia de sí misma, pasando el valor para la variable

preferences. Luego en la línea 14, hay una asignación a la variable *firstTime*, que toma el valor *onBoardingAlreadyBeenSeen* guardado en las preferencias del usuario; si es la primera vez que el usuario usa la aplicación, ese valor viene como nulo ya que esa variable no existe en las preferencias del usuario y a partir de ese punto ya estará creada. En la línea 15, la variable es creada en las preferencias del usuario con el valor 1, lo que indica que el usuario ya ha visto las pantallas. Después en la línea 55, se comprueba una condición, siempre que la aplicación inicia, que verifica si el valor de la variable *firstTime* es diferente de uno; en caso que verdadero, el usuario es encaminado para el widget de *OnBoarding*, sino es encaminado para el widget *Home*.

Como la aplicación almacena datos localmente, en el caso de que el usuario borre los datos de la aplicación (lo cual incluye las preferencias del mismo) en la próxima utilización de la aplicación le mostrará nuevamente las pantallas de *onboarding*.

4.3.3. Prototipos del diseño de la aplicación

La parte de creación de prototipos fue una parte muy importante en todo el proceso de desarrollo de la aplicación. Los prototipos hechos fueron una manera de testear las ideas de la aplicación antes de empezar a desarrollarla. Fue básicamente el borrador de la idea, que ha tenido la función de validar la jerarquía del árbol de widgets, de las informaciones, el flujo de las pantallas y de la navegación entre pantallas.

Como el objetivo de los prototipos en el caso de Neat no eran demostrar toda la aplicación, sino validar y tornar la idea más madura, los prototipos no contienen todos los detalles y pantallas presentes en la aplicación real. En otras palabras, la aplicación real tiene más pantallas y detalles que los descritos en los prototipos.

Como se ha mencionado anteriormente, para el diseño de los prototipos se utilizó la herramienta Figma y los prototipos pueden ser vistos en el enlace <https://bit.ly/3vqWwKJ>, donde se puede mirar con más riquezas de detalles. En la ilustración 39 se puede ver los prototipos, pero de forma estática. Es importante decir también que la aplicación real ha sido muy fiel a los prototipos creados, teniendo poca diferencia entre la app y los prototipos.

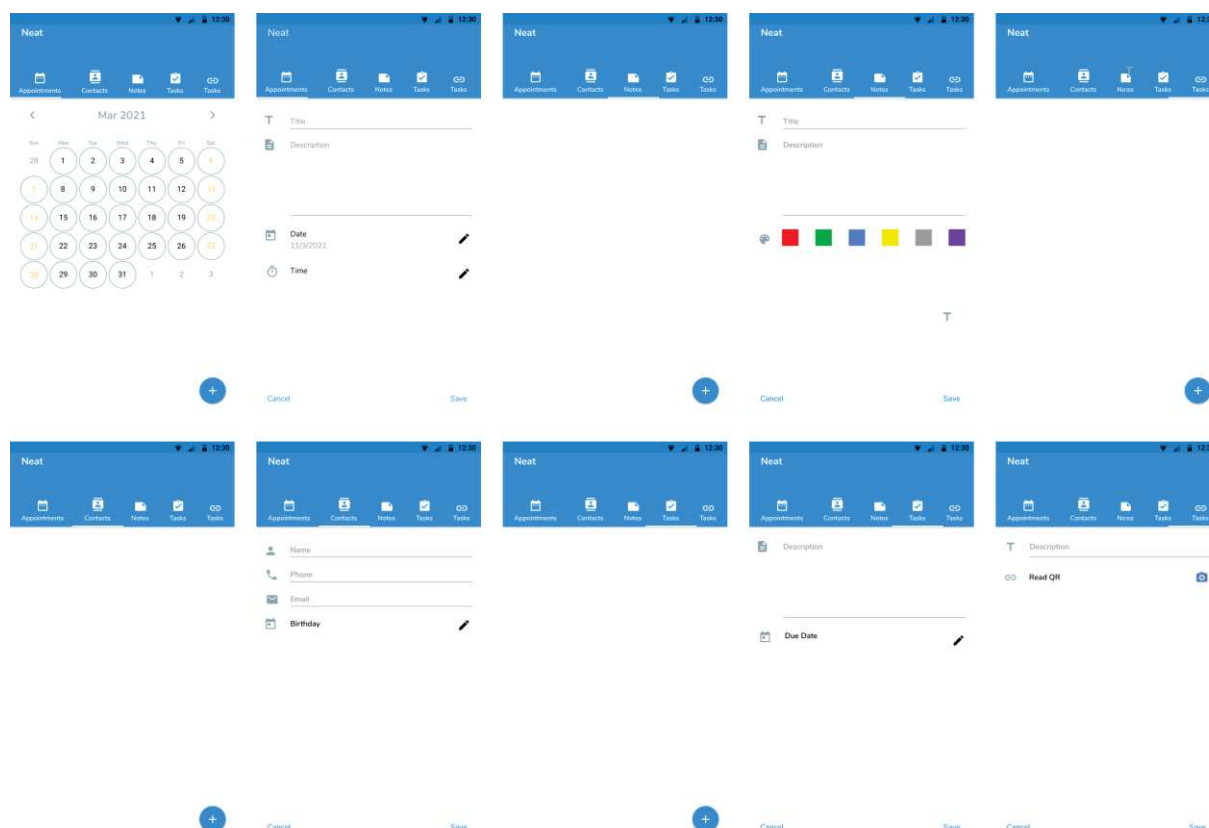


Ilustración 39. Prototipos de diseño de la aplicación

Elaboración propia

4.3.4. Base de datos

Dado que el almacenamiento de datos es un requisito para que la aplicación funcione de manera útil para el usuario, se hizo imprescindible la elección de una base de datos; concretamente, para la aplicación Neat, la opción fue SQLite. Por ser una de las bases de datos más utilizadas del mundo [44], ser una base de datos popular entre los móviles, ser rápida, pequeña, de almacenamiento local y fácil de implementar. De esta forma, SQLite se ha convertido en una base de datos increíble para aplicaciones individuales que enfaticen eficiencia, simplicidad y confiabilidad.

Por ser una aplicación sencilla y que no contiene muchas entidades (Notas, Contactos, Tareas, Citas y Enlaces) y ninguna relación entre esas entidades, el modelo de diseño de la base de datos tampoco es muy complejo. La ilustración 40, muestra como el diseño de la base de datos es sencillo donde también se puede notar que cada entidad de la aplicación tiene su propia base de datos y que cada una tiene su propia tabla y sus campos con el tipo entero o texto, que son los dos tipos primitivos que SQLite reconoce.

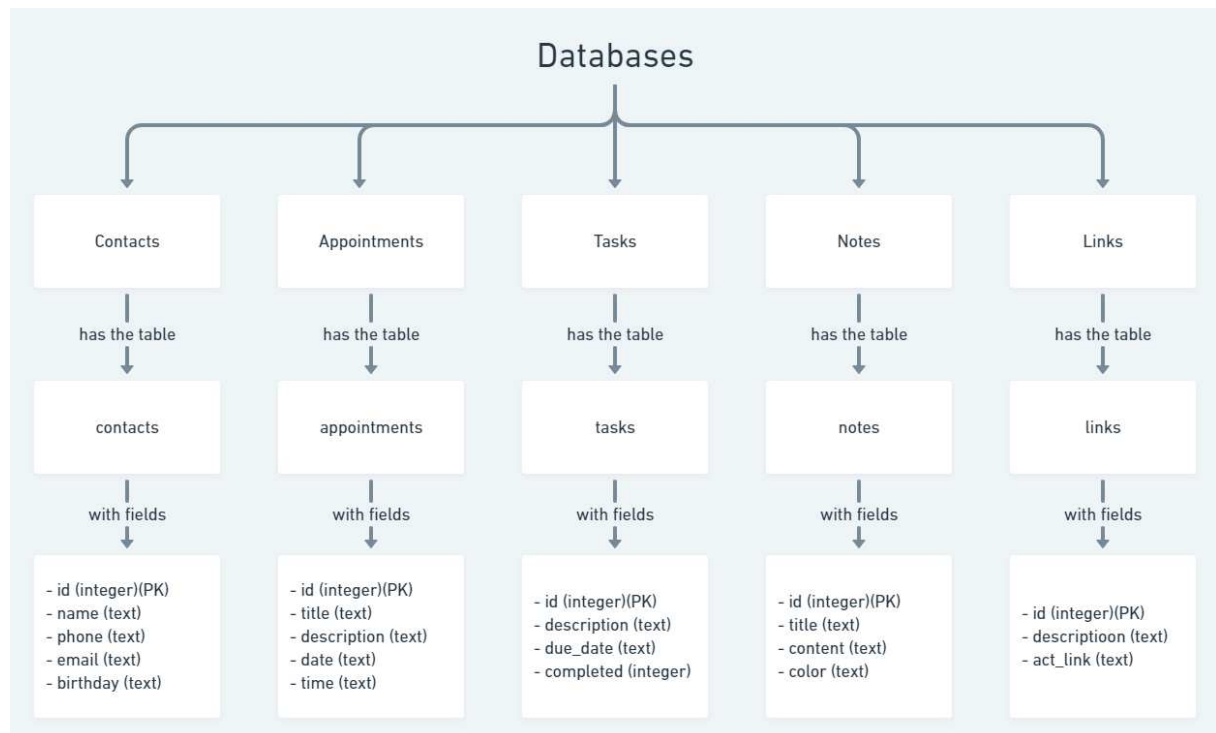


Ilustración 40. Modelo de diseño de la base de datos

Elaboración propia

La base de datos fue planificada para que cada entidad tenga su propia base de datos para que así, en un futuro, en el caso de necesitar hacer algún cambio con alguna entidad (como agregar más detalles o relaciones con otras herramientas o las propias tablas), la aplicación esté lista para recibir los cambios sin mucho desarrollo adicional.

4.3.5. Estructura del proyecto

Al igual que la mayoría de las aplicaciones desarrolladas en Flutter, todo el código, toda la aplicación se concentra prácticamente al 100% dentro de la carpeta *lib*. Y dentro de la carpeta *lib*, cada entidad tiene su propia carpeta con todos los contenidos relacionados a entidad dentro de ella. Así, todo lo relacionado a la entidad Contactos, por ejemplo, van estar en la carpeta *contacts*.

Dentro de cada carpeta de cada entidad se puede observar que hay un patrón en la nomenclatura de los archivos Dart. A modo de ejemplo teniendo en cuenta la entidad Notas:

- `Notes.dart` – Ese es el archivo principal de lo cual se puede acceder a las otras pantallas de Notas. En ese archivo también es donde está configurado el Scoped Model y el widget IndexedStack.
- `NotesDBWorker.dart` – Este archivo contiene el código que es entendido por la base de datos SQLite. Proporciona una capa de abstracción que permite cambiar de base de datos sin la necesidad de cambiar todo el código de la aplicación.
- `NotesEntry.dart` – Este archivo contiene todo el código necesario para dibujar la pantalla de entrada de datos de esa entidad.
- `NotesList.dart` – Este archivo contiene todo el código necesario para listar los datos ya insertados en la aplicación por parte del usuario.
- `NotesModel.dart` – Este archivo contiene una clase que representa la entidad, en ese caso Notas. Esa clase representa un objeto del modelo Notas que es exigido por Scoped Model.

Fuera de las carpetas de entidad, se pueden encontrar otras carpetas que representan pantallas y/o widgets como es el caso de las carpetas *home* y *onboarding*. También se pueden encontrar archivos individuales que de modo general son archivos o códigos comunes o útiles a varias partes de la aplicación. De este modo tenemos archivos como: *app_localizations.dart* (que es el archivo que contiene todo el código que hace la internacionalización del texto), *constants.dart* (que contiene las constantes que son usadas en la aplicación como colores, tiempo de animación), *size_config.dart* (que es un archivo que es útil para coger el tamaño de la pantalla del usuario), *utils.dart* (que contiene muchas funciones útiles para trabajar con formatos de fecha, etc.), *main.dart* (que es el archivo principal de la aplicación, es el primer archivo a ser leído por el compilador de Dart) y también *BaseModel.dart* (que contiene toda la configuración para la utilización de Scoped Model).

Fuera de la carpeta *lib*, se puede encontrar carpetas como *android*, que es la carpeta que contiene todos los archivos necesarios para que la aplicación sea compilada para Android, así como la carpeta *ios*. En algunos casos es necesario hacer modificaciones en archivos individuales de cada plataforma y justamente para eso las carpetas con las configuraciones individuales de cada plataforma están presentes. También se puede encontrar la carpeta *assets*, donde están los archivos que no

pertenecen a la aplicación, pero que se desea utilizar como imágenes, fuentes, iconos, etc. Adicionalmente encontramos la carpeta *lang*, que contiene tres ficheros JSON, los cuales almacenan todos los textos utilizados en la aplicación. Un JSON para cada idioma, así que si deseas añadir más idiomas, basta con crear un fichero JSON con las traducciones y algunas configuraciones extras en el archivo *main.dart*.

Otro archivo muy importante que está fuera de todas las carpetas, esto es, en la raíz del proyecto, es el archivo *pubspec.yaml* que es un archivo que contiene todas las dependencias utilizadas en la aplicación, así como muchas configuraciones tales como la configuración de fuentes o de imágenes (*assets*), entre otras.

Toda la estructura del proyecto puede ser verificada en la ilustración 41, donde también se han incluido archivos como *.gitignore* y *README.md*; ambos son archivos de configuración de Git, así como los otros archivos que no fueron citados son archivos de configuraciones de Flutter.

