UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO

GRADO EN INGENIERÍA DEL SOFTWARE

Ingeniería de Software aplicada al desarrollo de aplicaciones multiplataforma en equipos pequeños con Flutter

Autor: Francisco Javier Morales Sánchez de Prados

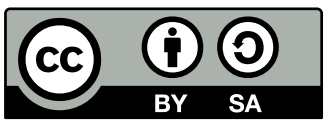
Director: Guillermo Iglesias Hernández

Madrid, July 2023

Francisco Javier Morales Sánchez de Prados

Ingeniería de Software aplicada al desarrollo de aplicaciones multiplataforma en equipos pequeños con Flutter
Proyecto Fin de Grado, Tuesday 11º July, 2023
Director: Guillermo Iglesias Hernández

E.T.S. de Ingeniería de Sistemas Informáticos
Campus Sur UPM, Carretera de Valencia (A-3), km. 7
28031, Madrid, España



Yo, Francisco Javier Morales Sánchez de Prados, estudiante de la titulación Grado en Ingeniería del Software de la de E.T.S. de Ingeniería de Sistemas Informáticos de la Universidad Politécnica de Madrid, como autor del Proyecto Fin de Grado titulado:

Ingeniería de Software aplicada al desarrollo de aplicaciones multiplataforma en equipos pequeños con Flutter

DECLARO QUE

Este proyecto es una obra original y que todas las fuentes utilizadas para su realización han sido debidamente citadas en el mismo. Del mismo modo, asumo frente a la Universidad cualquier responsabilidad que pudiera derivarse de la autoría o falta de originalidad del contenido de la memoria presentada de conformidad con el ordenamiento jurídico vigente.

Madrid, a Tuesday 1 I^{o} July, 2023

Fdo.: Francisco Javier Morales Sánchez de Prados
Autor del Proyecto Fin de Grado

Resumen

El presente proyecto de fin de grado explora el uso e implementación de procesos de ingeniería de software en el desarrollo del front-end de aplicaciones con estado almacenado, mostrado y transferido mediante el uso de APIs REST para crear Wemon.

Wemon es una Red Social en el contexto de la Sociedad de la Información desarrollada mediante el uso de el framework Flutter.

En este TFG exploraremos los métodos y patrones seguidos para desarrollar las distintas pantallas de la aplicación partiendo de un diseño gráfico conceptual de la misma, todo ello en el contexto de un equipo de desarrollo y diseño de 6 a 8 integrantes.

Para implementar las distintas pantallas, primeramente, se analiza y divide su diseño y funcionalidades esperadas en historias de usuario, que serán refinadas y divididas en subtareas asignadas a y desarrolladas por cada uno de los equipos de desarrollo, siendo estos dos equipos los de front-end y back-end.

Abstract

This final degree project explores the use and implementation of software engineering processes in the development of the front-end of applications with stored, displayed, and transferred state using REST APIs to create the Wemon application, a Social Network in the context of the Information Society, using the Flutter framework, as well as the methods and patterns followed to develop the different screens of the application based on a conceptual graphic design of the same, all within the context of a development and design team of 6 to 8 members.

To implement the different screens, firstly, their design and expected functionalities are analyzed and divided into user stories, which will be refined and divided into subtasks assigned to and developed by each of the development teams, namely the front-end and back-end teams.

Agradecimientos

A mis padres, Encarna y Paco, por apoyarme desde siempre y por traerme a este mundo del que soy, seré y serán siempre parte. Gracias por tener paciencia conmigo pese a todo.

A mi hermana Blanca, por ser lo suficientemente pesada como para convencerme de que terminase bachillerato, a pesar de que no tuviese ganas ni fuerzas.

A mis tíos Espe, Jesús y Cesar y mi prima Lucía, por ser como sois y tratarme como lo hacéis. Os quiero.

A mi perro Turrón, por alegrarme desde siempre. Sé que no me entiendes cuando te hablo, pero me miras y sé que todo va bien.

A Edu, gracias por todo, por las risas y por todo lo que nos ha unido estos cuatro años y lo seguirá haciendo. Confía siempre en ti como yo lo hago.

A Guille, mi tutor del TFG, por tener la paciencia que has tenido para leerte todo este chorro de palabras, y acogerme como tutelado en un momento en el que estaba más perdido que enfocado.

A todo el que me haya acompañado en la carrera, en la universidad y en la vida, incluido a quien lea esto. Estés donde estés sigue adelante y no te desmotives; trabaja, estudia y sobre todo disfruta de los que te rodean.

A todos, gracias por el apoyo y sed muy felices. :)

Índice general

Índice de figuras

1. Introducción

Los teléfonos móviles inteligentes, los ordenadores y la informática en general se ha convertido en un componente vital y casi omnipresente en nuestras vidas, y con ellos las aplicaciones que nos permiten sacarles el máximo partido. No es poco común por lo tanto, encontrar buenas ideas por parte de estudiantes de informática, profesionales y aficionados para perfeccionar esta experiencia.

Sin embargo, materializar las ideas puede ser algo complejo que en ocasiones puede parecer una labor titánica.

El presente PFG pretende explorar la aplicación práctica de conceptos de la ingeniería de software como son las metodologías ágiles, así como procesos de organización del trabajo y patrones de implementación, para servir como guía a aquellos que deseen implementar sus ideas de desarrollo de aplicaciones.

El ejemplo que se seguirá durante todo el PFG es Wemon, una red social orientada a planes en cuyo desarrollo llevo trabajando más de dos años y medio siguiendo un rol de analista y desarrollador front-end dentro de la estructura organizativa del equipo.

Este equipo está compuesto por dos desarrolladores del front-end, dos del back-end, una diseñadora gráfica y dos personas encargadas del marketing y gestión del lado empresarial del proyecto,

Primeramente, se explicará el estado del arte del desarrollo de aplicaciones multiplataforma, para posteriormente explicar de forma resumida conceptos clave para la organización del desarrollo como son las metodologías ágiles y conceptos básicos del desarrollo como son ciertas pinceladas de diseño gráfico, disciplina vital de entender mínimamente para una buena comunicación con el equipo de diseño.

Seguidamente, se explicará el proceso seguido desde que se recibe un diseño, hasta que este está

listo para producción.

Adicionalmente, se explicarán los procesos seguidos para la modificación de una pantalla ya puesta en producción.

Durante estas dos últimas fases, se hará hincapié en la gestión del cambio.

Todo esto se explora desde el punto de vista de un equipo de desarrollo pequeño geográficamente distribuido, con los retos y restricciones que ello conlleva.

1.1 Motivación

En el ámbito profesional, la elección de este PFG viene motivada por la falta de ejemplos prácticos y accesibles de aplicación real de metodologías ágiles en equipos pequeños en el desarrollo de aplicaciones multiplataforma, y en concreto por la falta de todo lo mencionado en el ámbito de Flutter.

Esta falta de información dificulta la entrada al mercado de potenciales desarrolladores en etapas tempranas de la vida profesional de un ingeniero de software como lo es la etapa universitaria, donde es muy probable que un estudiante se junte con sus compañeros de carrera con una idea, pero sin saber cómo enfocarla y cómo proceder de una forma productiva.

Como ejemplo de un producto real desarrollado utilizando los métodos explorados por este PFG presento Wemon, una aplicación desarrollada por un equipo pequeño de desarrollo de entre 4 y 6 personas, con otras 2 personas adicionales para diseño de UI y UX.

1.2 Objetivos

En el ámbito personal, he decidido enfocar mi PFG en el desarrollo de aplicaciones con frameworks multiplataforma, Flutter en concreto, por mis conocimientos del propio framework, así como mi convicción en la idea de que desarrollar productos útiles en equipo, con lo que ello conlleva en términos de planificación y ejecución, es el complemento ideal a la formación teórica y práctica en el ámbito académico. Los problemas y cuestiones inesperadas surgidas en estas fases de desarrollo, así como la retroalimentación proveniente de los usuarios al hacer uso de esta, me han dado una perspectiva más clara de lo que es el desarrollo real de soluciones en equipo con grados de experiencia variados.

- Aprender a convertir diseños en pantallas funcionales.
- Aprender y aplicar patrones reutilizables, mantenibles y sencillos de implementar para desarrollar pantallas funcionales con capacidad de representar estados y cambios entre estados.

2. Estado del Arte

En la actualidad, y gracias a la creciente necesidad de integrar software en todo tipo de soluciones, el desarrollo de aplicaciones multiplataforma se encuentra en auge. Con la introducción y adopción masiva en el mercado de conceptos como el diseño responsive, que hacen referencia a la necesidad de contenidos representados por pantalla para adaptarse a distintos tipos, tamaños y resoluciones de pantalla, esta realidad queda patente. Todas las empresas quieren tener una presencia en internet, e internet se encuentra presente en una amplia variedad de dispositivos, que vienen en formas y tamaños de todo tipo.

Un claro ejemplo de la necesidad para una empresa de estar presente en todo tipo de plataformas son las redes sociales. Instagram, WhatsApp, Twitter, Facebook y miles de redes sociales poseen no solo una versión móvil de sus aplicaciones, sino también una plataforma web y aplicaciones de escritorio que permiten a sus usuarios utilizarlas en todo tipo de circunstancias. En este mismo sentido, Wemon pretende incorporarse al mercado como una red social más, y por ello necesita disponer de la capacidad de expandir su terreno de juego de forma rápida y eficiente.

La capacidad de una misma base de código para ser transpilada y posteriormente ejecutada en diferentes sistemas con requisitos diferentes, como pueden ser Android, iOS, Windows, Mac, Linux, Web o sistemas integrados ahorra tiempo y recursos en desarrollo, y estandariza elementos visuales que, de otra forma, quedarían sujetos a la capacidad del equipo de desarrollo para intentar replicar los resultados obtenidos en un lenguaje y plataforma en el código a ejecutar en la plataforma siguiente.

2.1 Aplicaciones Multiplataforma

Como se ha expuesto anteriormente, las aplicaciones multiplataforma están presentes en nuestro día a día [1]. Un ejemplo muy conocido y usado por todos es Netflix, que dispone de una misma base de código para sus aplicaciones de escritorio, Android, iOS y Web.

El concepto tras el desarrollo usando frameworks multiplataforma es programar una vez, y dejar que el framework haga la conversión a las plataformas finales deseadas.

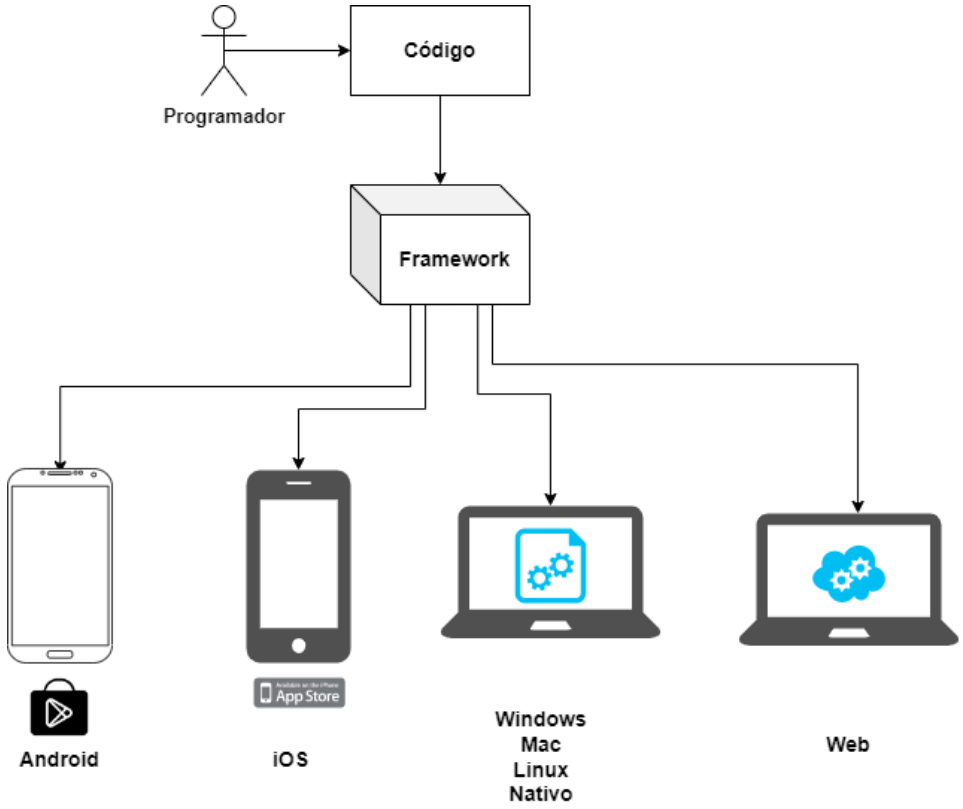


Figura 2.1: Una única base de código para varias plataformas

- **Código**: una única base de código para la aplicación.
- **Framework**: encargado de proporcionar las capacidad de la base de código única para ser transpilada al código específico de cada plataforma.
- **Android, iOS...**: las correspondientes plataformas en las que se desea publicar la aplicación.