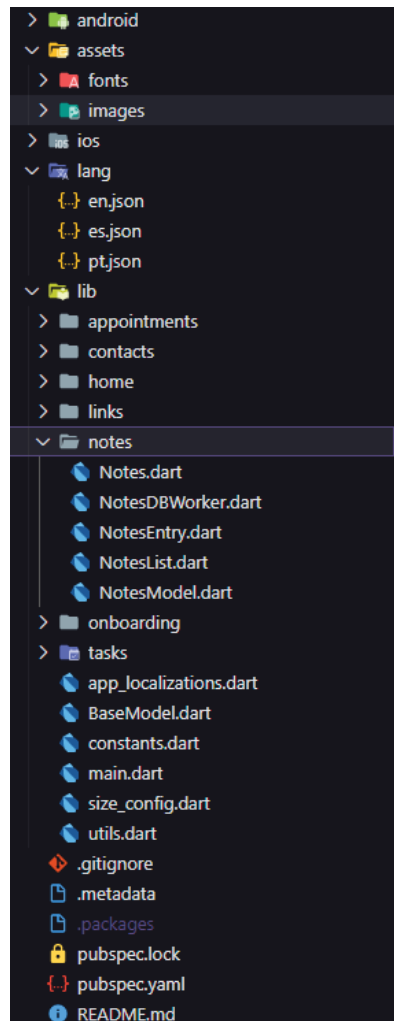


Ilustración 41. Estructura del proyecto

Elaboración propia

4.3.6. La lógica por detrás de Scoped Model

Antes de empezar a explicar la lógica por detrás de Scoped Model es importante dejar claro algunos puntos importantes.

El primer punto es que, así como muchas herramientas actuales que trabajan con programación reactiva (React, React Native, etc.), en Flutter existe la gestión de estados. De modo general podemos decir que en Flutter siempre hay dos estados: **estado local**, en el cual los widgets consumen y producen datos localmente, o sea solo el widget que gestiona esos datos tiene acceso a ellos; y **estado global**, que es justamente lo mismo a la diferencia que en esa situación, los widgets consumen o producen informaciones y datos entre ellos.

El segundo punto es que, como se ha dicho en capítulos anteriores, Flutter no ofrece una forma oficial de Google para trabajar con gestión de estados en la aplicación. Es decir, Flutter proporciona widgets que hacen gestión de estados, como por ejemplo *StatefulWidget* y a pesar de que este widget sea suficiente para muchos casos y sea bastante útil, no lo es en casos donde la gestión de estados tenga la necesidad de ser global. Ese widget solo ofrece la gestión de estados localmente, o sea, solo gestiona el estado local de un determinado widget. Pero por ejemplo en situaciones donde es necesario compartir los estados entre widgets (hijos y padres), ese widget se torna una herramienta limitada y difícil de trabajar.

El tercer punto a tener en cuenta es que, así como Google no ofrece una forma oficial de trabajar con la gestión de estados, no hay una forma canónica de hacerlo. Hay muchos paquetes en la tienda de paquetes de Flutter que ofrecen la funcionalidad de gestionar los estados local o globalmente. Pero la elección de la herramienta recae en el desarrollador, en función de cuál es su necesidad y lo que la herramienta te puede ofrecer. A modo de ejemplo, los paquetes que hacen gestión de estado en Flutter, más conocidos y usados son: BLoC, Redux, Provider, Scoped Model, InheritedWidget & InheritedModel, Fish-Redux y hay muchos otros. Pero cada uno con un enfoque distinto con sus ventajas y desventajas.

Dicho eso, para el desarrollo de Neat, la opción que más se encajaba en los criterios de la aplicación y que dispone de ventajas y facilidad en la implementación fue Scoped Model. Esa herramienta dispone de simplicidad al mismo tiempo que te ofrece gran control sobre la gestión de estados, ofreciéndote un conjunto de utilidades que permite pasar con facilidad una entidad de datos de un widget padre para sus hijos. Además de todo eso, cuando un dato de la entidad cambia, todos los widgets hijos de la entidad son reconstruidos para que la interfaz gráfica muestre los datos actualizados en tiempo real. Y esta es la herramienta ideal ya que la aplicación trabaja con entidades de datos (Notas, Citas, etc.) y a la medida que alguno dato de esas entidades cambien, todos los widgets hijos se reconstruyen.

Para el correcto funcionamiento de Scoped Model son necesarias algunas etapas: la primera de ellas exige la creación de una clase modelo, esa que extenderá de la clase *Model* de Scoped Model, y con eso en su clase modelo podrá añadir los métodos de lógica y/o de datos. Importante decir que ese primero paso sirve para que

el método *notifyListeners* pueda ser llamado dentro de su clase modelo y así notificar todos los widgets hijos a esta y así los widgets hacer la renderización de acuerdo con el cambio provocado. En las ilustraciones 42 y 43 tenemos un ejemplo de cómo eso está implementado en el caso de las entidades de Notas y Enlaces.

```
...
1  import '../BaseModel.dart';
2
3  NotesModel notesModel = NotesModel();
4
5  ...
6
7  class Note {
8    int id;
9    String title;
10   String content;
11   String color;
12   String toString() {
13     return "{ id=$id, title=$title, content=$content, color=$color }";
14   }
15 }
16
17 ...
18
19 class NotesModel extends BaseModel<Note> {
20   String color;
21   void setColor(String color) {
22     this.color = color;
23     notifyListeners();
24   }
25 }
```

Ilustración 42. Implementación de Scoped Model con la entidad de Notas

Elaboración propia

```
1 import 'package:Neat/BaseModel.dart';
2
3 LinksModel linksModel = LinksModel();
4
5 You, 6 days ago | 1 author (You)
6 class Link {
7   int id;
8   String actLink;
9   String description;
10  bool completed = false;
11
12  bool hasLink() {
13    return actLink != null;
14  }
15 }
16 class LinksModel extends BaseModel<Link> with LinkSelection {}
```

Ilustración 43. Implementación de Scoped Model con la entidad de Enlaces

Elaboración propia

En el caso de la ilustración 42, la clase *NotesModel* extiende de *BaseModel* que usa la clase *Notes* para establecer el tipo del modelo. Adicionalmente declara una variable color y un método para establecer el color (*setColor*) y dentro de ese método hace una llamada al método de *BaseModel* llamado *notifyListeners*. En otras palabras, en cualquier parte de la aplicación, cuando este método *setColor* es llamado, él ejecuta el método *notifyListeners* que provoca que todos los hijos de ese modelo (*listeners*) se rendericen nuevamente, ya que se supone un cambio de color en ese caso (cambio en la entidad de Notas).

En la ilustración 43, pasa lo mismo con la diferencia de que dentro de la clase de *LinksModel* no hay ninguna implementación de método de lógica y/o datos, lo que muestra que la necesidad de no tener un método implementado es normal. Eso no quiere decir que los métodos extendidos de *BaseModel* no pueden ser llamados; al contrario, incluso sin ninguna implementación todos los métodos extendidos pueden ser llamados, haciendo necesario solo que una instancia de esa clase sea creada. Otra diferencia es que en la ilustración además de extender *BaseModel* también extiende de *LinkSelection* (la palabra *with* en Dart permite hacer muchas extensiones, que en Dart es llamado de *Mixin*).

La segunda etapa exigida por *Scoped Model* es solo colocar el widget de *ScopedModel* en el árbol de widgets. La forma más común de hacer eso es envolviendo el widget que necesita de acceso al modelo con el widget *ScopedModel*. Generalmente, se envuelve con el widget de *ScopedModel* el widget que esté situado en la parte superior del árbol, es decir, el padre. La ilustración 44 muestra un ejemplo de cómo el widget *Scaffold* fue envuelto por *ScopedModel* (destacado con la línea roja) en la clase *ContactsList* (que sirve para listar los contactos creados por el usuario).

```
9 class ContactsList extends StatelessWidget {  
10   @override  
11   Widget build(BuildContext context) {  
12     return ScopedModel<ContactsModel>(  
13       model: contactsModel,  
14       child: ScopedModelDescendant<ContactsModel>(  
15         builder: (BuildContext context, Widget child, ContactsModel model) {  
16           return Scaffold(  
17             floatingActionButton: FloatingActionButton(  
18               child: Icon(Icons.add, color: Colors.white),  
19               onPressed: () async {  
20                 contactsModel.entityBeingEdited = Contact();  
21                 contactsModel.setBirthday(null);  
22                 contactsModel.setStackIndex(1);  
23               },  
24             ), // FloatingActionButton
```

Ilustración 44. Ejemplo del widget *ScopedModel* en el árbol de widgets

Elaboración propia

La tercera etapa es envolver los widgets hijos (aquellos que estén debajo de *ScopedModel*) con el widget *ScopedModelDescendant*. Al igual que en *ScopedModel* solo es necesario envolver los widgets que desean que se rendericen. Así como en el anterior, generalmente se envuelve el widget hijo más arriba en el árbol para que así todos sus hijos hereden las características de sus padres. En la ilustración 44 es posible mirar un ejemplo (destacado con la línea amarilla) de cómo esa tercera etapa es implementada.

A partir de ese instante, cualquier alteración en los datos de la entidad, es decir, en los modelos, hará que los widgets cambien de estado y así harán una renderización de acuerdo con la necesidad que el algoritmo de Flutter determine.

4.3.7. Informaciones adicionales

Como ya se ha comentado, las carpetas *android* y *ios* están presentes en todo proyecto creado con Flutter para dispositivos móviles. Eso porque hasta el momento, Flutter solo soporta esas dos plataformas móviles (Android y iOS), que dominan el mercado de dispositivos móviles. Las carpetas están presentes primeramente porque contienen configuraciones para la compilación en cada plataforma individualmente. O sea, la carpeta de Android tiene configuraciones para la compilación del código de Flutter para un dispositivo Android, de forma similar ocurre con iOS. Pero también algunas veces, se hace necesario cambiar algunas configuraciones individualmente para cada plataforma. En otras palabras, algunas veces, por más que la aplicación sea multiplataforma, es necesario hacer cambios individuales en una plataforma que no es necesario en otra.

En Android, por ejemplo, cuando es necesario utilizar algún sensor, o permiso del móvil en su aplicación Flutter, es necesario escribir el permiso en un archivo llamado *AndroidManifest.xml* que está presente dentro de la carpeta de *android*. En Neat, fue necesario modificar la carpeta de ambas plataformas individualmente para hacer configuraciones del icono de lanzador, ya que Android y iOS tienen configuraciones distintas para hacer eso.

Un punto importante es que eso no debe ser una preocupación para el desarrollador – trabajar con archivos nativos –, porque Flutter ofrece una documentación muy amplia y diversa sobre casi todas las configuraciones necesarias caso quiera hacer algo que involucre código nativo de las plataformas.

4.3.8. Validación de la aplicación

Este apartado irá mostrar la percepción de uso de la aplicación, obtenida tras poner la aplicación disponible a algunos usuarios finales. El objetivo final de la validación fue comprobar que la aplicación cumple con los requisitos mínimos y comprobar que su funcionamiento sea útil y no presente ningún error.

Para realizar esta validación, se creó un formulario utilizando la herramienta de Google Forms, donde las personas, que tuvieron acceso a la aplicación tenían que contestar preguntas como:

- ¿Cuál es su percepción o sensación principal al utilizar la aplicación?
- ¿Qué pensaste de la fluidez al usar la aplicación?
- ¿Podría decirnos qué diferencias ha notado entre el uso de esta aplicación y otras que utiliza habitualmente?
- ¿Tiene alguna sugerencia sobre cómo podría mejorar la aplicación?
- ¿Tiene algún conocimiento técnico en el desarrollo de aplicaciones móviles?

Teniendo eso en cuenta, es importante decir que todas las valoraciones recibidas a través del formulario fueron de personas que utilizan sistema operativo Android, ya que la aplicación disponible se ejecutaba solo en Android (gracias a que Android permite instalaciones de aplicaciones fuera de su tienda oficial de apps). Por el contrario, los móviles con iOS no permiten tal acción. A pesar de ello, se ha podido comprobar su funcionamiento en ambas plataformas.

La aplicación fue puesta disponible para 14 personas, de las cual, la gran mayoría ya tenía conocimientos previos en desarrollo móvil, lo que demuestra una experiencia previa con aplicaciones móviles. La ilustración 45 muestra el porcentaje de candidatos que ya tenían un conocimiento previo en desarrollo de aplicaciones móviles.

Finalmente, ¿tiene algún conocimiento técnico en el desarrollo de aplicaciones móviles?

14 respuestas

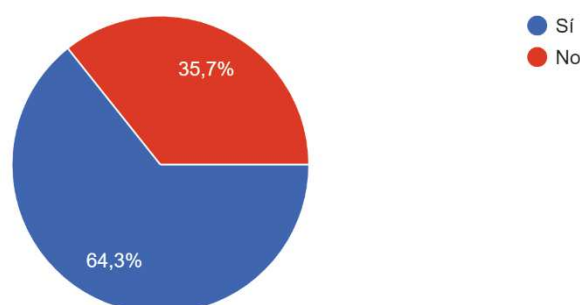


Ilustración 45. Usuarios que ya tenían conocimientos previos en desarrollo móvil

Elaboración propia

En relación a la fluidez de la aplicación, que engloba toda la usabilidad de ella como el flujo de pantallas, el diseño de la misma, si no ocurre ningún error o es una aplicación lenta, los usuarios que probaron la aplicación han elegido entre “Fluida” y “Muy fluida”. La ilustración 46 muestra el porcentaje de elección de los usuarios.

¿Qué pensaste de la fluidez al usar la aplicación?

14 respuestas

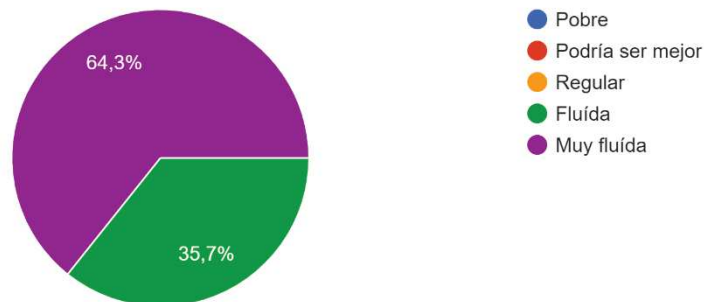


Ilustración 46. Fluidez de la aplicación

Elaboración propia

En términos de percepción o sensación al utilizar la aplicación los usuarios podrían escribir sus respuestas en forma de texto. De forma general, los usuarios han contestado que la aplicación es sencilla y útil, lo que al final era el propósito de la aplicación. En la ilustración 47 se puede ver una recopilación de las respuestas de los usuarios; incluye respuestas en español y portugués, ya que la aplicación fue puesta a disposición de usuarios de ambos idiomas a fin de comprobar si la aplicación reacciona correctamente al idioma del móvil del usuario. En la actualidad, la aplicación está disponible en portugués, inglés (idioma predeterminado) y español.

¿Cuál es su percepción o sensación principal al utilizar la aplicación?

14 respuestas

Uma aplicação simples, direta ao ponto e bem fluída
aplicativo muito rápido
Bueno, mi primera percepción fue: sencilla pero útil!
Sencillo de utilizarse
Mi sensación fue genial ya que hay un buen diseño y una experiencia de usuario muy agradable, así que pude entender cuál es la función de la aplicación.
Aplicação perfeita
Bien estructurado y de fácil manejo.
O aplicativo é de fácil utilização e intuitivo.
Me ha parecido una app muy útil y con diseño muy bonito
Interface amigável e similar as aplicações já conhecidas o que faz a utilização ser mais intuitiva
Facilidade e agilidade, pois ao utilizar o aplicativo tudo é muito intuitivo.
Fácil de usar
Aplicativo com layout clean e fácil para utilizar.
bastante útil para organizar a rotina diária

Ilustración 47. Percepción al utilizar la aplicación

Elaboración propia

En relación, a sugerencias de mejoras en la aplicación, los usuarios también pudieron expresar sus ideas en forma de texto. En ese caso, no hay un consenso entre la mayoría de los usuarios, dado que cada uno tiene una necesidad distinta de acuerdo con sus necesidades específicas. Pero la idea de poner notificaciones, alarmas o una alerta es la sugerencia que aparece más veces. De hecho, eso es uno de los requisitos de la aplicación, que puede ser visto en el apartado de casos de uso, pero que todavía no ha sido implementado. Muchas de esas ideas, que se puede encontrar en la ilustración 48, pueden ser tenidas en cuenta ya que algunas son válidas y quizás puedan ser implementadas en trabajos futuros.

¿Tiene alguna sugerencia sobre cómo podría mejorar la aplicación?

14 respuestas

Na aba de Tasks o teclado poderia ser retraído quando eu estivesse digitando a descrição e clicasse no ícone de data

não

Lo que he contestado arriba.

Para mi esta muy funcional

Para mejorar esta aplicación, implementaría notificaciones, y también tal vez niveles de prioridad en los eventos (alto, medio, bajo). Y tal vez para que sea más obvio sobre algunas funciones, por ejemplo, cómo eliminar las notas, eventos ... etc.

Um contador de dias e um cronometro

No tengo sugerencia.

Poder criar grupo de notas e poder adicionar links que não possuem qrcode

Quizás una guía inicial mostrando para que sirve la app y como funciona las principales herramientas.

O cadastro das tarefas possuir a possibilidade de incluir um tópico principal e vincular várias subtarefas

Seria muito interessante se adiciona-se um alerta/alarme/notificação no horário dos agendamentos.

Não

Resposta acima

notificação de quando chegar o momento de um evento

Ilustración 48. Sugerencias de mejoras para la aplicación

Elaboración propia

En cuanto a la diferencia entre esta aplicación y otras aplicaciones que los usuarios utilizan habitualmente, tampoco ha existido consenso entre los usuarios. Algunos han sugerido nuevas mejoras, mientras otros han hecho una comparativa entre Neat y las aplicaciones que utilizan. En la ilustración 49, se puede ver todas las respuestas.

¿Podría decirnos qué diferencias ha notado entre el uso de esta aplicación y otras que utiliza habitualmente?

14 respuestas

As animações sem pequenos travamentos nas transições de animação
A principal diferença foi a fluidez do app
Yo pienso que el design podría ser mejorado. No sé, algo como los botones, dibujos. Algo que hiciera la aplicación más colorida.
En otras aplicaciones hay muchas opciones y informaciones haciendo que sus funcionalidades no sean utilizadas del todo o con efectividad.
La primera impresión es que podría usar esta aplicación para muchos propósitos y sería útil ya que no necesitaría muchas otras aplicaciones.
Muito ágil, ajuda nas aplicações das atividades, agendas do dia a dia.
En este se puede leer QR Codes.
Os menus serem de fácil acesso na parte superior do aplicativo.
Un texto de ejemplo que muestra para que sirve cada herramienta, como por ejemplo "escriba aquí notas de algo que no puedes olvidar". Parece una tontería pero es muy importante en un primero contacto con la app. Pero de resto, la app se ve con un diseño muy bonito y es muy útil para usar en el día a día.
Utilizo muito ferramentas para controle de tarefas e tive mais dificuldade para organiza-las usando o aplicativo
A principal diferença é ter várias ferramentas em uma só!
Ele é intuitivo fácil de usar, a gente não apanha pra aprender
Clicar na data do calendário para incluir a tarefa. Atualmente só permite na data corrente e assim alterar a data.
melhor desempenho

Ilustración 49. Percepción de diferencias entre Neat y otras aplicaciones

Elaboración propia

De ese modo, teniendo en cuenta todas las respuestas, preguntas y el objetivo final de la aplicación, se puede concluir que la aplicación ha cumplido con su objetivo final de ser una aplicación útil, sencilla y de fácil utilización. Todas las respuestas del formulario pueden ser encontradas en: <https://bit.ly/31obzad>.

5. CONCLUSIONES Y LÍNEAS FUTURAS

5.1. Conclusiones

Con la realización de ese proyecto, con el intuito de analizar el kit de herramientas de Flutter y desarrollar un prototipo de aplicación multiplataforma, se ha podido notar que Flutter es una herramienta increíble. Aunque sea una herramienta relativamente nueva, ha presentado ventajas que ninguna otra herramienta del mismo campo tiene ahora mismo. No obstante, también ofrece un buen conjunto de desventajas, lo cual es hasta cierto punto normal dado que es una herramienta relativamente nueva.

Desde su primer lanzamiento estable (aproximadamente 2 años), Flutter ha evolucionado bastante. Hay muchas compañías invirtiendo no solo el tiempo en esa herramienta, sino que también soporte y dinero; lo que hace que la herramienta obtenga madurez de forma rápida y quién sabe si podría volverse la herramienta más fuerte en ese campo de IT. Así, encontramos que ya hay muchas compañías invirtiendo en Flutter como Microsoft o Canonical, igualmente encontramos casos de éxito de compañías que han adoptado Flutter para sus aplicaciones como es el caso de NuBank, la mayor banca digital del mundo.

Flutter ofrece muchas ventajas para quien desea desarrollar una aplicación para multiplataformas. Su SDK ya tiene resuelto muchos problemas que asustaban a las empresas cuando pensaban en desarrollar aplicaciones para distintos sistemas operativos. Ahora más que nunca, con el lanzamiento de la versión 2 estable, Flutter se muestra más fuerte, agregando más plataformas que darán soporte, corrigiendo muchos errores de la versión anterior; de hecho, ha conseguido que más compañías ayuden en el crecimiento de Flutter, lo cual es todo un logro. Flutter tiene todo para convertirse en la mejor herramienta para desarrollo multiplataforma.

Para finalizar, teniendo en cuenta los objetivos de ese proyecto y a pesar de algunas desventajas y problemas enfrentados durante el desarrollo del prototipo, Flutter es una herramienta que merece una oportunidad. A final, el análisis de Flutter ha sido increíble, lo que ha permitido obtener gran conocimiento de una herramienta nueva que quizás en un futuro próximo dominará el mercado. Además, el prototipo,

construido con la versión anterior de Flutter, ha sido una aplicación bien estructurada y que al final ha sido una aplicación útil para las personas que lo probaron.

5.2. Líneas futuras

A pesar de que el prototipo desarrollado en este proyecto sea muy útil, hay muchas formas de incrementarlo, volviéndolo más útil. Empezando con los requisitos/casos de uso que están por hacer y quizás incrementando más funcionalidades.

Tras el lanzamiento de la nueva versión de Flutter, un proyecto futuro, además de crear nuevas funcionalidades para las aplicaciones ya existente, sería expandir Neat para plataformas web y/o aplicaciones de escritorio ya que ahora Flutter es estable en dichas plataformas y así poder exhibir el poder de Flutter.

6. BIBLIOGRAFIA

- [1] Quora (2011). "What factors led to the bursting of the Internet bubble of the late 1990s?". Recurso digital <https://www.quora.com/What-factors-led-to-the-bursting-of-the-Internet-bubble-of-the-late-1990s>. [Accedido el 15/12/2020]
- [2] Netstar (2020). "How mobile phone technology has changed over the last 40 years". Recurso digital <https://www.netstar.co.uk/mobile-phones-years/>. [Accedido el 15/12/2020]
- [3] PuroMarketing (2011). "Google y Apple consolidan su dominio en el mercado de los smartphones". Recurso digital <https://www.puromarketing.com/12/10750/google-apple-consolidan-dominio-mercado-smartphones.html>. [Accedido el 16/12/2020]
- [4] StatCounter (2017). "Android overtakes Windows for first time". Recurso digital <https://gs.statcounter.com/press/android-overtakes-windows-for-first-time>. [Accedido el 15/12/2020]
- [5] StatCounter (2020). "Operating System Market Share Worldwide". Recurso digital <https://gs.statcounter.com/os-market-share>. [Accedido el 24/11/2020]
- [6] BCG (2014). "The mobile Internet Economy in Europe". Recurso digital <https://www.bcg.com/publications/2014/telecommunications-technology-digital-mobile-internet-economy-europe>. [Accedido el 02/12/2020]
- [7] We are Social (2020). "DIGITAL 2020: 3.8 BILLION PEOPLE USE SOCIAL MEDIA". Recurso digital <https://wearesocial.com/blog/2020/01/digital-2020-3-8-billion-people-use-social-media>. [Accedido el 14/12/2020]
- [8] Zammetti, Frank. (2019). "Practical Flutter: Improve Your Mobile Development with Google's Latest Open-Source SDK". Apress. [Accedido el 16/11/2020]
- [9] Microsoft (2020). "What is Xamarin". Recurso digital <https://dotnet.microsoft.com/learn/xamarin/what-is-xamarin>. [Accedido el 15/12/2020]
- [10] Adobe PhoneGap (2020). "Build amazing mobile apps powered by open web tech". Recurso digital <https://phonegap.com/>. [Accedido el 15/12/2020].
- [11] Appcelerator (2020). "Titanium Mobile Development Environment". Recurso digital <https://www.appcelerator.com/Titanium/>. [Accedido el 15/12/2020]
- [12] GitHub (2020). "React Native". Recurso digital <https://github.com/facebook/react-native>. [Accedido el 15/12/2020]
- [13] Delia, Lisandro & Galdámez, Nicolás & Thomas, Pablo & Corbalán, Leonardo & Pesado, Patricia. (2015). "Multi-platform mobile application development analysis". ResearchGate. [Accedido el 15/12/2020]

- [14] Flutter (2020). "Flutter". Recurso digital <https://flutter.dev/>. [Accedido el 16/12/2020]
- [15] Skia (2021). "Skia Graphics Library". Recurso digital <https://skia.org/>. [Accedido el 18/01/2021]
- [16] Jaxenter (2019). "Results are in: Fully fledged Dart takes first in 2019 pool". Recurso digital <https://jaxenter.com/poll-results-dart-word-2019-154779.html>. [Accedido el 18/01/2021]
- [17] Flutter (2021). "Write your first Flutter app, part 1". Recurso digital <https://flutter.dev/docs/get-started/codelab>. [Accedido el 18/01/2021]
- [18] Flutter (2021). "Hot reload". Recurso digital <https://flutter.dev/docs/development/tools/hot-reload>. [Accedido el 18/01/2021]
- [19] Dart (2021). "Codelabs". Recurso digital <https://dart.dev/codelabs>. [Accedido el 18/01/2021]
- [20] Google Trends (2021). "Flutter". Recurso digital https://trends.google.es/trends/explore?date=2018-01-01%202021-01-01&q=%2Fg%2F11f03_rzbq. [Accedido el 18/01/2021]
- [21] Flutter (2021). "Introduction to widgets". Recurso digital <https://flutter.dev/docs/development/ui/widgets-intro>. [Accedido el 18/01/2021]
- [22] Flutter (2021). "Test drive". Recurso digital <https://flutter.dev/docs/get-started/test-drive>. [Accedido el 18/01/2021]
- [23] Flutter (2021). "Apps take flight with Flutter". Recurso digital <https://flutter.dev/showcase>. [Accedido el 18/01/2021]
- [24] Flutter (2021). "Install". Recurso digital <https://flutter.dev/docs/get-started/install>. [Accedido el 18/01/2021]
- [25] Flutter (2021). "Flutter for Android developers". Recurso digital <https://flutter.dev/docs/get-started/flutter-for/android-devs>. [Accedido el 18/01/2021]
- [26] Flutter (2021). "Flutter documentation". Recurso digital <https://flutter.dev/docs> [Accedido el 18/01/2021]
- [27] Dos Santos, Eduardo (2020). "A importância de certificações para o desenvolvimento profissional e pessoal". Recurso digital <https://www.viceri.com.br/insights/a-importancia-de-certificacoes-para-o-desenvolvimento-profissional-e-pessoal>. [Accedido el 02/02/2021]
- [28] Google Developers Certification (2021). "Google Developers Certification". Recurso digital <https://developers.google.com/certification>. [Accedido el 02/02/2021]