Los principales exponentes a día de hoy de estos frameworks de desarrollo multiplataforma son React Native y Flutter, desarrollados respectivamente por Facebook (ahora Meta) y Google.

Entre las aplicaciones más destacables desarrolladas con React Native, destacan:

- **Instagram**: aunque utiliza muchas adaptaciones y código adicional nativo a cada plataforma por cuestiones de optimización, ciertas partes de la misma como su sección de compras

están hechas con React Native para permitir una experiencia similar en múltiples plataformas.

- **Facebook Ads Manager**: encargado de proporcionar las capacidad de la base de código única para ser transpilada al código específico de cada plataforma.

Por el lado de Flutter, tenemos:

- **Google Pay**: una de las principales carteras digitales utiliza Flutter para construir su aplicación móvil.
- **eBay**: en efecto, el segundo marketplace a nivel mundial utiliza Flutter para construir su aplicación móvil de manera rápida y sencilla.
- **iRobot**: esta plataforma de programación orientada a robótica implementa tanto tu aplicación móvil como su aplicación web utilizando una única base de código mediante el uso de Flutter.

En este concepto clave de "Una base de código, múltiples plataformas objetivo" 2.1 nos basamos para definir el fundamento de Wemon, así como se han basado en él miles de plataformas ya no solo de redes sociales, sino de todo tipo incluyendo finanzas, comercio etc. a lo largo del globo para desarrollar su filosofía tecnológica.

En cuestión de cifras, la necesidad de tener presencia digital en múltiples plataformas se hace evidente si tenemos en cuenta las tendencias de uso y descarga de aplicaciones a nivel mundial, que no han hecho más que crecer y casi se han duplicado en tan solo seis años.

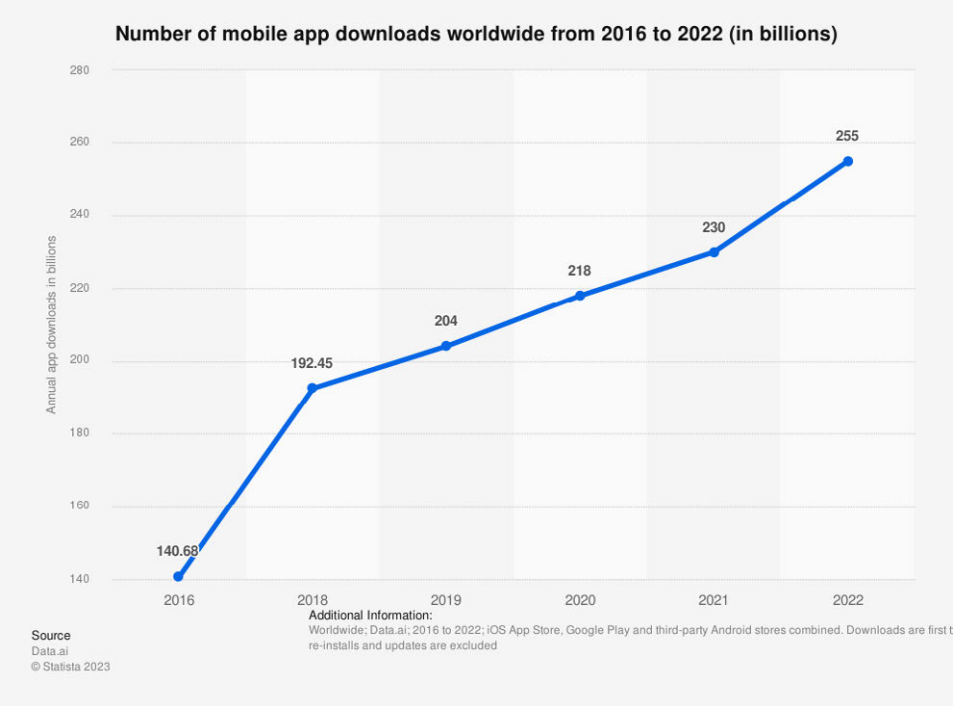


Figura 2.2: Numero de descargas de apps móviles a nivel mundial (2016-2022). Fuente [2]

Esto se traduce en un gran potencial de beneficio tanto para los creadores de aplicaciones como para los anunciantes, proveedores y negocios que se benefician de las aplicaciones para atraer nuevos clientes. A esto se le suma el valor añadido para los consumidores, que tendrán la posibilidad de elegir qué aplicaciones les son más útiles de entre la gran variedad de aplicaciones disponibles en el presente y en el futuro.

Además, el abaratamiento en los costes de producción y distribución de tecnología lo suficientemente potente como para ejecutar navegadores, así como la potente inversión en infraestructura de redes a escala global de la última mitad de siglo, han hecho que el número de usuarios con acceso a internet se haya quintuplicado en los últimos 15 años, superando la cifra de los cuatro mil seiscientos millones de usuarios, casi un cincuenta y nueve por ciento de la población mundial, y creciendo.

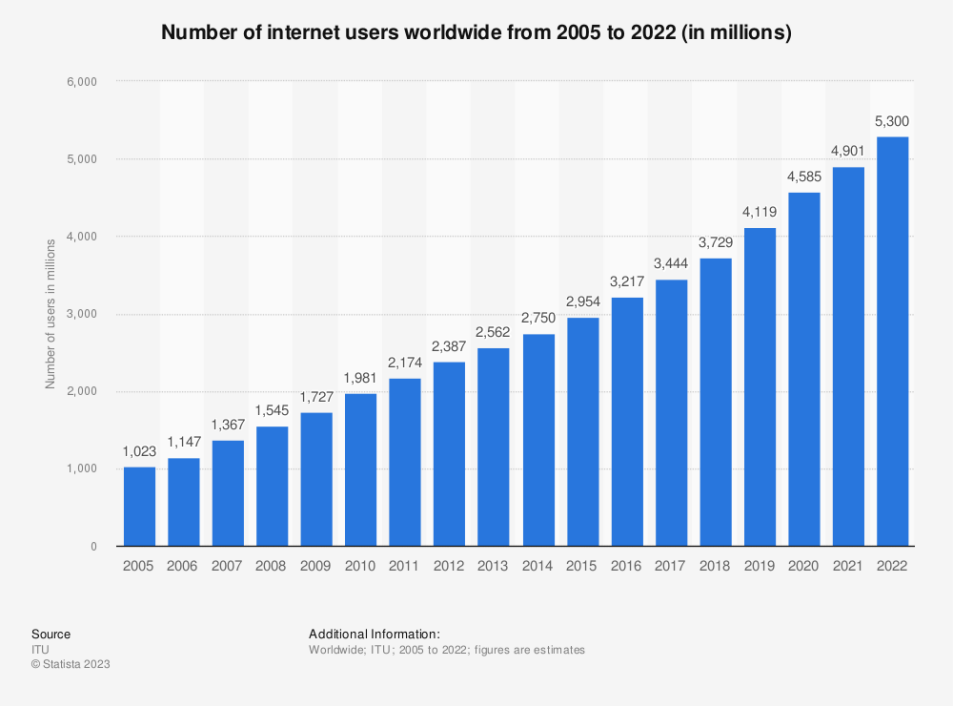


Figura 2.3: Numero de descargas de apps móviles a nivel mundial (2016-2022). Fuente [3]

En definitiva, el avance tecnológico global supone un reto para las empresas y organizaciones que pretendan tener una presencia relevante en las vidas de todo el mundo, y la entrada en el mercado de frameworks multiplataforma de desarrollo permite la entrada de forma sincronizada y con bajo coste de cualquier desarrollo

2.2 Redes Sociales

Desde poco después de la creación de internet en los años 80, las redes sociales han sido uno de los principales medios de difusión e intercambio de información, experiencias y momentos a escala global.

Estas plataformas han experimentado un rápido crecimiento en los últimos años tanto en número como en sus características y enfoque, centrándose algunas en compartir fotos, como es el caso de Instagram y Snapchat, mientras que otras se centran en microblogging y comunidades, como son los casos de Twitter y Reddit. La propularidad de este formato de plataforma es tal, que hoy en día existen incluso redes sociales enfocadas al trabajo como pueden ser Linkedin o Xing en el caso de países de habla germana.

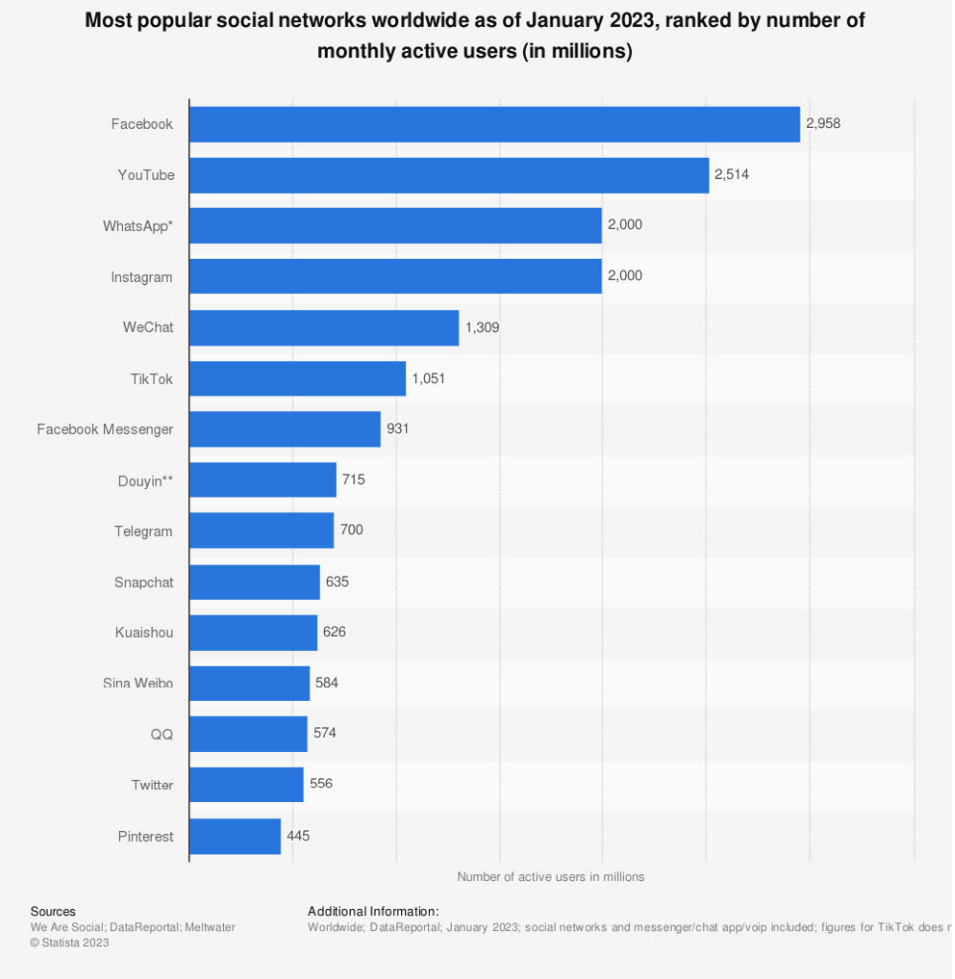


Figura 2.4: Número de usuarios activos por cada red social. Fuente [4]

A medida que las redes sociales aumentan su presencia en la sociedad, están también han dejado su huella en la salud mental, especialmente la de los jóvenes, quienes son especialmente susceptibles a su influencia.