- [29] ATC (2021). "Flutter Certified Application Developer". Recurso digital <https://androidatc.com/pages/Eng/Flutter-Certified-Application-Developer>. [Accedido el 02/02/2021]
- [30] Pub.dev (2021). "PostgreSQL". Recurso digital <https://pub.dev/packages?q=postgresql>. [Accedido el 02/02/2021]
- [31] Flutter (2021). "Add Flutter to existing app". Recurso digital <https://flutter.dev/docs/development/add-to-app>. [Accedido el 02/02/2021]
- [32] Nubank (2021). "Sobre nós". Recurso digital <https://nubank.com.br/sobre-nos/>. [Accedido el 02/02/2021]
- [33] Medium (2019). "Porquê nós achamos que Flutter vai nos ajudar a escalar o desenvolvimento mobile no Nubank". Recurso digital <https://medium.com/flutter-comunidade-br/porqu%C3%AA-n%C3%B3s-achamos-que-flutter-vai-nos-ajudar-a-escalar-o-desenvolvimento-mobile-no-nubank-95d07b4554d7>. [Accedido el 02/02/2021]
- [34] Flutter (2021). "Null Safety in Flutter". Recurso digital <https://flutter.dev/docs/null-safety>. [Accedido el 03/02/2021]
- [35] Dart (2021). "Sound null safety". Recurso digital <https://dart.dev/null-safety>. [Accedido el 03/02/2021]
- [36] Api Flutter (2021). "AnimatedContainer class". Recurso digital <https://api.flutter.dev/flutter/widgets/AnimatedContainer-class.html>. [Accedido el 03/03/2021]
- [37] Flutter Central (2019). "AnimatedCrossFade Widget in Flutter". Recurso digital <https://fluttercentral.com/Articles/Post/1113/AnimatedCrossFade Widget in Flutter>. [Accedido el 03/03/2021]
- [38] Flutter Widget Livebook (2021). "AnimatedDefaultTextStyle". Recurso digital <https://flutter-widget.live/widgets/AnimatedDefaultTextStyle> [Accedido el 03/02/2021]
- [39] Flutter (2021). "Flutter widget index". Recurso digital <https://flutter.dev/docs/reference/widgets> [Accedido el 04/02/2021]
- [40] Flutter (2021). "Widget catalog". Recurso digital <https://flutter.dev/docs/development/ui/widgets>. [Accedido el 04/02/2021]
- [41] Medium (2020). "Flutter: GestureDetector in-depth". Recurso digital <https://medium.com/litslink-mobile-development/flutter-gesturedetector-86fc937aaf17>. [Accedido el 04/02/2021]
- [42] Dart (2021). "A tour of the Dart language". Recurso digital <https://dart.dev/guides/language/language-tour?ref=hackernoon.com>. [Accedido el 05/02/2021]

- [43] Pub.dev (2021). “Flutter Slidable”. Recurso digital https://pub.dev/packages/flutter_slidable. [Accedido el 10/03/2021]
- [44] SQLite (2021). “What Is SQLite?”. Recurso digital <https://www.sqlite.org/index.html>. [Accedido el 15/03/2021]
- [45] Google Developers (2021). “Announcing Flutter 2”. Recurso digital <https://developers.googleblog.com/2021/03/announcing-flutter-2.html>. [Accedido el 16/03/2021]

APÉNDICES

1. MANUAL PARA EL USUARIO

En ese apartado se puede encontrar el manual de usuario, mediante el cual el usuario puede aprender a utilizar la aplicación y así sacar mayor provecho y utilidad de la app.

1.1. Onboarding

Cuando se abre la aplicación por la primera vez nos aparece las pantallas de *onboarding*, que sirven para mostrar todo lo que la aplicación puede hacer de forma sencilla y directa; a modo de instruir el usuario sobre el alcance de la aplicación, conforme se puede ver en la ilustración 1.

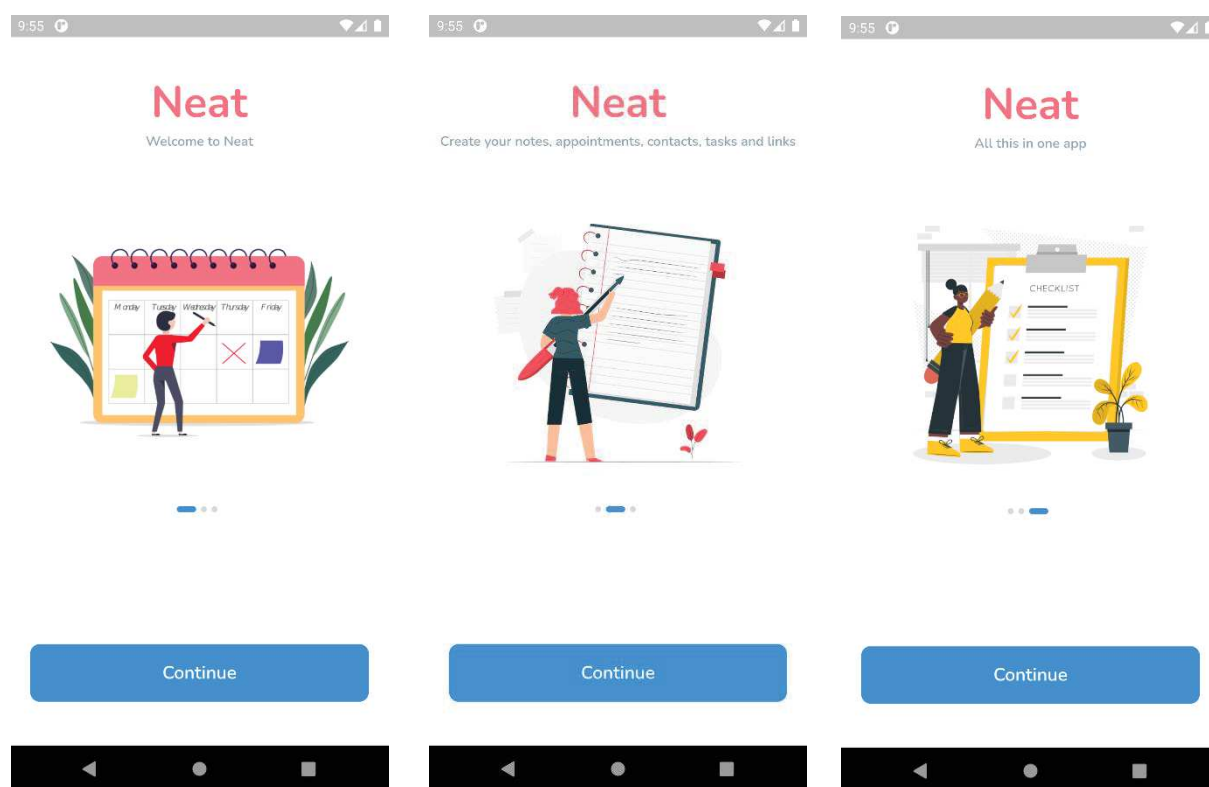


Ilustración 1. Pantallas de onboarding

Elaboración propia

Una vez que el usuario prosiga presionando el botón “*Continue*”, será redireccionado a la primera pantalla de la aplicación, a través de la cual el usuario ya tiene acceso a funcionalidades de la misma. En este caso, se muestra al usuario un

calendario que refleja todas las citas insertadas. Pero también se puede mirar las otras pestañas de la aplicación, que al ser elegida lleva el usuario a dicha funcionalidad. En la ilustración 2, las pestañas que dan acceso a las demás funcionalidades del sistema se puede observar en el rectángulo verde, mientras en el rectángulo rojo se observa la primera pantalla exhibida después de las pantallas de *onboarding*. También se puede observar en la misma ilustración (rectángulo naranja), que hay un icono que representa la acción de volver hacia atrás, o sea, volver a las pantallas de *onboarding*; es importante tener en cuenta que ese icono, así como las pantallas de *onboarding* solo están presentes durante la primera utilización de la aplicación.

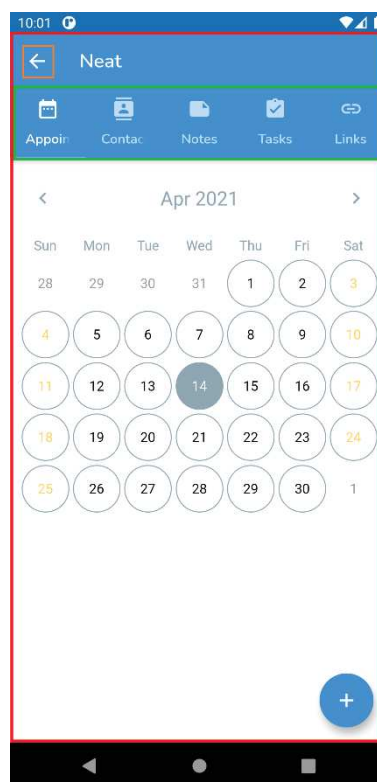


Ilustración 2. Pantalla tras pantallas de onboarding

Elaboración propia

1.2. Citas (*Appointments*)

La pantalla de Citas es la primera a la que el usuario tiene acceso tras las pantallas de *onboarding*. El calendario que se muestra, es una forma visual y útil que el usuario dispone para mirar todas sus citas añadidas. Esa pantalla ofrece todas las

funcionalidades necesarias para que el usuario pueda añadir, editar, visualizar y borrar una cita.

- Añadir una cita – para añadir una cita el usuario solo tiene que hacer clic sobre el botón presente en la parte inferior (con el icono de +) e insertar las informaciones referentes a su cita. Las informaciones solicitadas son: un título, una descripción, una fecha y una hora, siendo obligatorias solamente el título y la descripción. Por defecto, la aplicación elegirá la fecha del día actual como fecha para la cita, si desea cambiar, basta que haga clic en la fecha que saldrá un widget de calendario para que pueda elegir otra fecha. La ilustración 3, muestra el flujo mínimo necesario para añadir una cita.

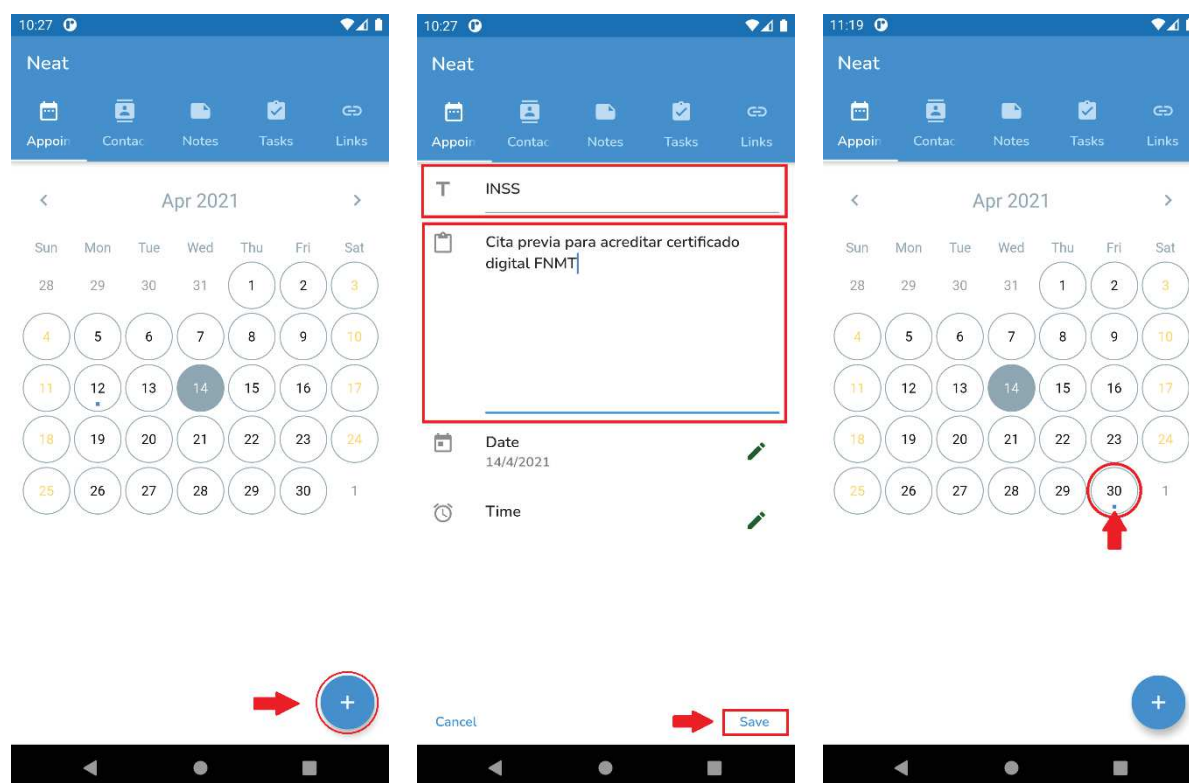


Ilustración 3. Pasos mínimos para añadir una cita

Elaboración propia

Adicionalmente, la ilustración 4 muestra los demás pasos necesarios (no obligatorios) en el caso de que sea necesario cambiar la fecha de la cita y/o añadir un horario para la misma.

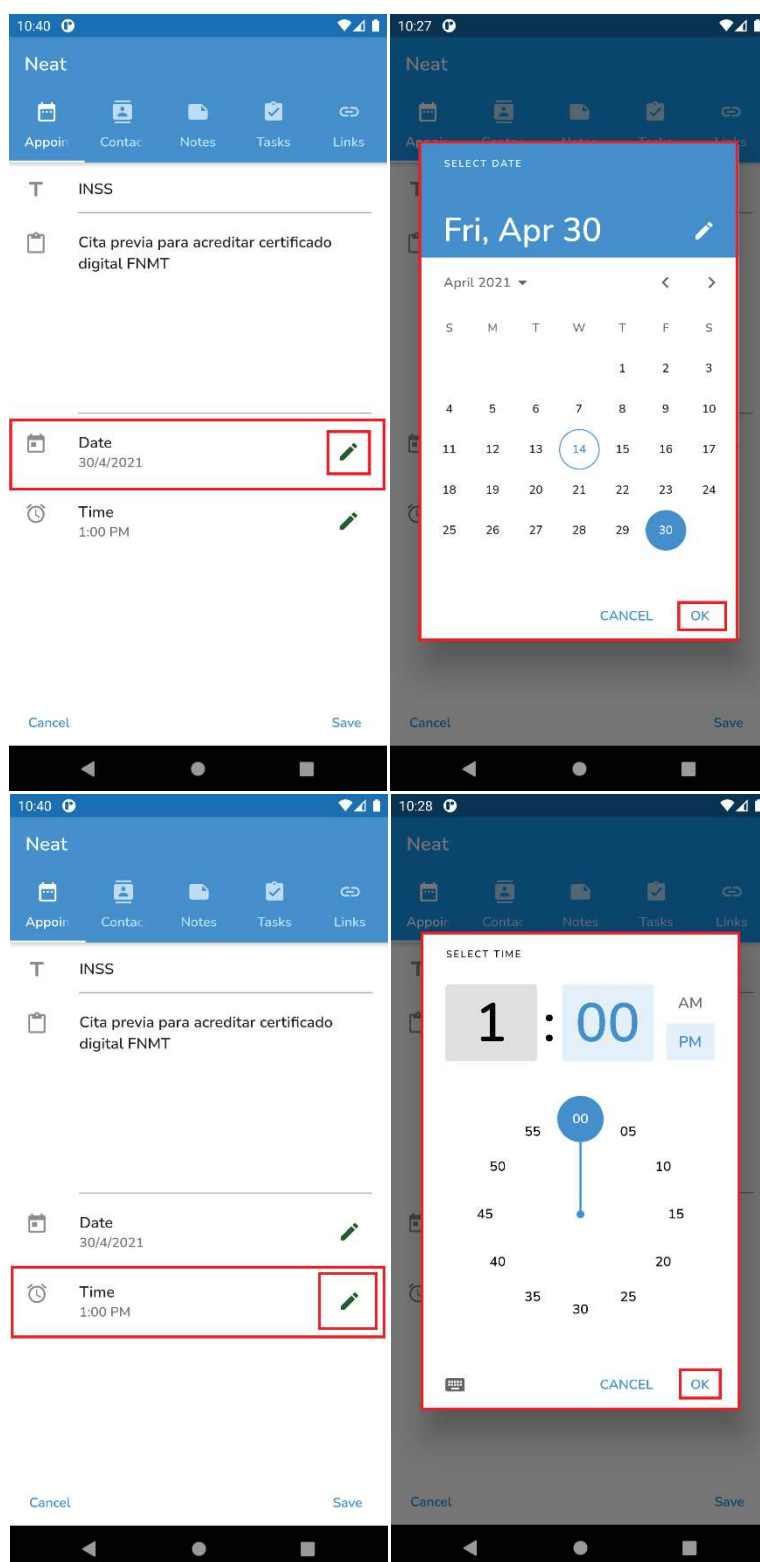


Ilustración 4. Pasos para cambiar fecha y/o hora

Elaboración propia

Después de haber insertado los datos y clicado el botón de guardar (save) mostrado, la cita ya está guardada y puede ser visualizada en el calendario, como se muestra en la ilustración 3.

- Editar una cita – Para editar una cita, basta que el usuario haga clic sobre un día que haya una cita ya añadida (representadas por un puntillo azul en el calendario debajo de los días, conforme se puede observar en la ilustración 3). Después de hacer clic, la aplicación exhibirá un listado con todas las citas del día, elija uno y haga clic sobre el mismo y así la aplicación abrirá el formulario con las informaciones anteriormente añadidas. Basta que el usuario edite las informaciones a su gusto y guarde las mismas. La ilustración 5 representa cómo es el flujo de edición de una cita.

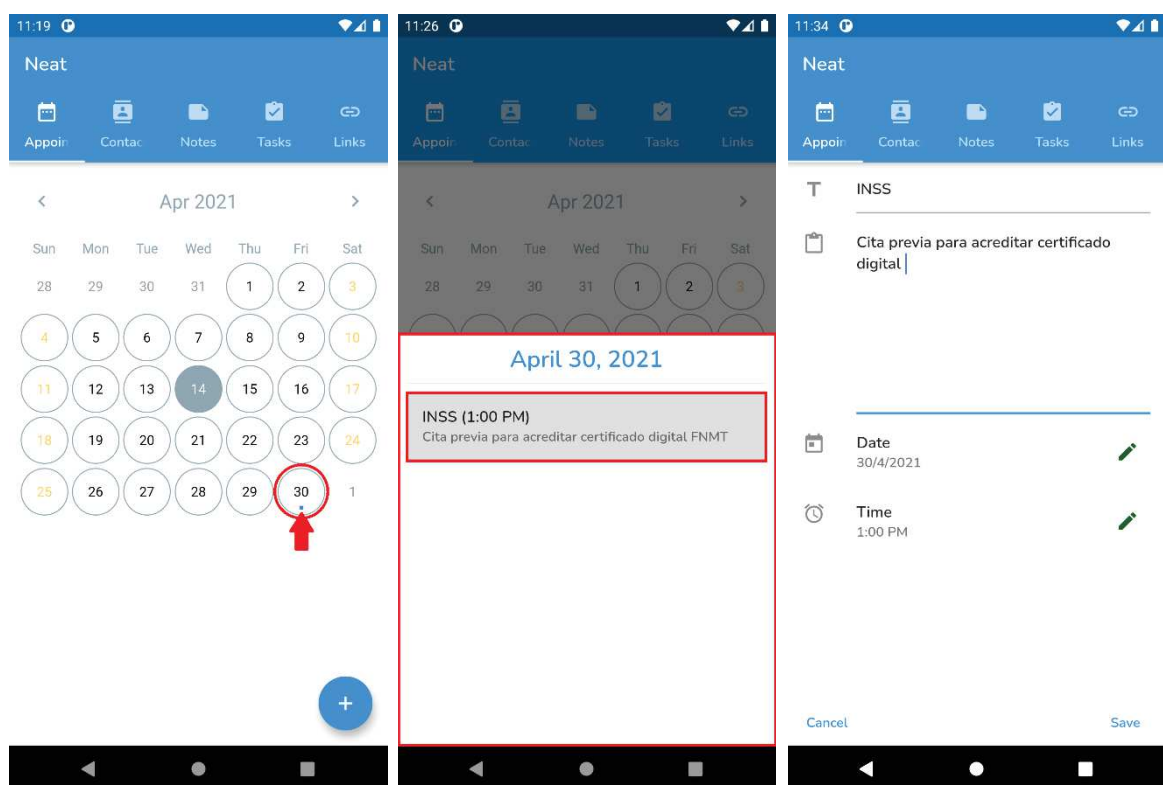


Ilustración 5. Pasos para editar una cita

Elaboración propia

- **Borrar una cita** – El proceso de borrar una cita es muy similar a editar una cita, con la diferencia de que ahora en vez de hacer clic sobre la cita, hay que deslizar la cita de la derecha hacia la izquierda, haciendo que aparezca el icono de borrar. Haciendo clic en este botón de borrar, la aplicación muestra un mensaje de confirmación, preguntando si el usuario está seguro que desea borrar dicha cita. La ilustración 6 muestra el flujo de borrar una cita.

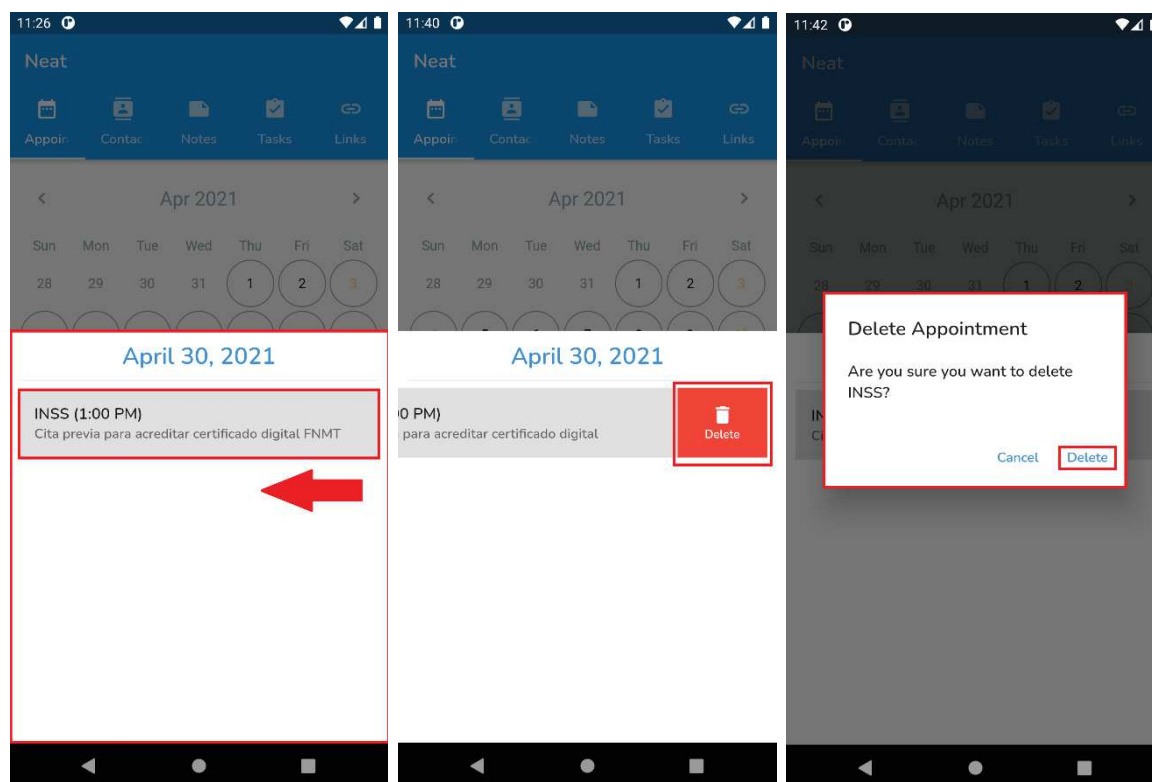


Ilustración 6. Pasos para borrar una cita

Elaboración propia

- **Visualizar citas** – La parte de visualización de citas ya está presente en la ilustración 3, bastando que el usuario haga clic sobre un día que haya citas inseridas.

1.3. Contactos (*Contacts*)

Esa funcionalidad permite crear, listar, editar y borrar contactos. Se puede acceder desde la pestaña de Contactos. Al hacer clic sobre la pestaña Contactos, la

pantalla exhibida muestra todos los contactos ya añadidos por el usuario. La ilustración 7 muestra la primera pantalla al acceder la pestaña de contactos.

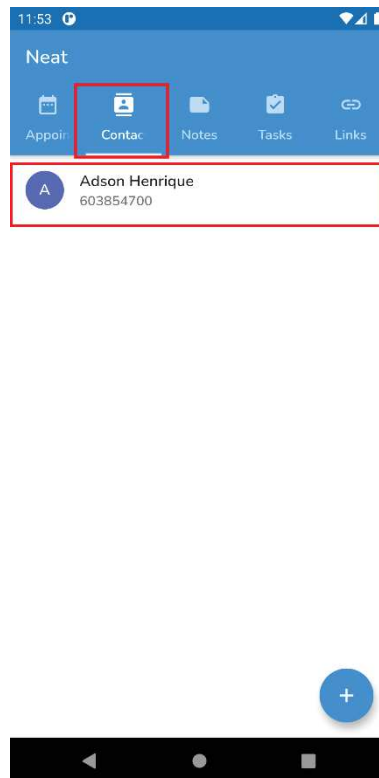


Ilustración 7. Primera pantalla al acceder a contactos

Elaboración propia

- Añadir un contacto – El proceso de añadir un contacto es muy similar al proceso de añadir una cita, con la salvedad de los datos necesarios para la inserción. Los datos obligatorios para la creación de un contacto son: el nombre y el correo electrónico. La ilustración 8 muestra los pasos mínimos necesarios para la creación de un contacto.