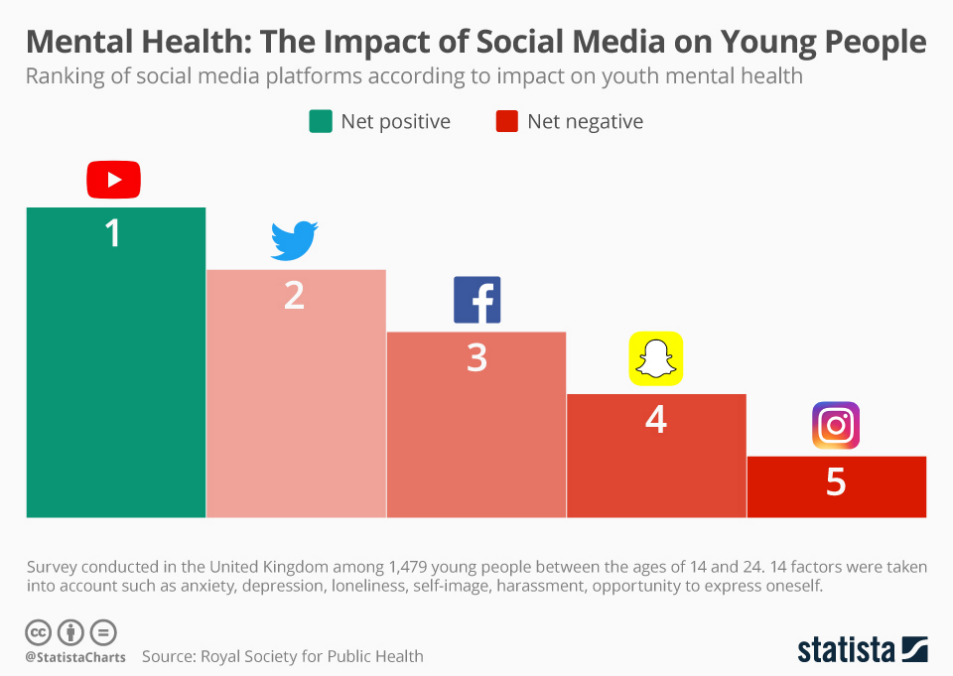


Figura 2.5: Ranking de impacto neto sobre la salud mental por red social. Fuente [5]

Redes sociales como Instagram, Snapchat o Tiktok, orientadas a la exposición de la imagen propia remunerada mediante likes, comentarios y visualizaciones, en combinación con el uso abusivo de efectos que ocultan todo aquello que el usuario categorice como imperfecciones, han exacerbado los ya existentes problemas de auto-imagen en los jóvenes hasta niveles preocupantes.

El ciberbullying, entendido como bullying llevado a cabo en la red, es uno de los causantes de este deterioro de la salud mental. Hoy en día no es necesario estar en un aula, o físicamente expuesto a nadie para recibir críticas pasadas de rosca, y es que gracias a internet todas las consecuencias de nuestra existencia en sociedad, tanto las buenas como las malas, nos acompañan a todos lados.

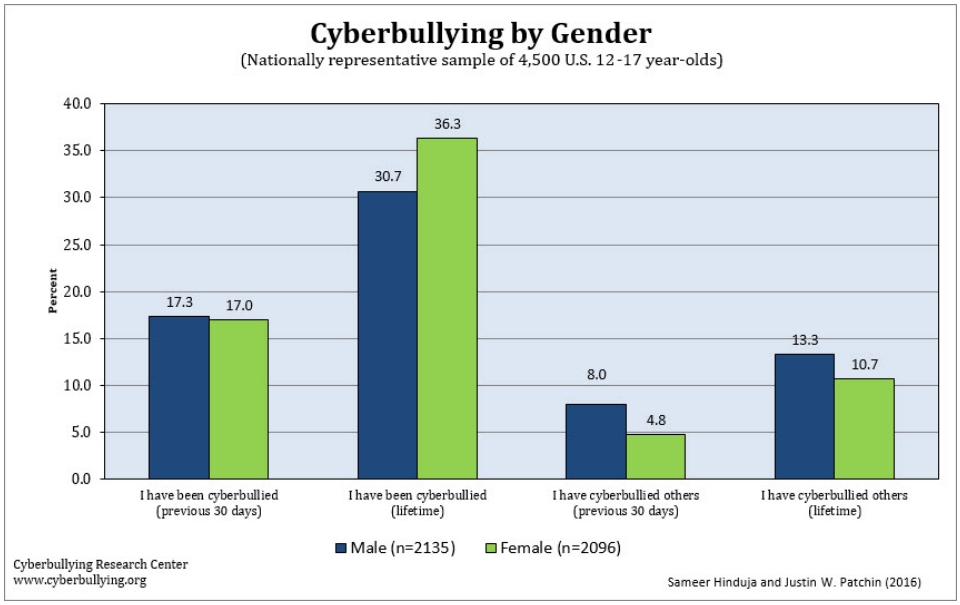


Figura 2.6: Ciberbullying, provocado y recivido, por género. Fuente [6]

El mal sabor de boca provocado por estas estadísticas, así como el que dejan los informes de la investigación de la ex-empleada de Facebook Frances Haugen sobre el impacto de las redes sociales de la empresa en la salud mental y la política mundiales, ha creado una necesidad de mercado de albergar modelos de redes sociales más saludables, en los que la imagen se trate de una forma no adulterada, y que premien la naturalidad.

La red social BeReal, lanzada en 2019 pero popularizada a mediados de 2022, es un claro ejemplo de una red social que ha entrado fuertemente en el mercado de las redes sociales, obteniendo una media de 73.5 millones de usuarios mensuales

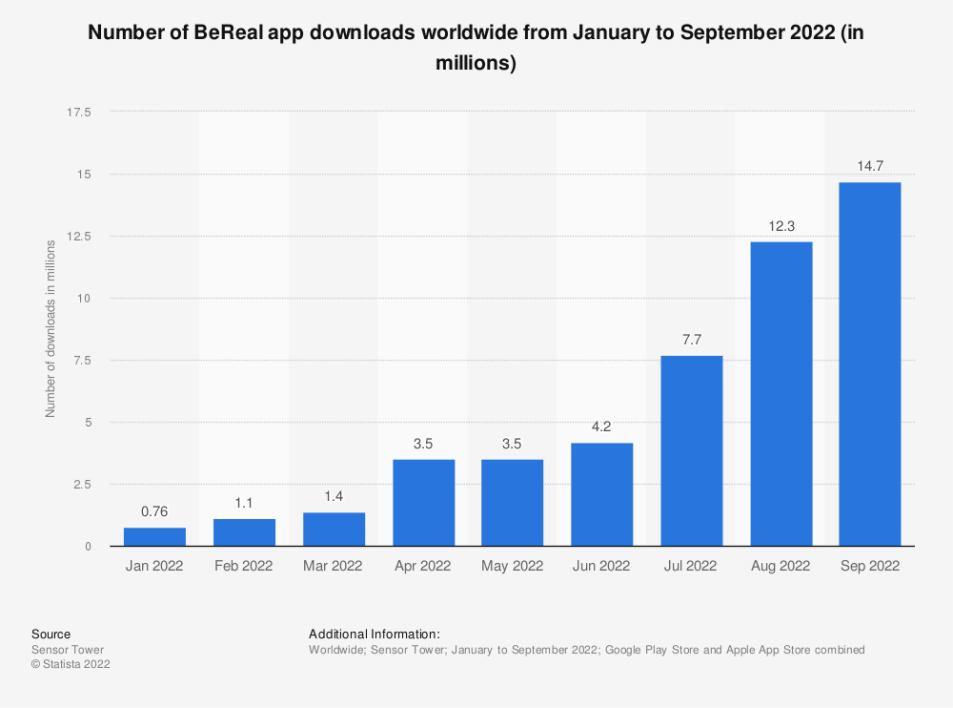


Figura 2.7: BeReal, descargas por mes a nivel mundial, 2022. Fuente [7]

El uso de BeReal es sencillo. Descarga la app, regístrate y serás notificado diariamente para subir una foto a la misma hora que el resto de sus usuarios. Las fotos se sacarán sin filtros y con ambas cámaras, mostrando tanto al usuario como su entorno y ofreciendo un formato de posts más real del que nos acostumbran otras redes sociales.

La naturalidad de esta red, en la que además se reacciona a los posts del resto con fotos de nuestra propia cara acompañadas del emoji correspondiente al sentimiento que pretendemos transmitir al creador de la foto a reaccionar, ayuda a crear ese nuevo modelo de red social del que hablábamos, y es la prueba fehaciente de que la tecnología y sus aplicaciones a un mismo campo puede tener una cara B, más positiva que sus implementaciones iniciales.

En definitiva, las redes sociales, al igual que cualquier otro servicio tecnológico, evolucionan y seguirán evolucionando según cambien las demandas de la sociedad. Estas a su vez cambiarán a la sociedad a través de su transformación, y el ciclo se repetirá, esperemos que con resultados crecientemente positivos para sus usuarios y la sociedad en general.

3. Conceptos Previos

3.1 Metodologías Ágiles

Las metodologías ágiles dentro de la industria del software son marcos de referencia de trabajo y organización de equipos de desarrollo de software que se caracterizan por su flexibilidad, adaptabilidad y tener como objetivo la entrega rápida y continua de valor; esto se consigue gracias a flujos de retroalimentación constantes, que permiten a los miembros de los equipos ágiles responder a los cambios necesarios en el proyecto de forma ágil.[8][9]

Algunos ejemplos de estas metodologías son Scrum, Kanban O Extreme Programming (XP), pero para los propósitos de explicar las herramientas empleadas en este PFG, nos enfocaremos en Scrum, Kanban y Scrumban, siendo este último una mezcla de los otros dos.

Scrum

Scrum es una metodología ágil con origen aplicado a procesos a mediados de los 80 gracias a Hirotaka Takeuchi e Ikujiro Nonaka[10], posteriormente aplicada al software en los 90 por Sutherland, Scumniotales y McKenna, y formalizada por Ken Schwaber en 1995.[11]

Los principios rectores de las metodologías ágiles, siendo estos la entrega continua y rápida de valor mediante retroalimentación, pretenden ser acentuados e implementados en el caso de Scrum durante el denominado ”Sprint”, que es una unidad de tiempo de duración fija definida por el equipo, con una duración de una o dos semanas, que se repite de forma cíclica.

En este contexto, se llevan a cabo las siguientes actividades:

- **Sprint Planning**: realizada al comienzo de cada Sprint, su objetivo es definir las metas y los elementos de trabajo que se abordarán en el sprint.

- **Daily Scrum**: de repetición diaria, en ella cada miembro del equipo comparte su progreso, lo que le bloquea en conseguirlo, y se coordina con el resto del equipo.
- **Sprint Review**: presentada al final del Sprint, aquí se muestra a los Product Owners y Stakeholders las funcionalidades desarrolladas en el sprint; además se recogen sus comentarios y se ajustan los elementos de trabajo restantes de manera acorde.
- **Sprint Retrospective**: tras finalizar el sprint, el equipo reflexiona sobre el mismo y busca mejorarlo, identificando los aspectos positivos y negativos de este, así como posibles oportunidades de mejora.

Todas estas actividades serán llevadas a cabo por el equipo Scrum, divisible en los siguientes componentes:

- **Product Owner**: responsable de definir los requisitos del producto, priorizar la lista de elementos de trabajo en un ”backlog”, y asegurarse de que el trabajo adecuado se hace en el momento adecuado.
- **Scrum Master**: que toma el rol de facilitador, asegurándose que se sigan las buenas prácticas y los principios de Scrum y eliminando aquello que pueda afectar a la productividad del equipo.
- **Equipo de desarrollo**: que es auto-organizado y multidisciplinario, es decir, todos tienen las habilidades para completar cualquier trabajo del backlog de forma individual.

Kanban

Kanban, del japonés 看 (Kàn), que significa “signo” o “señal visual”, y 板 (Bǎn), que significa “tablero”[12]. Esta metodología ágil enfoca su estilo en una gestión visual basada en tableros que marcan el estado del trabajo (en cola, haciéndose, hecho etc.), ofreciendo así la posibilidad de limitar el Work In Progress, o la cantidad de trabajo que se realiza en un mismo momento.

Al igual que Scrum, Kanban también basa su metodología en la retroalimentación continua que genere mejoras incrementales, tanto en los productos como en los procesos utilizados para el desarrollo de esos productos.

Los elementos clave de esta metodología son:

- **Métricas de Kanban**: destinadas a medir el rendimiento del equipo mediante, por ejemplo, el tiempo de finalización de una tarea, el tiempo medio de una tarea en espera etc.
- **Tablero Kanban**: dividido en columnas, que representan las etapas del flujo de trabajo para las distintas tareas.
- **Tarjetas Kanban**: que representan las tareas a hacer.
- **Límites al WIP**: que es la cantidad máxima de trabajo realizable en cada momento, se establecen para no sobrecargar al equipo.

ScrumBan

ScrumBan es una metodología ágil que combina elementos fuertes de Scrum y de Kanban, para intentar obtener una metodología más eficiente que las dos anteriores por separado.

En este sentido, se combinan los sprints de Scrum, con la visualización y la limitación del WIP de Kanban[13], obteniendo una metodología ágil combinada que se adapte mejor a las necesidades de los equipos de lo que lo harían las metodologías anteriores por sí solas.

Además, cabe recordar que la mayoría de metodologías ágiles abogan por ser adaptadas a las circunstancias particulares de cada equipo y proyecto.

3.2 Diseño Gráfico

El diseño gráfico es una disciplina artística que pretende transmitir información de forma efectiva mediante el uso de principios estéticas y técnicas de composición para combinar imágenes, colores y tipografías dispuestos de distintas formas según el contexto y lo que se pretenda transmitir.[14]

Dentro de esta disciplina, y aplicándola al contexto de la sociedad de la información y la informática, encontramos dos conceptos claves, el UI o User Interface design, y el UX o User eXperience design, que exploramos a continuación.[15]

Asimismo, repasaremos el concepto de Wireframes, ya que su relevancia es notable para el contexto de este PFG.

UI

El diseño UI, o de Interfaces de Usuario, tiene como objetivo obtener una representación final de lo que será la parte visual de la aplicación, incluyendo sus todos los elementos de cada una de sus pantallas con sus colores, disposición en el espacio, tamaño y tipografía correspondientes.

Este paso es clave dentro del desarrollo de cualquier interfaz, ya que define los elementos con los que el usuario interactuará en última instancia.

Algunas elementos base de esta disciplina son:

- **Fundamentos del diseño**: para comprender qué es el diseño y cuáles son sus distintas áreas.
- **Leyes de Gestalt**: que establecen pautas y relaciones psicológicas entre distintos elementos visuales y su disposición relativa[16]
- **Tipografía e iconografía**: que son especialidades en las que se pretenden representar y transmitir acciones, sensaciones e información mediante el uso de distintas formas en el texto, así como imágenes con significado.[17]
- **Patrones de diseño**: utilizados para aprovechar los beneficios de la estandarización a nivel de industria, evitando caer en anti-patrones.
- **Buenas prácticas**: como establecer una jerarquía visual, agrupar visualmente elementos relacionados etc.[18]

UX

El diseño UX, o de Experiencia de Usuario, es el proceso mediante el cual una interfaz de usuario es evaluada con respecto a la satisfacción que su utilización genera en el usuario, esto es, para que una interfaz cumpla con los principios de UX esta debe ser útil, funcional, debe de estar adecuadamente estructurada y debe mostrar su contenido de forma correcta.[19][20]

Exploremos brevemente el cumplimento de cada uno de estos puntos por parte de una interfaz:

- **Útil**: es decir, que con ella el usuario pueda completar la tarea que se le promete.
- **Funcional**: esto es, que está libre de errores, o que no puede dar lugar a equivocación dado un propósito concreto.
- **Estructura adecuada**: en lo que al orden y colocación de sus elementos se refiere.
- **Contenido correcto**: en tanto en cuanto los contenidos de la interfaz se adecuan al contexto de la misma, proporcionando información útil, correcta y que se presenta de forma coherente con el resto de elementos.

Adicionalmente y siguiendo con el tono retrospectivo de las metodologías ágiles, el UX aspira a evolucionar las interfaces de usuario mediante una retroalimentación y mejoras continuas basadas en las necesidades reales de los usuarios de las interfaces.

Wireframing

El wireframing es una técnica temprana de representación visual y esquemática de los diferentes elementos de una interfaz, así de la interrelación de los mismos.[21][22]

Esto permite a diseñadores y desarrolladores tener un marco común de entendimiento de la estructura y diseño de la interfaz, antes de invertir tiempo y recursos en desarrollarla de forma completa.

Al igual que en las metodologías ágiles, el wireframing es un proceso iterativo en el que una interpretación inicial puede ser potencialmente refinada y mejorada.[23]

4. Tecnologías utilizadas

Para el desarrollo de Wemon ha sido necesario el uso de Flutter en el entorno integrado de desarrollo Android Studio, así como emuladores de Android, XCode, Firebase para el backend, AWS para almacenar imágenes, Heroku para la API.

A nivel de coordinación de equipo, se ha utilizado Slack, Google Meet y WhatsApp, así como Github Projects para una coordinación técnica y planificación de los sprints.

4.1 Desarrollo

A continuación exploraremos las tecnologías que han sido utilizadas, así como los motivos para elegirlas y su función.

Flutter

Flutter es un framework open-source con licencia BSD de desarrollo de aplicaciones multiplataforma, que actualmente soporta Android, iOS, MacOS, Windows, Linux y sistemas embebidos.

Actualmente hay más de 350.000 aplicaciones creadas con Flutter, entre las que destacan aplicaciones de empresas como Alibaba Group, BMW, eBay, Google y Toyota entre otras.[24]

La arquitectura utilizada por Flutter a nivel interno para convertir una misma base de código a código nativo específico para cada plataforma es la siguiente:

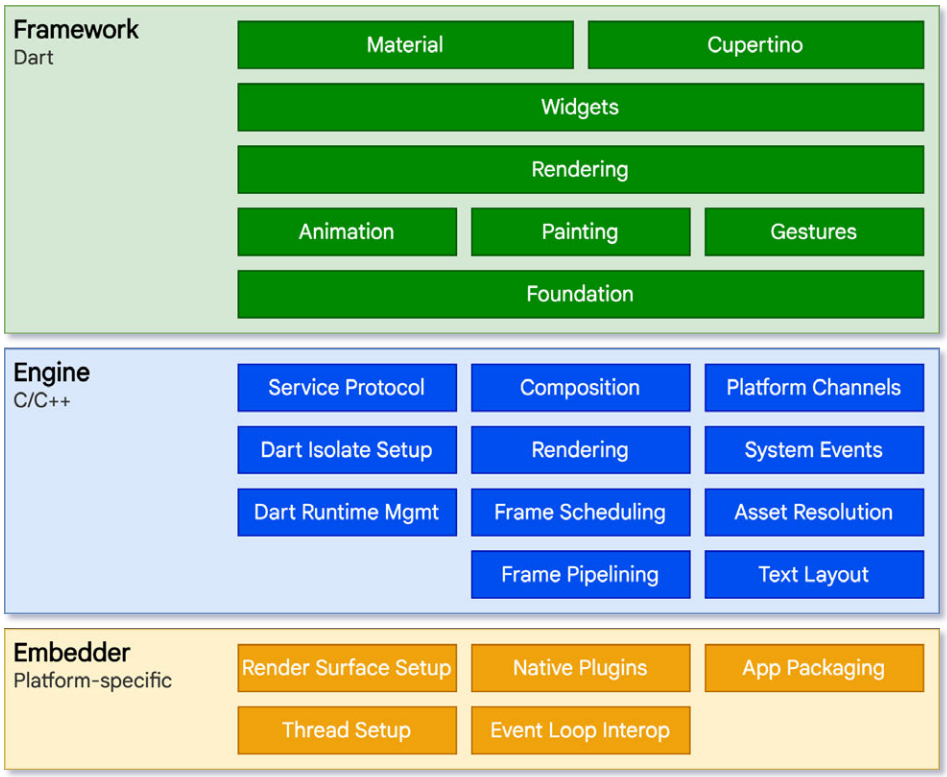


Figura 4.1: Capas arquitectónicas de Flutter. [25]

Esta arquitectura de capas provee una base mínima de servicios básicos como animaciones, figuras y detección de gestos.

- **Framework**: escrito en Dart, incluye la capa y las abstracciones a las que accede el desarrollador, como la clase Widget[26], los distintos tipos de animación[27], la detección de gestos[28], librerías Material[29] y Cupertino[30] etc.
- **Engine**: escrito en C++, se encarga de proporcionar las capacidad básicas del código para pintar cada fotograma.
- **Embedder**: específico para cada plataforma objetivo, se encarga de obtener los plugins nativos necesarios para la ejecución de la aplicación, empaquetar la aplicación de la forma que corresponda según la plataforma etc.

Android Studio

Android Studio es un Entorno de Desarrollo Integrado o IDE, diseñado para el desarrollo de aplicaciones Android.[31]

Este entorno está basado en IntelliJ IDEA, ya que ambos son de la empresa JetBrains, y proporcionas interfaces de usuario similares en cuanto a distribución y manejo.

Como muchos IDEs, Android Studio posee un editor de código, que en su caso viene acompañado de herramientas especializadas en desarrollo Android, como el SDK o Kit de Desarrollo de Software de Android, Gradle[32], asistentes al diseño de interfaces de usuario, herramientas de administración de proyectos Android y toda una serie de plugins ofrecida en su tienda de plugins[33].

Emuladores de Android

Los emuladores de Android utilizados en este caso son Android Virtual Devices ofrecidos por Android Studio[34].

Estos emuladores o dispositivos virtuales permiten simular un teléfono de forma fiel al teléfono real, creando así un entorno realista de pruebas durante el desarrollo.

Los principales parámetros configurables en un AVD son:

- **Tamaño**: entendido como las dimensiones de la pantalla del dispositivo, en pulgadas, aspect ratio de la pantalla y resolución física.
- **Sistema Operativo**: pudiendo elegir la versión de Android que se desee ejecutar.
- **Estado del teléfono**: es decir, podemos elegir si el teléfono está conectado o no a una red celular emulada, a una red Wi-Fi, si se activa o no la conexión por Bluetooth, y así sucesivamente con cualquier parámetro configurable en un teléfono real. Adicionalmente, se puede cambiar la ubicación del dispositivo en el mapa a conveniencia.

XCode

XCode es un IDE creado por Apple para desarrollar aplicaciones para sus sistemas operativos, como iOS, macOS, watchOS etc.

Este IDE proporciona las herramientas necesarias para diseñar, programar y compilar aplicaciones.

Al igual que Android Studio, Xcode ofrece un editor de código, un constructor de interfaces, herramientas de depuración y pruebas, gestión de proyectos y compilación y distribución del código.[35]

Firebase

Firebase es una plataforma BaaS o Backend as a Service propiedad de Google hecha para el desarrollo de aplicaciones móviles y web.

En Firebase, la división del trabajo se hace por proyectos. A cada proyecto, se le pueden asociar distintas plataformas (iOS, Android, Web...).

Entre los servicios ofrecidos por Firebase se encuentran

- **Analytics Dashboard**: donde puedes seguir las estadísticas de uso de tus diferentes plataformas, incluyendo sus estadísticas de uso por versiones, los usuarios, el tiempo de uso etc. [36]
- **Remote Config**: donde puedes definir parámetros en remoto para hacer cambios prácticamente instantáneos al funcionamiento de tu aplicación sin requerir que tus usuarios se bajen una nueva versión de la misma. [37]
- **Messaging**: donde puedes crear mensajes masivos para todos los usuarios de tu aplicación, para promociones, anuncios generales etc. además de ofrecer un servicio de notificaciones. [38]
- **Authentication**: donde se ofrece una forma inmediata de implementación de un Social Login, que permite obtener los datos en tiempo de registro desde el perfil del usuario en otras aplicaciones, para agilizar el proceso de registro e inicio de sesión. [39]

AWS

AWS es una plataforma de Amazon que ofrece diversos servicios de computación en la nube con multitud de casos de uso. [40]

Entre los principales servicios por categorías tecnológicas que ofrecen se encuentran

- **Análisis de datos**: ofrecido, entre otros, por sus productos Athena, EMR y Kinesis. [41]
- **Machine Learning**: que se ofrece por productos como su línea SageMaker para Machine Learning, y Comprehend entre otros para Inteligencia Artificial. [42]
- **Informática Serverless**: de la mano de sus Lambda, Fargate, Step Functions etc. [43]
- **Almacenamiento**: donde destacan los S3, EFS y EBS entre muchos otros. [44]

Heroku

Heroku, de manera similar a AWS, es una plataforma de servicios en la nube que ofrece servicios de plataforma y datos.