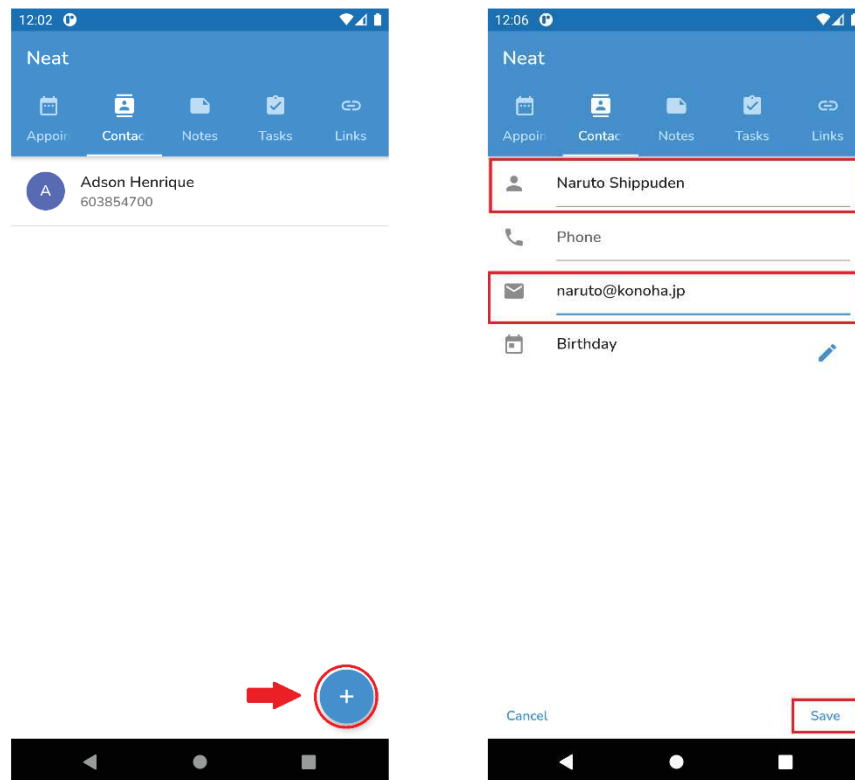


Ilustración 8. Pasos mínimos necesarios para la creación de un contacto

Elaboración propia

- Editar un contacto – Para editar un contacto ya previamente creado es necesario que el usuario acceda a pestaña de contactos y haga clic sobre el contacto que desea editar en el listado de contactos. La aplicación exhibirá el formulario con las informaciones, para que el usuario altere las informaciones que desea y guarde los cambios. La ilustración 9 muestra el flujo para la edición de un contacto.

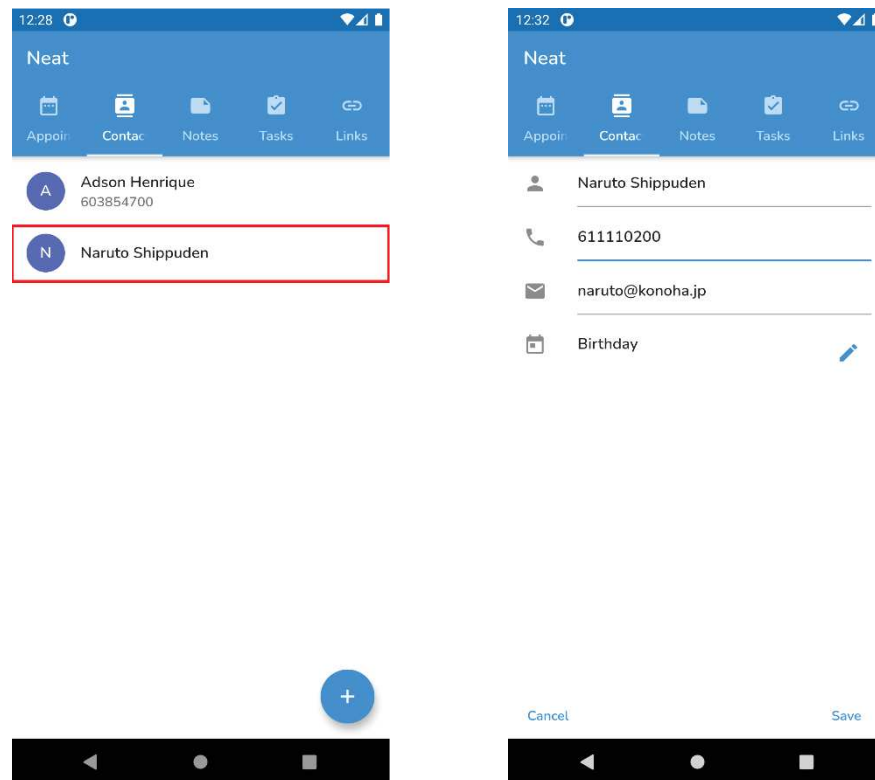


Ilustración 9. Pasos necesarios para la edición de un contacto

Elaboración propia

- **Borrar un contacto** – El proceso de borrar un contacto es similar al de editar; nuevamente, en vez de hacer clic en el contacto, es necesario que el usuario deslice el contacto de la derecha hacia la izquierda, haciendo visible el icono de borrar. Haciendo clic en ese icono, la aplicación exhibirá un mensaje preguntando si el usuario está seguro de la acción. La ilustración 10 muestra el flujo necesario para borrar un contacto.

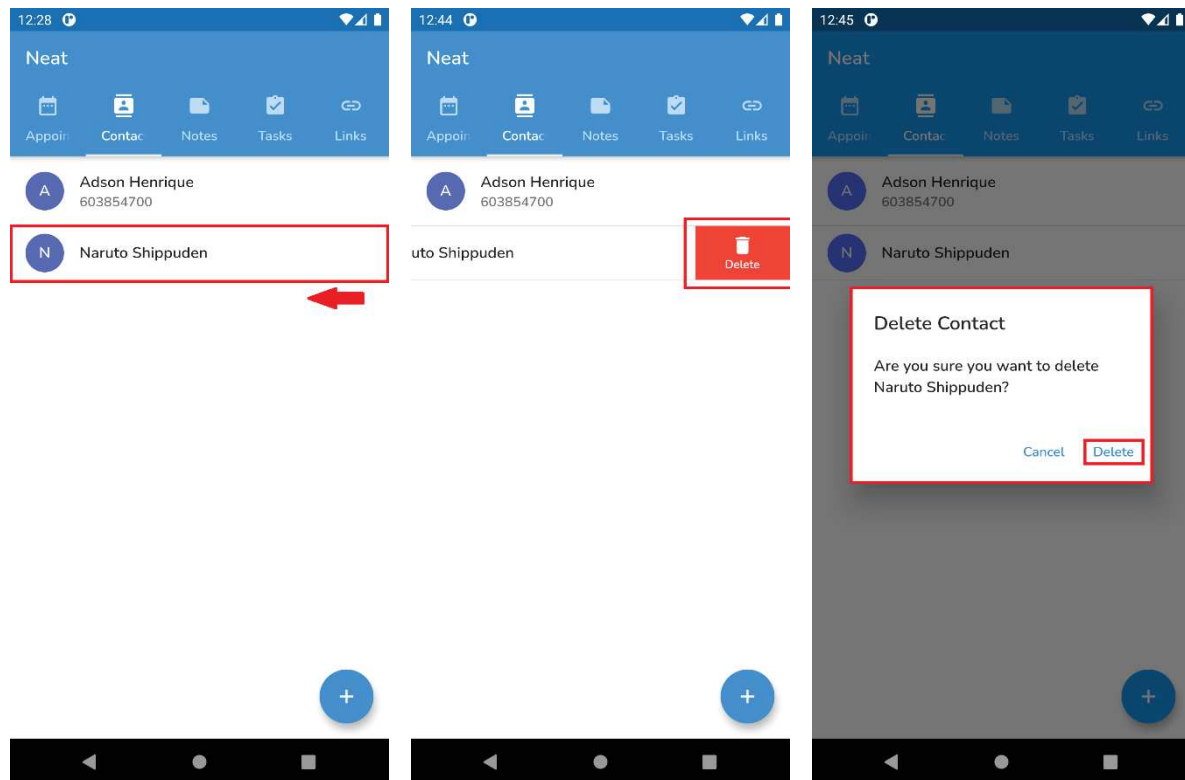


Ilustración 10. Pasos necesarios para borrar un contacto

Elaboración propia

- Listar contactos – La parte de visualización de contactos ya está presente en la ilustración 7, siendo suficiente que el usuario haga clic en la pestaña de contactos.

1.4. Notas (*Notes*)

Esa funcionalidad permite crear, listar, editar y borrar notas. Se puede acceder desde la pestaña de Notas. Al hacer clic sobre la pestaña Notas, la pantalla exhibida muestra todas las notas ya añadidas por el usuario. La ilustración 11 muestra la primera pantalla al acceder la pestaña de notas.

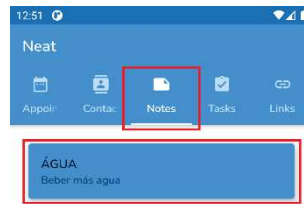


Ilustración 11. Pantalla inicial de la pestaña notas

Elaboración propia

- Añadir una nota – El proceso de añadir una nota es parecido a los anteriores. Haga clic en botón de añadir una nota, inserta los datos (siendo los datos obligatorios el título y la descripción, además de poder elegir un color para la nota) y guardar los cambios hechos. La ilustración 12 muestra el flujo de añadir una nota.

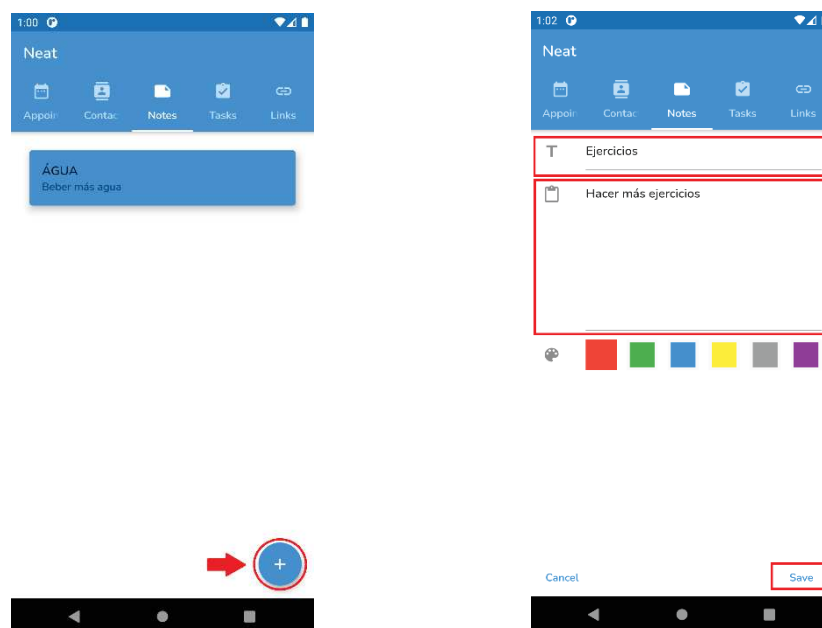


Ilustración 12. Pasos necesarios para añadir una nota

Elaboración propia

- Editar una nota – para editar una nota ya previamente creada es necesario que el usuario acceda a pestaña de notas y haga clic sobre la nota deseada, en el listado de notas. La aplicación exhibirá el formulario con las informaciones, para que el usuario altere las informaciones que desea y guarde los cambios. La ilustración 13 muestra el flujo para la edición de una nota.

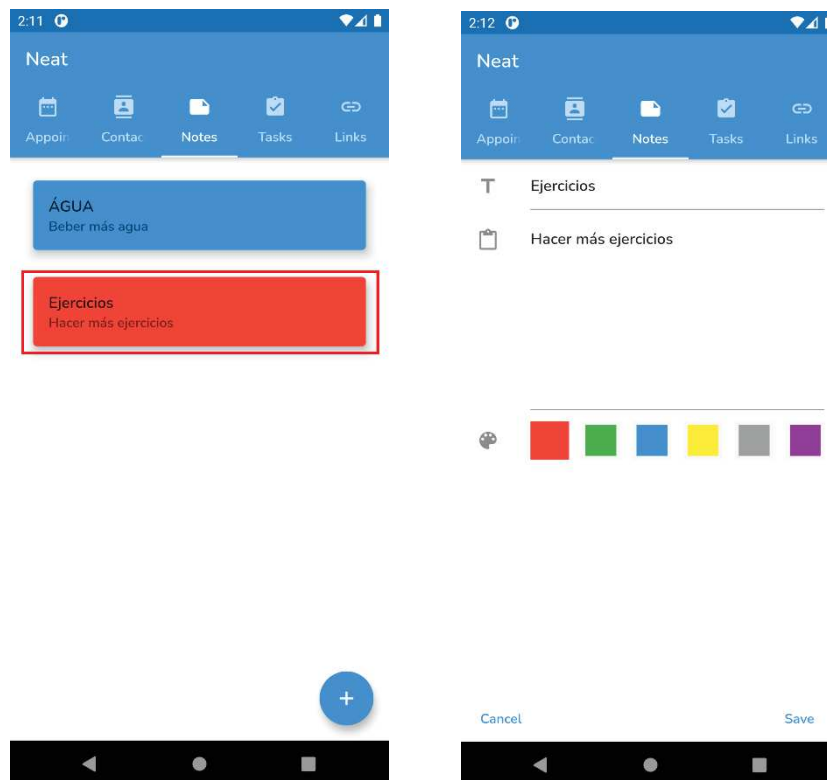


Ilustración 13. Pasos necesarios para editar una nota

Elaboración propia

- Borrar una nota - El proceso de borrar una nota es similar al de editar, a la diferencia que ahora en vez de hacer clic en la nota, es necesario que el usuario deslice la nota de la derecha hacia la izquierda, haciendo con que sea visible el icono de borrar. Haciendo clic en ese icono, la aplicación exhibirá un mensaje preguntando si el usuario está seguro de la acción. La ilustración 14 muestra el flujo necesario para borrar una nota.