Esta plataforma soporta lenguajes de programación como Python, Node.js, Java y PHP entre otros.

Además, Heroku ofrece servicios de bases de datos con Postgres, Redis y Apache Kafka.[45]

Dentro del contexto de este TFG, Heroku se usa para alojar el servicio de nuestra API REST gracias a su servicio Heroku Platform. [46]

4.2 Coordinación del desarrollo

Github Projects

Github Projects es una herramienta ofrecida por Github que, asociada a un proyecto, permite dividir las tareas del mismo a gusto del equipo de desarrollo para facilitar el control y planificación del desarrollo, así como el seguimiento efectivo de los objetivos definidos por el equipo. [47]

Entre las vistas ofrecidas por Github project se encuentra la vista de Backlog, Sprint en curso, una vista de calendario para el desarrollo, así como de pizarras personalizables. [48]

Slack

Slack es una plataforma de comunicación empresarial basada en mensajería instantánea tanto entre miembros del equipo, uno a uno, como entre miembros con distintos roles.

La comunicación en Slack se realiza mediante canales, a los que se tiene acceso en base a lo que defina el administrador de Slack en cada empresa; normalmente el acceso a un canal viene definido por las responsabilidades de cada miembro del equipo a nivel de la empresa. [49]

Como tal, Slack ofrece la posibilidad de compartir archivos por sus canales, crear notificaciones y recordatorios, así como realizar llamadas y videoconferencias.

Además, Slack ofrece integración con aplicaciones de terceros como Google Drive, Trello y Github entre otros. [50]

Google Meet

Google Meet es una plataforma de comunicación orientada a videoconferencias desarrollada por Google.

Las llamadas se pueden generar con una fecha y hora predeterminadas con el objetivo de programarlas en el calendario, así como en el mismo momento si así se desea.

Además, Google Meet ofrece la posibilidad de compartir la pantalla de los usuarios que así lo deseen, agilizando la presentación de información entre usuarios.[51]

WhatsApp

WhatsApp es un servicio de mensajería instantánea basada en chats, que además permite crear grupos, canales de difusión, hacer llamadas por voz y videollamadas, además de compartir fotos y archivos de forma rápida.

La utilidad de WhatsApp a nivel de coordinación de equipo es evidente si todos los miembros están acostumbrados a su uso.

Así mismo, otras aplicaciones de mensajería como Telegram[52] cumplirían una función similar de facilitar una comunicación rápida e informal entre los miembros del equipo para cuestiones de coordinación puntuales. [53]

5. Metodología

5.1 Fase Previa: Organización del Desarrollo

Una de las fases más importantes de todo nuevo proceso de desarrollo es decidir qué se va a hacer y cómo se va a hacer.

Poniendo el caso de Wemon, los objetivos técnicos eran claros desde el principio:

- **Desarrollar una aplicación para móvil**: ya que así es como se concibió el proyecto en los inicios.
- **Alcanzar al máximo número de usuarios posibles**: objetivo base de toda red social.

así como también eran claras las restricciones presentes:

- **El equipo de desarrollo es pequeño**: de unas cuatro a seis personas.
- **La experiencia del equipo es limitada**: no hay nadie en el equipo con experiencia específica en desarrollo de aplicaciones.

Dados estos objetivos y restricciones, las cuestiones a resolver eran:

1. **¿Cómo nos dividiremos como equipos de desarrollo?**
2. **¿Qué rol tomará cada miembro del equipo?**
3. **¿Cómo nos organizaremos entre los equipos de la organización?**

4. **¿Qué tecnologías utilizaremos y por qué?**
5. **¿Qué patrones de desarrollo seguiremos?**

Respondamos a las siguientes cuestiones de forma breve

División del Equipo de Desarrollo

En lo que a la división del equipo de desarrollo se refiere, la opción elegida vino dada por la arquitectura de aplicación distribuida elegida, cliente-servidor en nuestro caso dadas las necesidades y circunstancias del proyecto, y fue:

- **Equipo de Front-end**: compuesto en la actualidad por dos personas, es el encargado del desarrollo de la aplicación cliente, y labor del cual hace seguimiento este PFG.
- **Equipo de Back-end**: compuesto por dos personas, es el equipo encargado del desarrollo del lado de la aplicación del lado del servidor.

La arquitectura cliente servidor se eligió en nuestro caso dada la naturaleza del proyecto, como se comenta anteriormente.

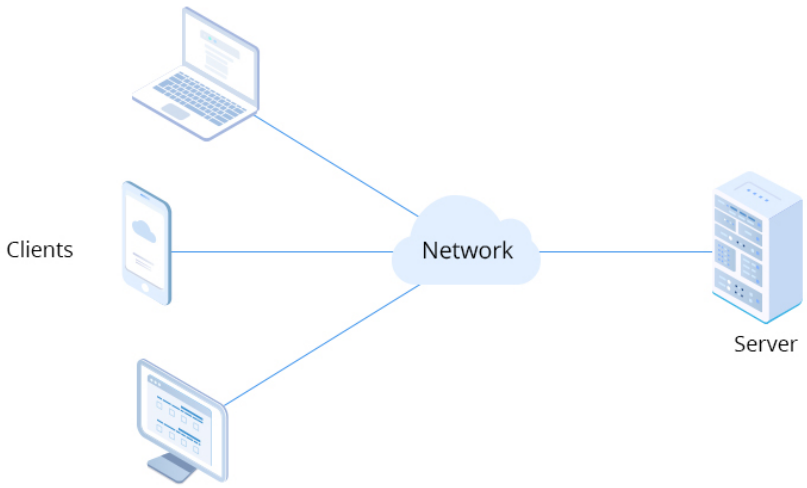


Figura 5.1: Esquema de la arquitectura cliente-servidor. Fuente[54]

Este patrón arquitectónico se ajustaba perfectamente a las necesidades que presentaba nuestro futuro sistema, ya que ofrece ventajas como una gestión centralizada y mayor control sobre datos

potencialmente sensibles[55], como pueden ser las publicaciones de los usuarios o sus propios datos.

Además, la diferencia entre las tareas a realizar por cada equipo es evidente.

Por un lado, el equipo de front-end se encargará de todo el apartado visual y de gestión y presentación de datos dentro de la lógica de negocio de la aplicación, mientras que el equipo de back-end se encargará de su almacenamiento, modificación, lectura y borrado con todo lo que ello conlleva.

El despliegue y puesta en producción de cada elemento se llevará a cabo por cada equipo, coordinándose ambos para cambios importantes y durante todo el proceso de desarrollo.

Roles dentro de los equipos de Desarrollo

Con respecto a los roles internos de cada uno de los miembros de los equipos de front-end y back-end, la elección fue relativamente sencilla una vez conocida la metodología de desarrollo a seguir por el equipo.

Conocidas las restricciones, y puesto que se trata de un equipo pequeño con profesionales independientes y geográficamente dispersos que dedican un tiempo limitado de su día a día al proyecto, la elección fueron las metodologías ágiles, en concreto ScrumBan3.1, al tratarse de una metodología en la que todo el equipo ha trabajado o al menos oído. Además, ScrumBan ofrece la flexibilidad necesaria dadas las circunstancias para dividir las tareas y coordinarlas de forma cómoda.

Organización entre equipos

Para organizar los equipos, las tecnologías escogidas, su propósito específico de uso y causas fueron:

- **WhatsApp**4.2: para comunicación general entre front-end, back-end y dirección; acordar reuniones etc. por su uso de forma cotidiana y personal por parte de todos los miembros del equipo.
- **Slack**4.2: para comunicación entre equipos de desarrollo y con equipo de diseño, sobre cuestiones referentes al desarrollo, ya que permite una comunicación por canales e hilos de conversación.

- **GitHub Projects**4.2: para organizar las tareas derivadas del análisis y hacer su seguimiento de forma cómoda, ya que permite relacionar las ramas de desarrollo dentro de GitHub de forma cómoda y rápida.
- **GitHub**: ya que, al usar GitHub Projects, era la herramienta de control de versiones a utilizar; además es la herramienta de control de versiones con la que más experiencia cuenta el equipo, y permite organizar el trabajo por ramas.
- **GMail**: para el correo corporativo y cuestiones relacionadas con el proyecto.
- **Google Meet**4.2: ya que permite hacer videollamadas programadas de forma cómoda para cuentas de GMail.

Gracias al uso de todas estas herramientas para su labor designada, la coordinación del equipo y del trabajo producido ha sido facilitada y agilizada.

Tecnologías, y el por qué

La principal decisión tecnológica a nivel de desarrollo del la aplicación de Wemon fue qué herramienta utilizar para desarrollar el front-end, ya que el back-end iba a estar basado en soluciones similares implementadas por anterioridad por miembros experimentados del equipo.

Con el objetivo de permitir a la aplicación crecer de la forma más rápida posible si así se necesitase, y el de permitirla operar tanto en sistemas operativo móviles Android como iOS, la decisión fue la de usar Flutter4.1 dadas sus capacidades como framework multiplataforma, aunque inicialmente se planteó hacer el código Android e iOS por separado, pero esta opción era inviable tanto en desarrollo como en mantenimiento y a la hora de hacer cambios dadas las restricciones del equipo.

Patrones de desarrollo, y el por qué

Estandarizar la estructura interna que va a tener el código de tu proyecto es una cuestión de suma importancia.

La estandarización permite simplificar procesos tanto de creación, como de modificación, pruebas y mantenimiento.[56]

En el caso de Flutter, un patrón popular con el que nos encontramos tras buscar información sobre el framework fue el patrón BLoC[57][58], que recibe su nombre de ”Business Logic Components”

y permite dividir la aplicación de Flutter en componentes estandarizados que se encargen de la gestión de estados en cada pantalla.

En nuestro caso, y adaptando el patrón a las necesidades del proyecto, el patrón aplicado a código queda de la siguiente forma para cada una de las pantallas:

- **screenName_screen**: que es la pantalla donde se renderizan los elementos visibles por el usuario y con los que este puede interactuar.
- **screenName_event**: donde se especifican los posibles eventos generados por el usuario al interactuar con la lógica de negocio visualizada en la pantalla.
- **screenName_state**: donde se especifican los posibles estados asociados a cada evento, siendo estos ”loading”, ”succcess” o ”failed”.
- **screenName_bloc**: el cual escucha los eventos de la pantalla, y devuelve el estado correspondiente.
- **screenName_repository**: donde se hacen las llamadas a la API para gestionar los datos almacenados correspondientes a la pantalla como imágenes, usuarios etc. mediante transformaciones de JSON a Dart y Dart a JSON.

De tal forma que el flujo de datos, iniciado en el usuario, es el que se puede observar en la figura 5.2:

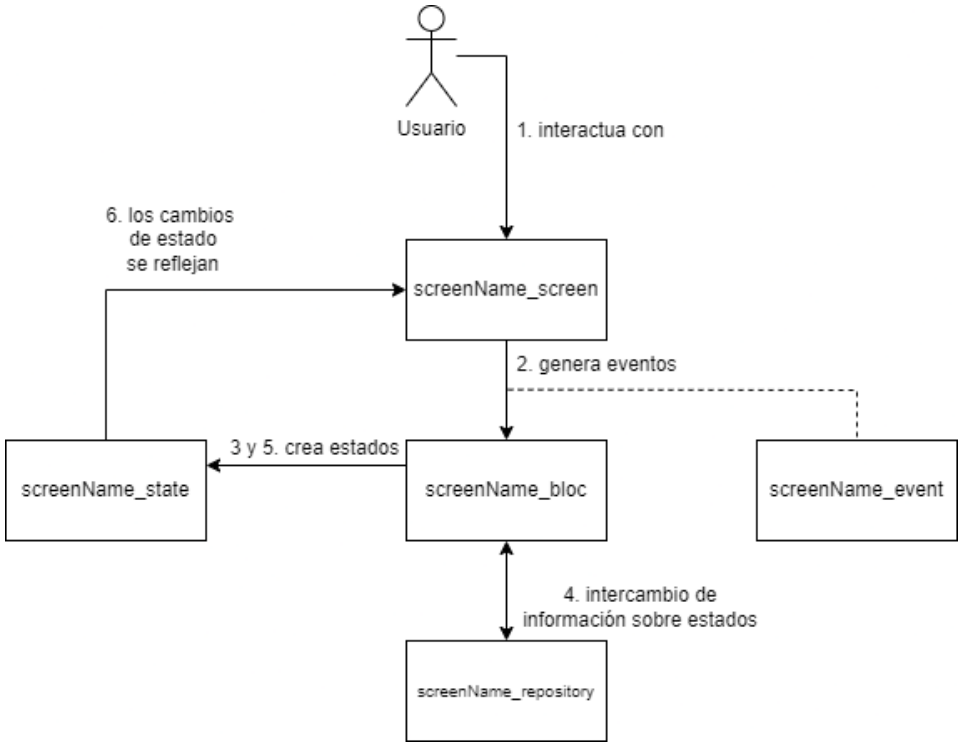


Figura 5.2: Esquema del flujo de eventos de interacción del usuario con la pantalla usando el patrón BLoC adaptado. Fuente propia.

Como podemos ver, el usuario interactúa con los elementos mostrados en la screen. Esto genera una serie de eventos almacenados en el screenName_event que son recogidos por el bloc, el cual intercambiará información con el repository para general los distintos estados que se verán reflejados en la screen. Una vez que el flujo del bloc termine, lo normal es que el estado mostrado en pantalla sea uno de éxito, o uno de fallo; el resto del tiempo, será uno de carga.

5.2 Primer Caso de Desarrollo: Implementación de una Nueva Pantalla desde Cero

En esta sección, repasaremos los pasos a seguir para desarrollar una nueva pantalla desde cero, aplicando el patrón BLoC y explicando la comunicación típica necesaria en el proceso.

Análisis del diseño

Para este caso, se parte de la base de que el equipo de diseño ha presentado una propuesta para una nueva pantalla, con su correspondiente diseño de interfaz de usuario.

En ese momento, la pantalla será revisada por todos los miembros de los equipos de desarrollo de forma breve, para identificar posibles errores o cuestiones que susciten dudas y se puedan ver en una pasada rápida.

En caso de que haya duda, estas se comentan con el equipo de diseño para que las aclaren, o, si fuese necesario, hacer las modificaciones necesarias.

Estas dudas suelen surgir en casos de desarrollos de pantallas que tengan otras pantallas adicionales vinculadas, como puede ser la pantalla de configuración dentro de la pantalla del usuario. En ese caso, habría una pantalla secundaria ”contenida” en su flujo dentro de la principal.

División de tareas y organización del trabajo

A continuación, y si no hay más dudas, esta tarea será asignada a un miembro del equipo de front-end, y un miembro del equipo de back-end. El criterio de selección es en primera instancia uno de disponibilidad, es decir, se le asigna al miembro del equipo que mayor disponibilidad tenga.

Si hay varios miembros del equipo igualmente disponibles, la tarea será asignada por criterio de prioridad y experiencia, esto es, si la tarea es de prioridad alta se escogerá al miembro del equipo que más experiencia tenga.

Con el fin de poder asignar discernir entre los distintos niveles de prioridad, establecimos un sistema de prioridad de tarea basado en los siguientes criterios:

- **Prioridad alta**: normalmente asignada a tareas que son prioritarias para cuestiones de negocio, como reuniones con potenciales accionistas, fechas de salida acordadas con marketing etc. Asignada también a errores en funcionalidades que no permitan al usuario acceder a funcionalidades básicas.
- **Prioridad media**: asignada a características importantes, como el desarrollo de una nueva funcionalidad dentro de una pantalla o una pantalla completa. Esta prioridad es la mínima asignable a desarrollos que vayan a ser funcionalidades básicas de cara al usuario.
- **Prioridad baja**: asignada a funcionalidades adicionales que deberán ser incorporadas a lo largo de futuras versiones, o a errores visuales que no afecten a la experiencia del usuario de forma significativa.

Por último, y en caso de que la tarea tenga una prioridad baja, se le asignará al miembro del equipo que menos experiencia tenga con el objetivo de que este coja soltura en el desarrollo, y pueda resolver las dudas que le surjan con cualquier miembro del equipo.

Los seleccionados para el desarrollo de ambos equipos de front-end y back-end deberán estar en comunicación en todo momento y sobre todo en fases iniciales del desarrollo, con el objetivo principal de definir las estructuras de datos necesarias para representar el estado de la pantalla a nivel de API, las funciones permitidas por la misma, cuestiones como la paginación si fuera necesaria etc.

En ese momento, el desarrollador de front-end deberá dividir la pantalla en componentes, identificando los elementos reutilizables, para facilitar así su uso en otras pantallas y el mantenimiento de la aplicación en general.

Pongamos un ejemplo:

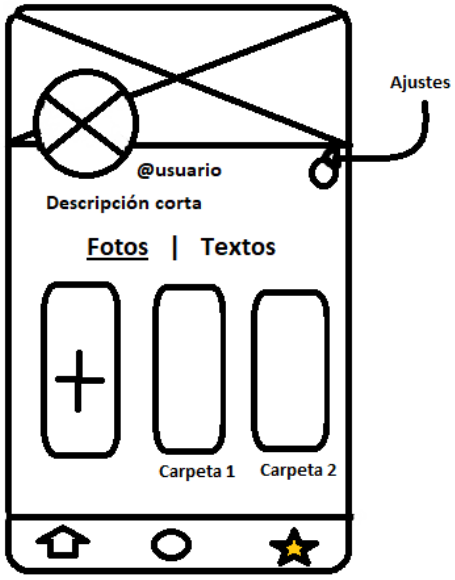


Figura 5.3: Ejemplo de un diseño inicial y funcional de la pantalla del perfil de un usuario. Fuente propia.

A primera vista, podemos identificar varios elementos, divisibles en las siguientes categorías:

- **Contexto de la pantalla**:
    1. La pantalla se accede mediante un icono, en ese ejemplo el de la estrella, situado en la barra de navegación inferior.
    2. Esta pantalla da acceso a otra pantalla, la de ”Ajustes”.
    3. Esta pantalla da acceso a otras dos pantallas, la de creación de nuevas carpetas de fotos, y la de cada carpeta en sí.
    4. Adicionalmente, dentro de los tipos de elementos bajo el perfil del usuario, encontramos ”Fotos” y ”Textos” como opciones alternativas representadas igualmente por carpetas, información que hemos obtenido tras una consulta con el equipo de diseño5.2.

Este contexto de pantalla saca a relucir el hecho de que se necesitarán diseños adicionales para completar el árbol de pantallas que cuelgan de esta, pero además nos dejan ver las siguientes estructuras de datos a consensuar con back-end:

- **Estructuras de Datos en pantalla**
  1. **Usuario**: que a su vez contiene
     - **id**: tipo int, necesario para identificarlo.
     - **foto_perfil**: tipo string, es la URL para poder representar la foto de perfil del usuario.
     - **foto_encaebzado**: tipo string, es la URL para poder representar la foto del encabezado del usuario.
     - **nombre_usuario**: tipo string, es el nombre del usuario.
     - **descripcion_usuario**: tipo string, es la descripción del usuario.
  2. **Carpeta**: compuesta por los parámetros
     - **id**: tipo int, necesario para identificarla y potencialmente recuperar desde la API los datos para su pantalla cuando corresponda.
     - **foto_carpeta**: tipo string, es la URL para poder representar la foto de portada de la carpeta.
     - **nombre_carpeta**: tipo string, es el nombre dela carpeta.
     - **tipo_carpeta**: tipo string, para distinguir entre carpetas de fotos y carpetas de textos.

Estas estructuras de datos y contexto definidos nos permiten organizar el trabajo de la forma que consideremos más cómoda, por ejemplo creando las siguientes micro-tareas en GitHub Projects4.2:

1. **Programar las Estructuras de Datos**: es decir, tener lista la representación en la aplicación a nivel de clase de las estructuras de datos de Usuario y Carpeta, con su correspondiente método de transformación de JSON a Dart.
2. **Programar el BLoC**: esto incluye la pantalla ”profile_screen” con sus eventos correspondientes ”profile_event”como pueden ser la carga de la pantalla al seleccionarla en la barra de navegación, los tres estados asociados a los eventos (”loading”, ”failure” y ”success”) en ”profile_state”, y la relación entre ambos en el ”profile_bloc”.

3. **Probar la pantalla**: haciendo pruebas de uso, probando el uso de valores nulos en campos clave como la foto de perfil o de encabezado, y pruebas en distintos dispositivos mediante el emulador de Android Studio4.1.

Con el objetivo de trabajar en la implementación de este diseño en paralelo con otros compañeros y sus correspondientes tareas, crearemos una rama en GitHub con un nombre lo suficientemente descriptivo y que parta de la última versión de la rama principal, la más estable del proyecto. Posteriormente y tras haber realizado las pruebas correspondientes, uniremos nuestra rama del proyecto a la rama principal.

Implementación del diseño

Una vez aclaradas las estructuras de datos, los posibles eventos y sus estados asociados, llega la hora de implementar el diseño en código.

Para esto, hay que tener en cuenta que esta pantalla convivirá con otras, y esto se traduce en la necesidad de organizar el código de forma coherente.

Las opciones para estructurar un proyecto son muchas, pero tras varias iteraciones, la estructura elegida para Wemon fue la siguiente:

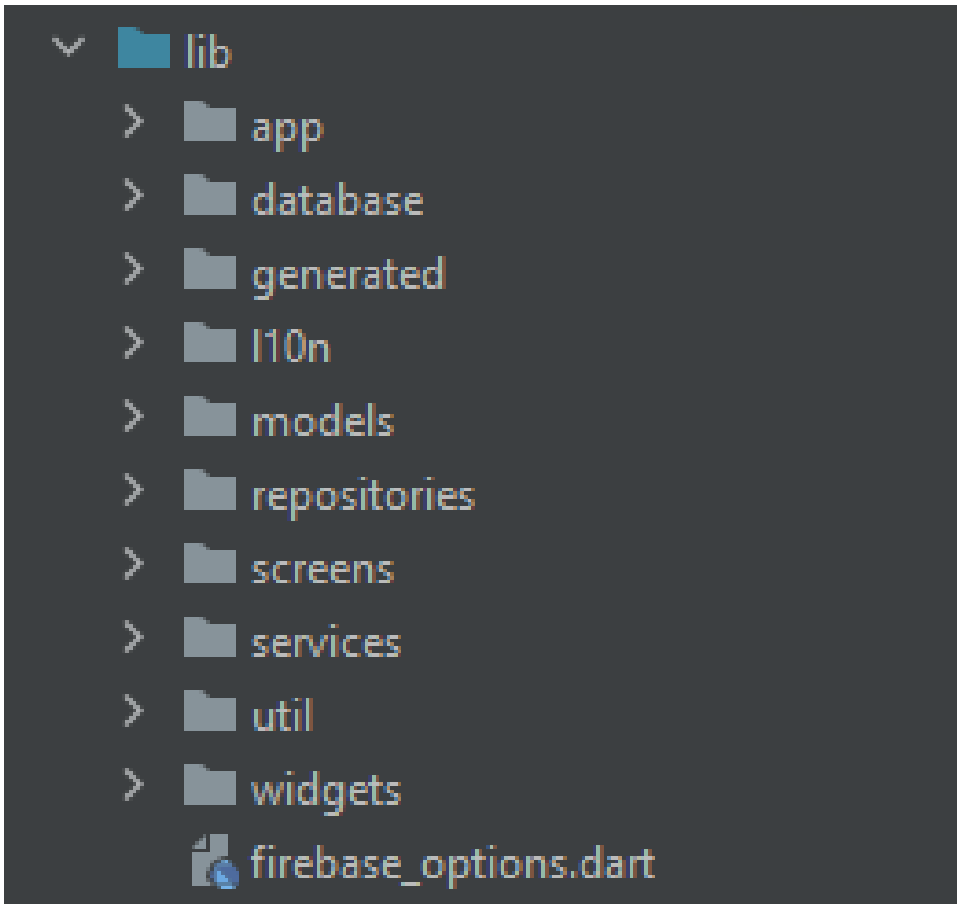


Figura 5.4: Estructura del proyecto. Fuente propia.