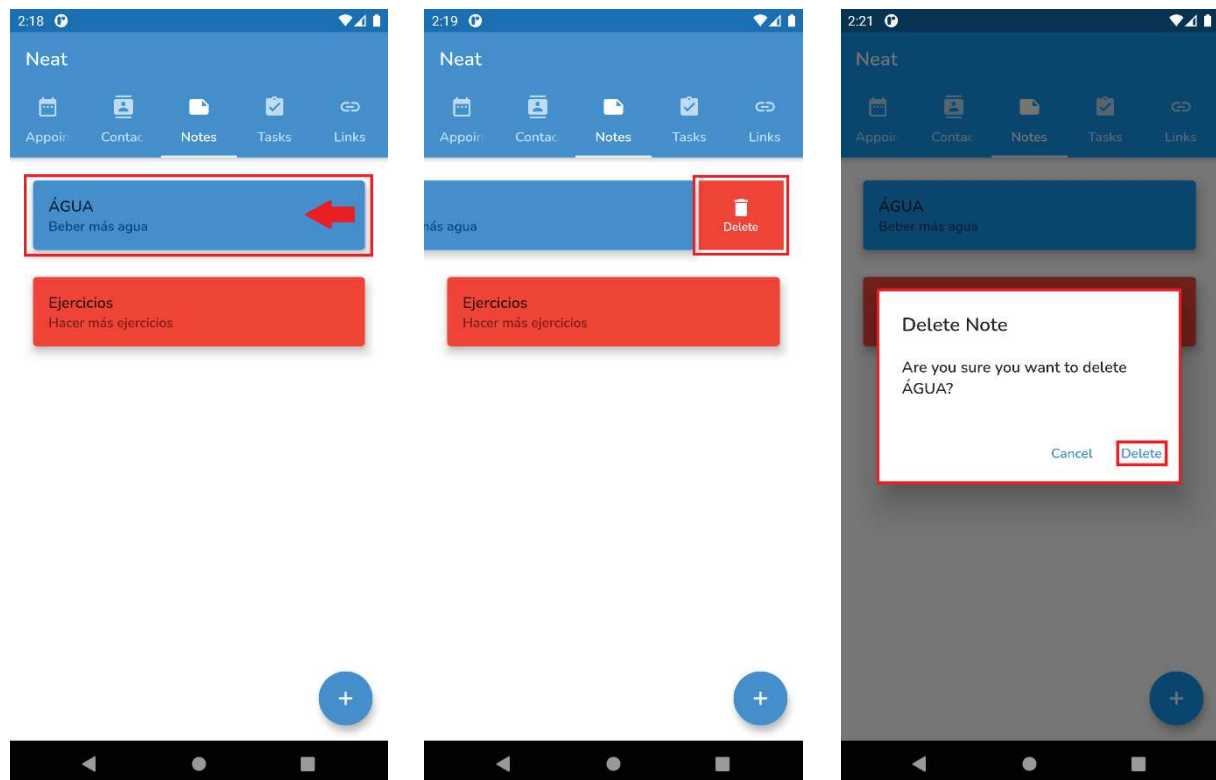


Ilustración 14. Pasos necesarios para borrar una nota

Elaboración propia

- Visualizar una nota – La parte de visualización de notas ya está presente en la ilustración 11, siendo suficiente que el usuario haga clic en la pestaña de notas.

1.5. Tareas (*Tasks*)

Esa funcionalidad permite crear, listar, editar y borrar tareas. A la diferencia de las demás, esta también te permite marcar si una tarea ya fue hecha o no. Se puede acceder desde la pestaña de Tareas. Al hacer clic sobre la pestaña Tareas, la pantalla exhibida muestra un listado con todas las tareas ya añadidas por el usuario. La ilustración 15 muestra la primera pantalla al acceder la pestaña de tareas y también la opción de marcar tareas completadas.

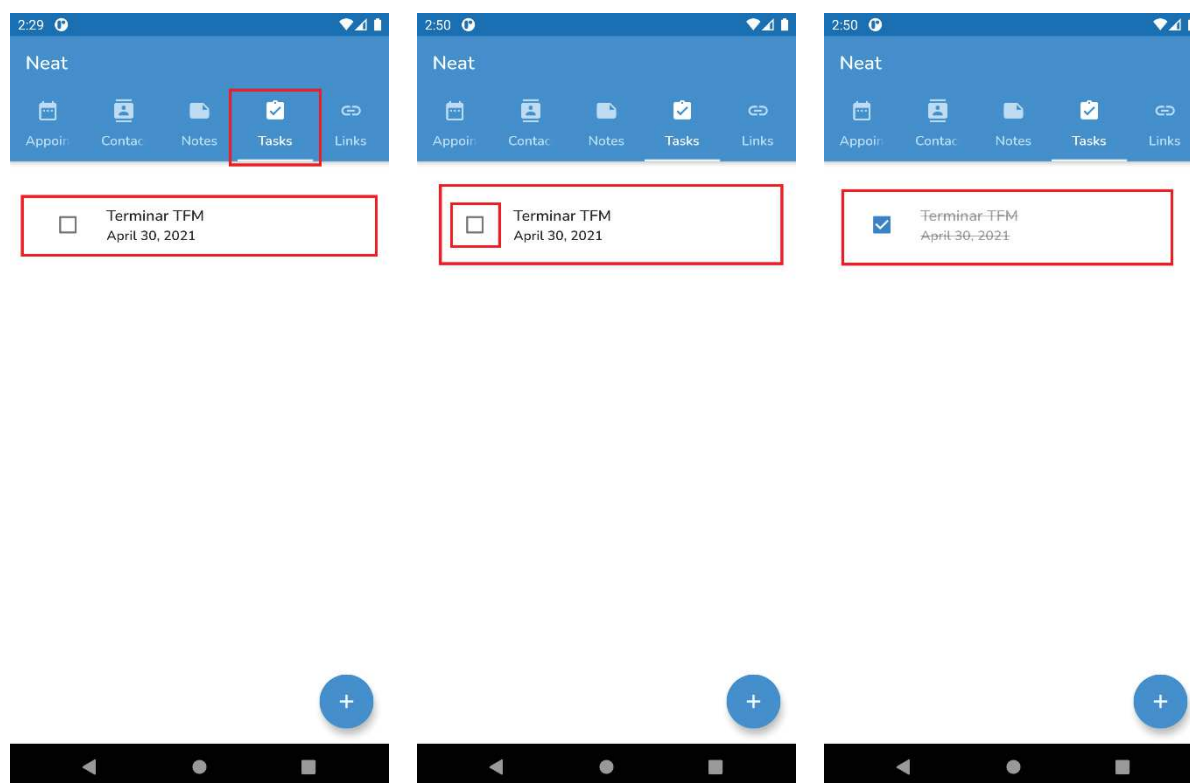


Ilustración 15. Primera pantalla al acceder la pestaña de tareas

Elaboración propia

- Añadir tarea – El proceso de añadir una tarea es parecido con los anteriores. Hacer clic en botón de añadir una nota, insertar los datos (siendo la descripción el único dato obligatorio) y guardar los cambios hechos. La ilustración 16 muestra el flujo de añadir una tarea.

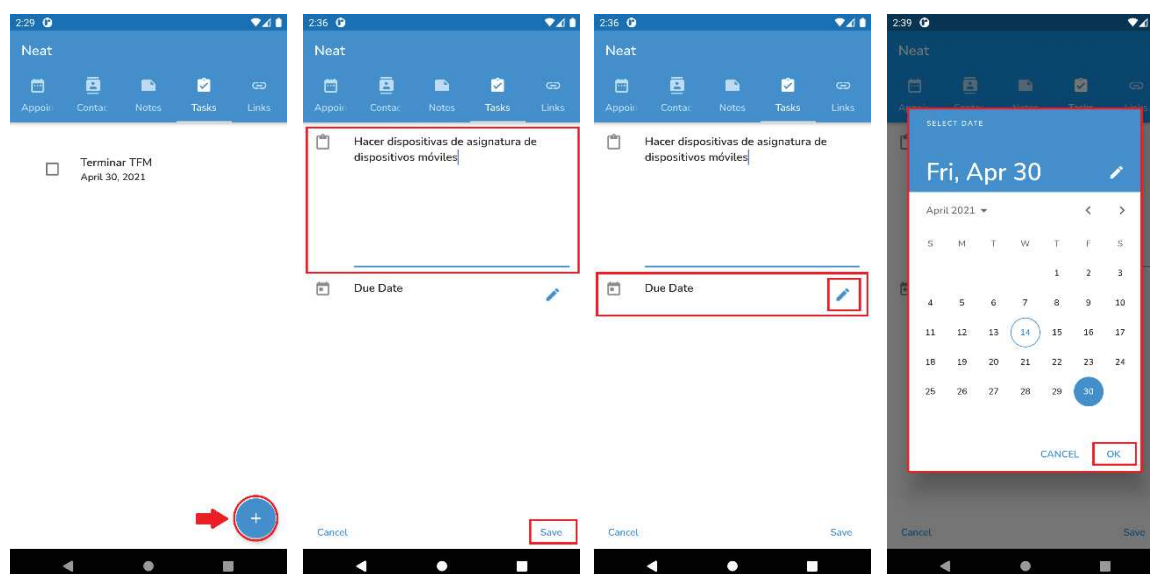


Ilustración 16. Pasos necesarios para creación de una tarea

Elaboración propia

- Editar tarea – para editar una tarea ya previamente creada es necesario que el usuario acceda a pestaña de tareas y haga clic sobre la tarea deseada, en el listado de tareas. La aplicación exhibirá el formulario con las informaciones, para que el usuario altere las informaciones que desea y guarde los cambios. La ilustración 17 muestra el flujo para la edición de una tarea.

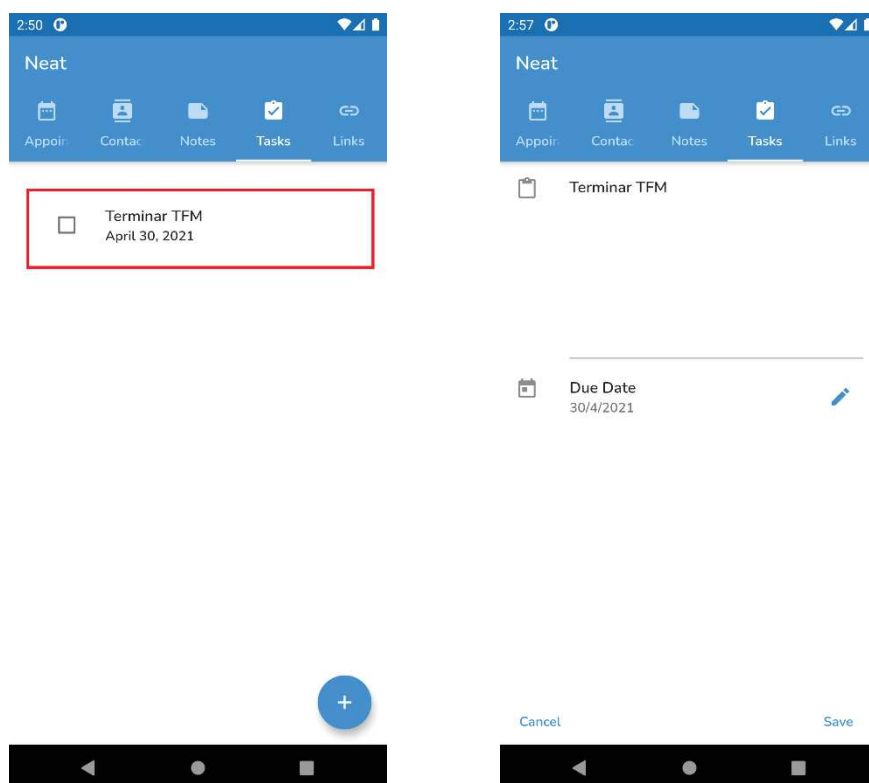


Ilustración 17. Pasos necesarios para la edición de una tarea

Elaboración propia

- Borrar tarea – El proceso de borrar una tarea es similar al de editar, a la diferencia que ahora en vez de hacer clic en la tarea, es necesario que el usuario deslice la tarea de la derecha hacia la izquierda, haciendo con que aparezca el icono de borrar. Haciendo clic en ese icono, la aplicación exhibirá un mensaje de confirmación preguntando si el usuario está seguro de la acción. La ilustración 18 muestra el flujo necesario para borrar una tarea.