Repasemos el contenido general de las principales carpetas en la estructura actual del proyecto:

- **database**: aquí gestionamos la caché, elemento importante para ahorrar llamadas innecesarias a la API[59].

- **generated**: desde aquí se seleccionan los textos en el idioma correspondiente, según la configuración del dispositivo del usuario.
- **l10n**: donde se dividen los literales de texto en distintos idiomas, para ser usados posteriormente por los archivos de la carpeta "generated". Recibe este nombre por ser así como se llama al proceso de localización del software, o adaptación a distintos idiomas[60].
- **models**: aquí se almacenan las estructuras de datos que componen el modelo de datos de la aplicación, como los ya mencionados "Usuarios" y "Carpetas".
- **repositories**: es donde se guardan los archivos de tipo "_repository". El motivo por el que se separan de sus pantallas es que, al estar distintas pantallas interconectadas y por el tipo de aplicación que se desarrolla en este caso concreto, puede requerirse el uso de funciones de un repositorio en una pantalla que no corresponda al nombre de ese repositorio. Por ejemplo, se puede requerir el "chat_repository" en la "plans_screen".
- **screens**: en esta carpeta se almacenan los archivos clave del patron BLoC5.1 aplicados a cada pantalla, salvo el repositorio por las razones comentadas anteriormente.
- **util**: que recopila una serie de funciones usadas de forma recurrente a lo largo de la aplicación como hojas de estilos, mocks para pruebas, constantes, funciones para dar formatos predefinidos a fechas etc.
- **widgets**; aquí se definen elementos reutilizables como botones, iconos, tarjetas y demás elementos interactivos que, por su diseño estilístico, van a ser reutilizados en varias pantallas. Esto ayuda a aumentar la productividad en el desarrollo, así como la calidad y consistencia del software desarrollado[61].

Como notas adicionales, el archivo "firebase_options.dart" contiene la configuración del proyecto en Firebase, sobre todo a nivel de IDs y APIKeys, lo cual permite al proyecto de Firebase4.1 enviar la información específica a la aplicación dependiendo de la plataforma en la que esta se ejecute; en la carpeta "app" quedan aún funciones en estado de migración a lo que ahora es la carpeta "util". El motivo de esta migración es el crecimiento en alcance y tamaño en código del proyecto, lo que ha hecho que sea necesario dividir las funciones de una forma más clara para facilitar la mantenibilidad del código.

Entrando en detalle en la estructura del código de pantalla del perfil del usuario, desarrollado dentro de la carpeta ”screens”, nos encontramos lo siguiente:

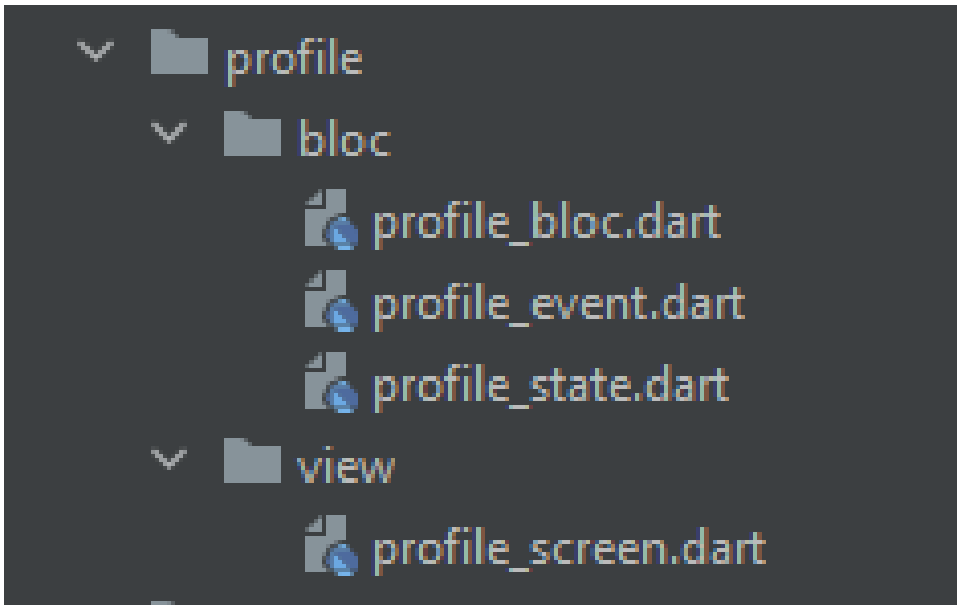


Figura 5.5: Estructura del BLoC aplicado al perfil del usuario. Fuente propia.

Como vemos, dentro de la carpeta correspondiente al perfil hay otras dos, la carpeta ”bloc”, que contiene el ”bloc”, ”state” y ”event”, y la carpeta ”view”, que contiene la ”screen”.

A continuación se muestra la implementación del evento ”Get Profile”, desde su llamada en la ”screen”, hasta su carga definitiva.

```dart
class ProfileScreen extends StatefulWidget {
  final bool isNavigating;

  ProfileScreen({this.isNavigating});

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

class _ProfileState extends State<ProfileScreen> {
  final GlobalKey<UserScreenState> _userScreenState =
      GlobalKey<UserScreenState>();
  BuildContext _blocContext;
  User _profile = Constants.emptyUser;
  List<Post> _recentPosts = [];
  bool isNavigating,
      _canClose = false,
      _isLoading = true,
      _showWeems = true,
      isFabVisible = true;
```

Figura 5.6: Declaración inicial de variables en la profile_screen. Fuente propia.

Primeramente, se declara la clase pública ”ProfileScreen”, que extiende a la clase de Dart ”StatefulWidget”4.1, esto es, un Widget con estado. Este estado viene representado por la clase privada ”_ProfileState”, que se presenta como un estado de la clase ”ProfileScreen”, hecho que indica la sentencia ”State<ProfileScreen>”.

Dentro de esta clase-estado, se declaran una serie de variables que ayudarán a controlar tanto los elementos representados en la pantalla, como cuándo se representan estos.

Pasamos ahora a analizar la parte de la pantalla que coloca de forma visual el resto de widgets del árbol de widgets en Flutter, y la que escucha los eventos de la pantalla:

```
@override
Widget build(BuildContext context) {
  return BlocProvider<ProfileBloc>(
    create: (context) => ProfileBloc()..add(GetProfile()),
    child: BlocListener<ProfileBloc, ProfileState>(
      listener: (context, state) {
        _blocContext = context;

        /// Get Profile
        if (state.getProfile != null) {
          if (state.getProfile.isLoading()) {
            Log.info('ProfileState: GetProfile Loading');
            _isLoading = true;
          } else if (state.getProfile.isFailure()) {
            Log.info('ProfileState: GetProfile Failure');
            _isLoading = false;
            Message.error(context, S.current.error_getting_profile);
          } else if (state.getProfile.isSuccess()) {
            Log.info('ProfileState: GetProfile Success');
            _isLoading = false;
            Session.updateProfile = false;
            setState(() {
              _profile = state.profile;
            });
            _blocContext.read<ProfileBloc>().add(GetRecent());
          }
        }
```

Figura 5.7: Implementación del listener de los eventos en pantalla definidos en el profile_event. Fuente propia.

De arriba hacia abajo, el Widget llamado ”build”, que sobreescribe el build por defecto del ”_ProfileState”hace lo siguiente:

1. En el parámetro ”**create**”, indica que al ejecutarse este widget, se añada al BLoC el evento ”GetProfile()”.

2. En el parámetro ”**child**”, se establece como Widget hijo al BlocListener, que haciendo uso del ProfileBloc y el ProfileState, comprueba y escucha los cambios de estado provocados en el BLoC por los eventos añadidos al mismo.

3. A continuación, se modifican las variables de la propia pantalla, creadas con anterioridad, según el estado devuelto por el BLoC. Si se el estado es ”**isLoading**”, la variable correspondiente se pone a ”**true**”, si el cambio de estado ha sido exitoso, se almacena mediante la

función "**setState()**" el perfil devuelto por el BLoC en el estado exitoso en la variable de correspondiente, y a continuación se pueden añadir al BLoC los eventos que correspondan, si es que corresponden; en este caso se añade "GetRecent()" porque así lo especifica la lógica de negocio.

Pasemos ahora a analizar la implementación del flujo de este evento, el "Get Profile", desde el punto de vista de otro de los componentes del BLoC, el **profile_state**:

```
class ProfileState {

  final User profile;
  final List<Post> recentPosts;
  final List<Post> myPosts;
  final int folderId;

  final Triple getProfile;
  final Triple getRecent;
  final Triple saveCache;
  final Triple getMyPosts;
  final Triple createFolder;
  final Triple updateFolderPosts;
  final Triple deleteFolder;

  ProfileState({
    this.profile,
    this.recentPosts,
    this.myPosts,
    this.folderId,
    this.getProfile,
    this.getRecent,
    this.saveCache,
```

```
  /// GET PROFILE

  factory ProfileState.getProfileLoading() {
    return ProfileState(
        profile: null,
        getProfile: Constants.blocLoading
    );
  }

  factory ProfileState.getProfileFailure() {
    return ProfileState(
        profile: null,
        getProfile: Constants.blocFailure
    );
  }

  factory ProfileState.getProfileSuccess(User profile) {
    return ProfileState(
        profile: profile,
        getProfile: Constants.blocSuccess
    );
  }
```

Figura 5.8: Implementación del profile_state a nivel de variables y funciones para el método "Get Profile". Fuente propia.

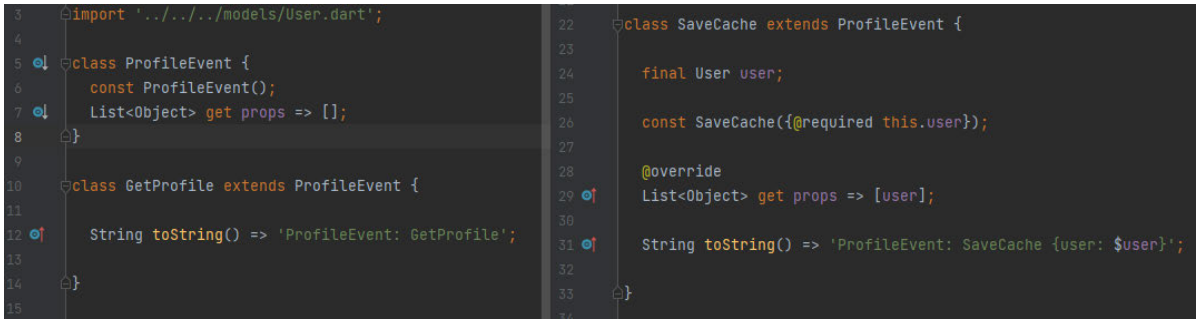
En este componente, podemos observar los siguientes elementos en su clase representativa, ProfileState:

1. Las variables que representarán el estado único de la pantalla del perfil, como son el User o las listas de Post, y variables útiles como el folderId. Las clases propias como User o Post están definidas en la carpeta "models".

2. Variables de tipo "**Triple**", una clase propia definida en la carpeta "models" que nos ayuda a representar mediante constantes los posibles estados de un evento.

3. Los métodos "**factory**"[62], que nos permiten crear distintos métodos constructores para representar los estados asociados a un evento.