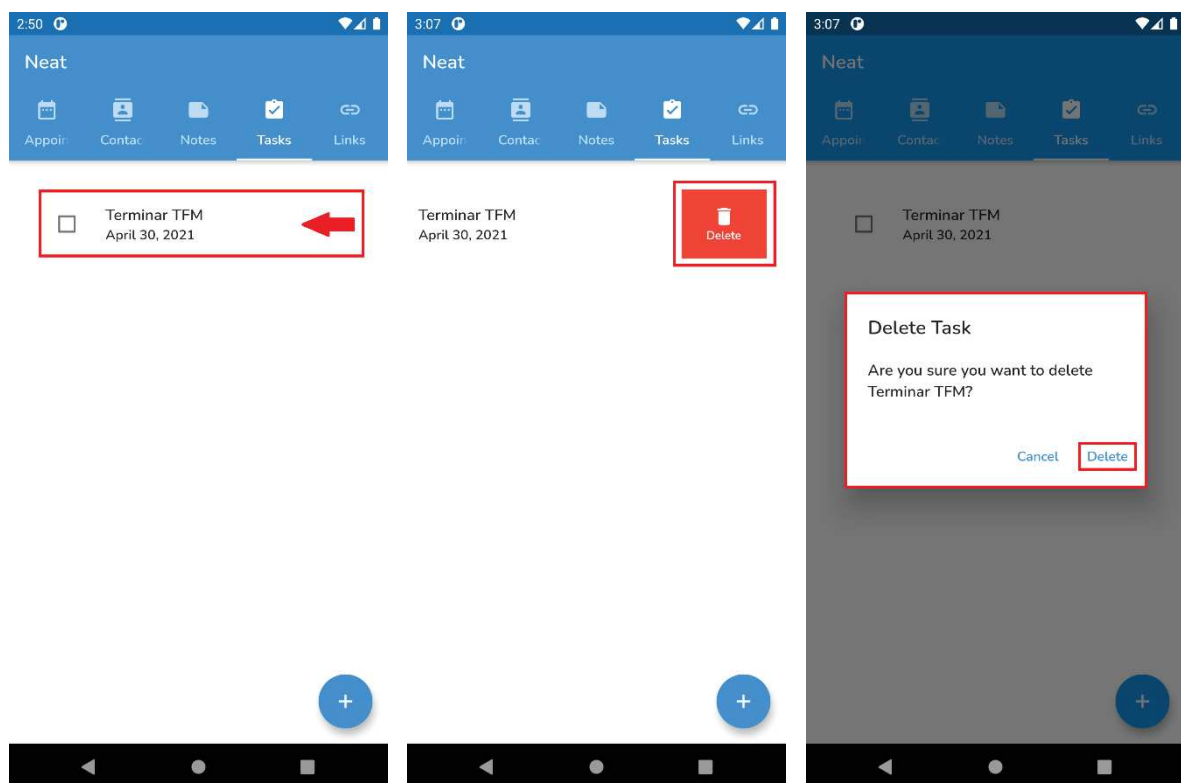


Ilustración 18. Pasos necesarios para borrar una tarea

Elaboración propia

- Visualizar tarea – La parte de visualización de tareas ya está presente en la ilustración 15, siendo suficiente que el usuario haga clic en la pestaña de tareas.

1.6. Enlaces (*Links*)

Esa funcionalidad permite crear, listar, editar y borrar enlaces. Una diferencia con respecto a las demás, es que esa funcionalidad solo te permite añadir enlaces por medio de QRCode, es decir, el usuario no puede añadir manualmente un enlace, sino que hay que escanearlo por medio de la cámara. Además de eso, la aplicación también te permite acceder a los enlaces sin la necesidad de usar un navegador de internet externo, siendo necesario solamente el usuario hacer doble clic sobre el enlace ya creado.

Se puede acceder desde la pestaña de Enlaces. Al hacer clic sobre la pestaña Enlaces, la pantalla exhibida muestra un listado con todos los enlaces ya añadidos

por el usuario. La ilustración 19 muestra la primera pantalla al acceder la pestaña de enlaces y como es el funcionamiento de la abertura del enlace.

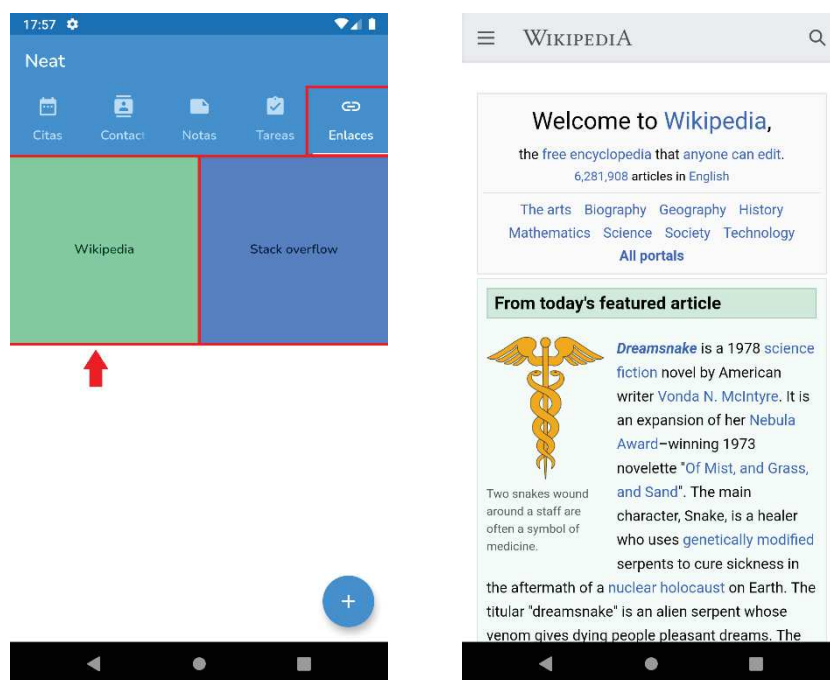


Ilustración 19. Pantallas de la pestaña Enlaces

Elaboración propia

- Añadir enlace – el proceso de añadir un enlace es parecido con los anteriores. Haga clic en botón de añadir, inserte los datos (siendo la descripción el único dato obligatorio) y guardar los cambios hechos. Para insertar el enlace, haga clic en el icono de la cámara, la aplicación solicitará permisos para acceder a la cámara y desde ahí el único que hay que hacer es apuntar a cámara hacia algún QRCode. La ilustración 20 muestra el flujo de añadir un enlace.

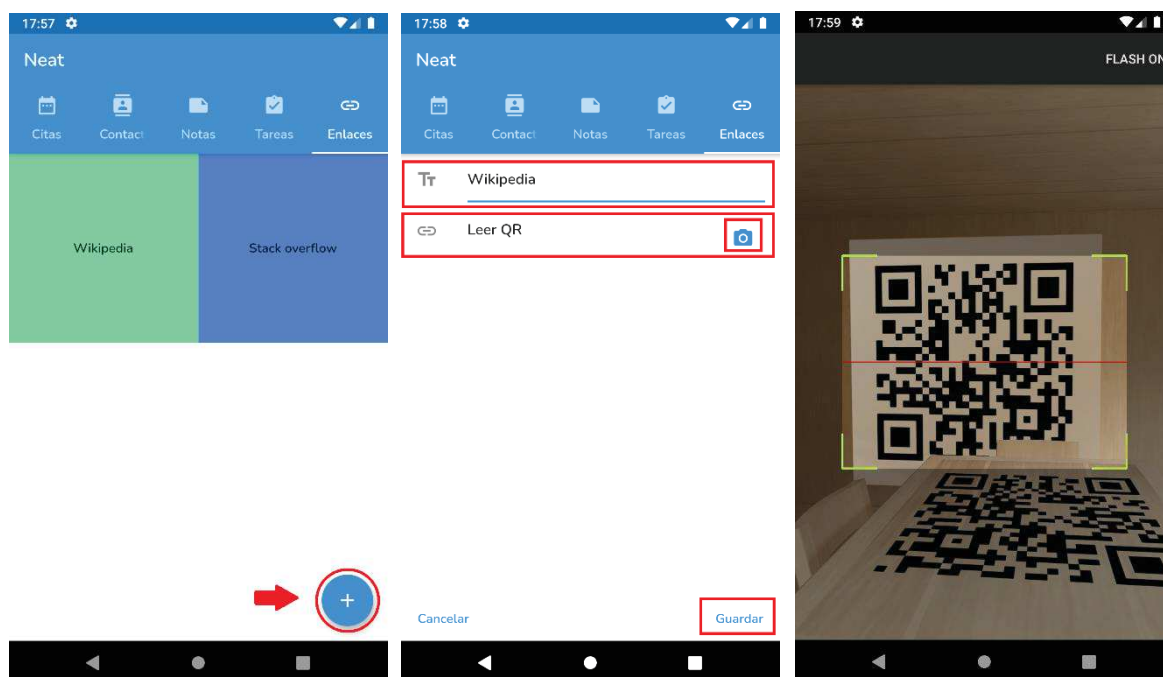


Ilustración 20. Pasos necesarios para crear un enlace

Elaboración propia

- Editar enlace – para editar un enlace ya previamente creado es necesario que el usuario haga clic sobre el enlace deseado, en el listado de enlaces. La aplicación exhibirá el formulario con las informaciones, para que el usuario altere las informaciones que desea y guarde los cambios. La ilustración 21 muestra el flujo para la edición de un enlace.

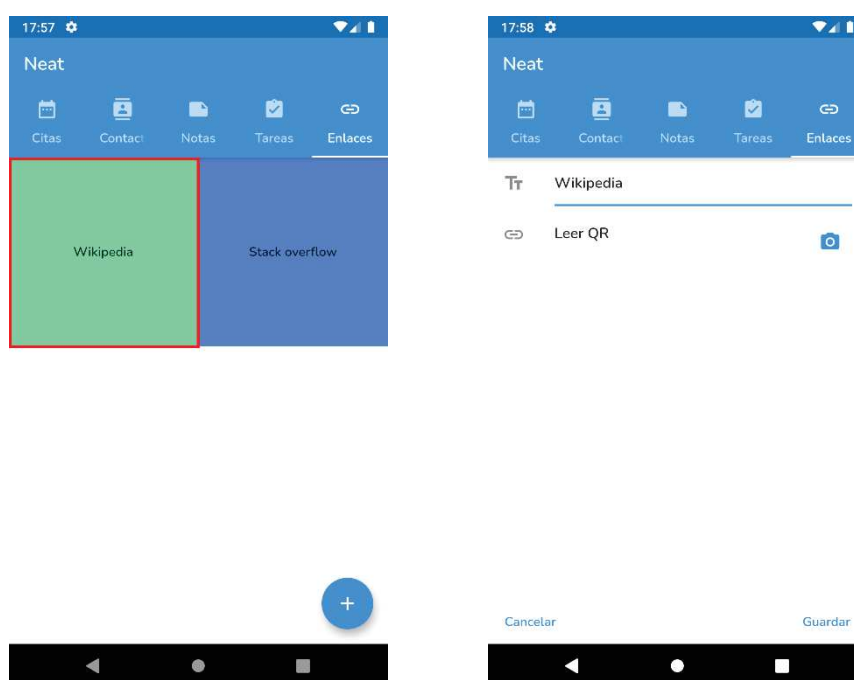


Ilustración 21. Pasos necesarios para editar un enlace

Elaboración propia

- **Borrar enlace** – El proceso de borrar un enlace, a la diferencia de editar que, en vez de hacer clic en el enlace, es necesario que el usuario haga un clic y mantenga el clic presionado hasta que la aplicación exhiba un mensaje preguntando si el usuario está seguro de la acción. La ilustración 22, muestra el flujo necesario para borrar un enlace.

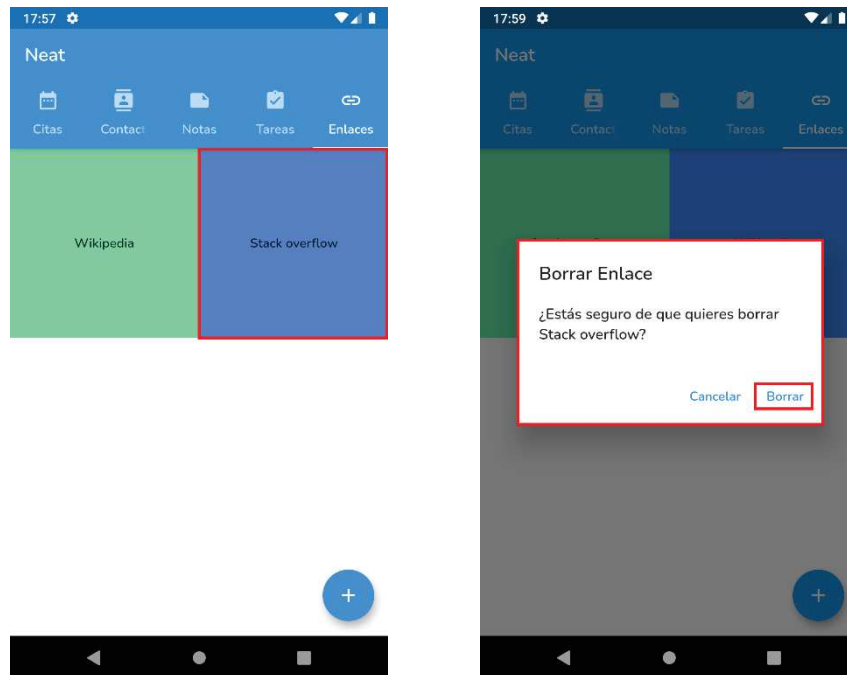


Ilustración 22. Pasos necesarios para borrar un enlace

Elaboración propia

- **Visualizar enlace** – La parte de visualización de enlaces ya está presente en la ilustración 19, siendo suficiente que el usuario haga clic en la pestaña de enlaces.