A continuación, exploraremos brevemente la implementación de los eventos en el **profile_event**:



```dart
import '../../../models/User.dart';

class ProfileEvent {
  const ProfileEvent();
  List<Object> get props => [];
}

class GetProfile extends ProfileEvent {

  String toString() => 'ProfileEvent: GetProfile';

}
```

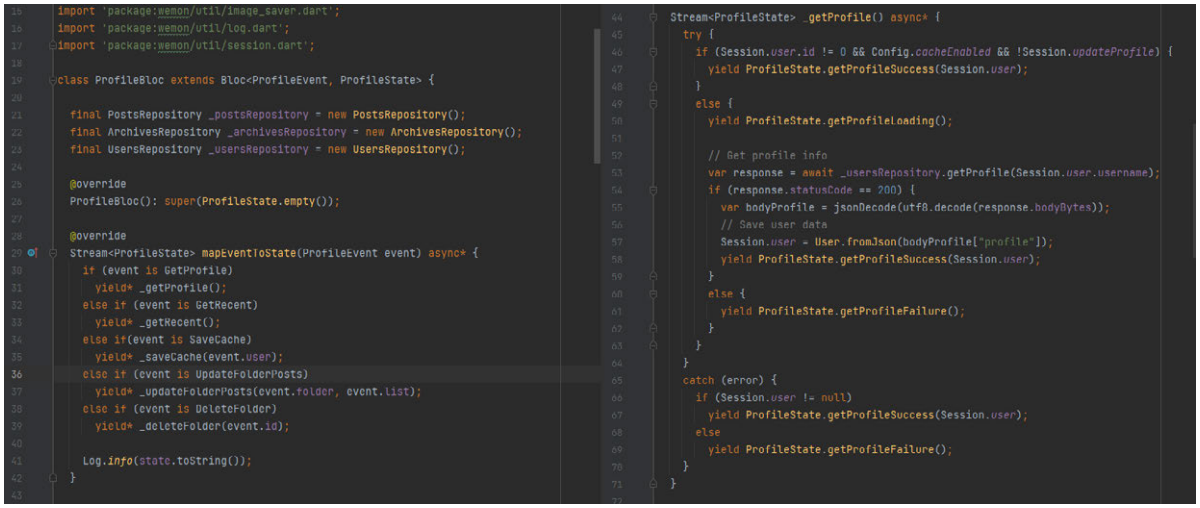
```dart
class SaveCache extends ProfileEvent {

  final User user;

  const SaveCache({@required this.user});

  @override
  List<Object> get props => [user];

  String toString() => 'ProfileEvent: SaveCache {user: $user}';

}
```

Figura 5.9: Implementación del profile_event a nivel de variables y funciones para el método ”Get Profile”. Fuente propia.

Como podemos ver, para representar los eventos asociados al perfil se crea una clase genérica ”ProfileEvent”, que es posteriormente extendida por otra clase ”GetProfile”, propia y única para el evento en cuestión.

Esta clase GetProfile no tiene variables de entrada ya que el evento no las requiere, pero como se puede ver a la derecha en la clase ”SaveCache”, asociada al evento del mismo nombre, se implementa el método ”**get props**” de la clase extendida para devolver el usuario como una de las propiedades del evento, que será posteriormente utilizada en el ”profile_bloc”.

Ahora analicemos el **profile_bloc** para ver cómo estos componentes se relacionan:



```dart
import 'package:wemon/util/image_saver.dart';
import 'package:wemon/util/log.dart';
import 'package:wemon/util/session.dart';

class ProfileBloc extends Bloc<ProfileEvent, ProfileState> {

  final PostsRepository _postsRepository = new PostsRepository();
  final ArchivesRepository _archivesRepository = new ArchivesRepository();
  final UsersRepository _usersRepository = new UsersRepository();

  @override
  ProfileBloc(): super(ProfileState.empty());

  @override
  Stream<ProfileState> mapEventToState(ProfileEvent event) async* {
    if (event is GetProfile)
      yield* _getProfile();
    else if (event is GetRecent)
      yield* _getRecent();
    else if(event is SaveCache)
      yield* _saveCache(event.user);
    else if (event is UpdateFolderPosts)
      yield* _updateFolderPosts(event.folder, event.list);
    else if (event is DeleteFolder)
      yield* _deleteFolder(event.id);

    Log.info(state.toString());
  }
```

```dart
  Stream<ProfileState> _getProfile() async* {
    try {
      if (Session.user.id != 0 && Config.cacheEnabled && !Session.updateProfile) {
        yield ProfileState.getProfileSuccess(Session.user);
      }
      else {
        yield ProfileState.getProfileLoading();

        // Get profile info
        var response = await _usersRepository.getProfile(Session.user.username);
        if (response.statusCode == 200) {
          var bodyProfile = jsonDecode(utf8.decode(response.bodyBytes));
          // Save user data
          Session.user = User.fromJson(bodyProfile["profile"]);
          yield ProfileState.getProfileSuccess(Session.user);
        }
        else {
          yield ProfileState.getProfileFailure();
        }
      }
    }
    catch (error) {
      if (Session.user != null)
        yield ProfileState.getProfileSuccess(Session.user);
      else
        yield ProfileState.getProfileFailure();
    }
  }
```

Figura 5.10: Implementación del profile_bloc a nivel de variables y funciones para el método ”Get Profile”. Fuente propia.

Como podemos ver, el BLoC se desarrolla y unifica en la clase **ProfileBloc**, en la que se declaran las clases de los correspondientes archivos **_repository**. Además esta clase implementa la clase abstracta "Bloc", que obtiene un Stream[63] de ProfileEvent que son transformados en los ProfileState correspondientes.

En la función principal "**mapEventToState**", vemos lo que ocurre realmente tras el patrón BLoC. Si el evento es de tipo GetProfile, se devuelve mediante un Stream el resultado de la función **_getProfile()**.

En esta función **_getProfile()**, primeramente se establece un bloque "try/catch" para controlar errores no esperados. En el bloque "try" se comprueba primeramente que los datos del perfil no estén almacenados en caché. En caso de no estarlo, se hace una llamada a la API mediante el **_userRepository** con su método correspondiente, y si la **respuesta** devuelve un statusCode de 200[64], entonces se procesa el cuerpo de la misma mediante transformaciones de JSON a Dart, para devolver el User mediante el estado de vuelta a la profile_screen.

En la profile_screen, como hemos visto antes, se cambiarán las variables locales como corresponda para representar por pantalla los distintos estados posibles asociados al evento al que se llama.

Ahora repasemos brevemente qué se incluye en el **user_repository** para este evento Get Profile:

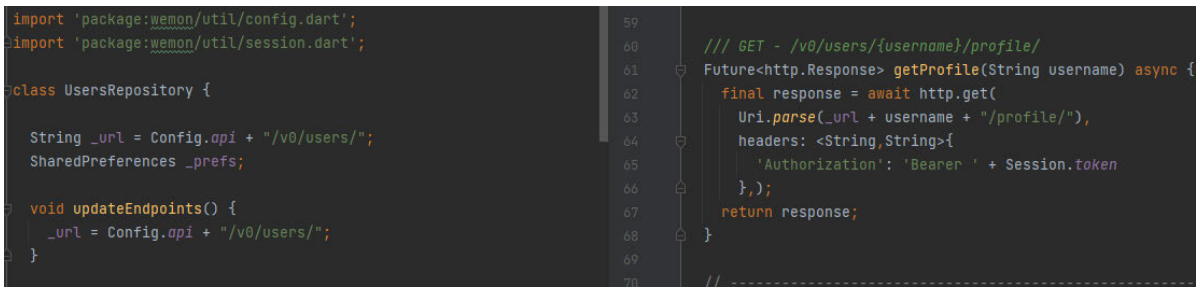


Figura 5.11: Implementación del users_repo a nivel de variables y funciones para el método "Get Profile". Fuente propia.

Como podemos observar, la parte correspondiente a las llamadas a la API se implementa mediante la clase **UsersRepository**, que tiene como variables los distintos endpoints adaptados que parten como base de la configuración de la app presente en la clase "Config" de la carpeta "Util".

Mediante el uso de estas variables se implementa la función utilizada por el evento "Get Profile", en este caso llamada **getProfile()**, que toma como parámetros el nombre de usuario para obtener de la API el usuario correspondiente. Esta llamada requiere el uso de un token de sesión, que se devuele en el momento del Login o inicio de sesión en la app.

```
Container(
    width: iconSize,
    height: iconSize,
    color: AppColors.grayLight),  // Container
(widget.user.image != null && widget.user.image != "")
    ? Image.network(
        widget.user.image,
        width: iconSize,
        height: iconSize,
        fit: BoxFit.cover,
      )  // Image.network
    : Image.asset(
        Constants.defaultImage,
        width: iconSize,
        height: iconSize,
        fit: BoxFit.cover,
      )  // Image.asset
```

Figura 5.12: Implementación de parte del componente de la foto de perfil a raíz de los datos obtenidos tras de ejecución en el BLoC del método "Get Profile". Fuente propia.

Así, finalmente llegamos a la implementación del núcleo uno de los componentes, en este caso de la imagen del usuario en la pantalla del perfil. Este extracto de código se engloba dentro de un widget "Stack"[65], que apila elementos para su posterior representación.

Como podemos observar, se define un widget "Container" con una altura y anchura basadas en constantes predefinidas; posteriormente se apila encima o bien un widget **Image.network**, o uno **Image.asset**, dependiendo de si los datos que se deberían haber recibido en el BLoC se han recibido correctamente, o no.

Este control sobre elementos nulos o vacíos deberá ser realizado a lo largo del desarrollo, cubriendo así todos los posibles casos en los que el estado de una aplicación se pueda manifestar, tanto los exitosos, como los fallidos, y cubriendo también falsos éxitos derivados de problemas en el almacenamiento de datos en el back-end.

5.3 Segundo Caso de Desarrollo: Modificación de una Pantalla Existente

Continuando con la explicación del desarrollo e implementación de los componentes de la aplicación, exploraremos ahora cómo proceder en los casos en los que el desarrollo no busque implementar un componente nuevo desde cero, sino aquel en el que el componente ya exista.

Esta exploración será breve, ya que las bases para el análisis, división de tareas, organización del trabajo e implementación del BLoC ya se sientan en el apartado anterior5.2.

Modificaciones Estéticas

Para llevar a cabo este tipo de modificaciones, es tan sencillo como ejecutar un **emulador** de Android Studio[34], junto con la funcionalidad de **Flutter Inspector**4.1[66] para encontrar en la pantalla correspondiente el elemento a modificar, y usar la funcionalidad de "**Hot Reload**[67] o "**Hot Restart**" a medida que se modifica el elemento en cuestión dentro del árbold de widgets para comprobar que los cambios se realizan correctamente.

Este proceso de pruebas en vivo se repetirá para emuladores de distinto tamaño y formato, para comprobar la correcta anidación de los elementos cambiados en el árbol de widgets de Flutter.

Modificaciones Funcionales

Este tipo de modificaciones suelen ser más costosas, pero siguen las líneas generales de las modificaciones estéticas5.3 a la hora de encontrar el lugar donde implementarlas, así como las guías de implementación del BLoC establecidas en el apartado de implementación de una nueva pantalla5.2.

Lo complejo del asunto muchas veces recae sobre el hecho de que estas modificaciones funcionales arrastran también modificaciones estéticas, lo que en ocasiones implica rehacer toda la estructura de la pantalla.

Afortunadamente, y con la experiencia, este tipo de modificaciones se hacen más livianas.

Modificaciones híbridas

Este tipo de modificaciones requieren cambios estéticos y funcionales.

Exploremos el desarrollo de una de estas modificaciones mediante el desarrollo de la **user_screen**, una pantalla que nace de la necesidad de los usuarios de poder ver los perfiles de otros usuarios.

Para esto, partimos de un diseño básico, al igual que lo hicimos para la profile_screen:

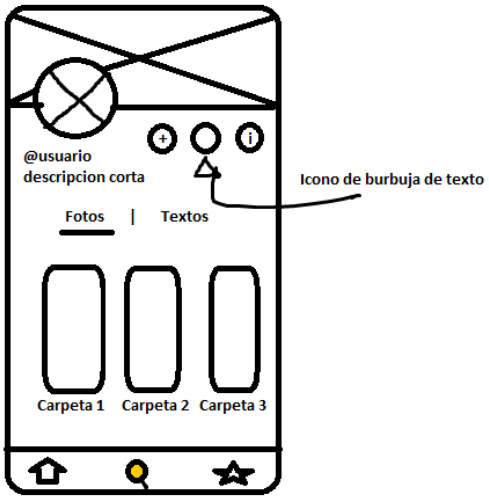


Figura 5.13: Ejemplo de un diseño inicial y funcional de la vista del perfil de otros usuarios. Fuente propia.

En este nuevo diseño podemos observar lo siguiente:

- Para mantener la consistencia visual a lo largo de la aplicación, la forma en la que se presenta el perfil del usuario de la sesión y de otros usuarios es idéntica.
- Se han añadido tres botones nuevos, uno para seguir al usuario, otro para hablarle, además de otro que parece ser de información.

Tras aclararlo con el equipo de diseño en varias reuniones, se confirma que el botón de información permitirá a los usuarios ver datos adicionales de otros usuarios como sus seguidores y segui-

dos en una pantalla modal, y que tendrán podrán realizar acciones adicionales como reportarle en caso de observar una conducta inadecuada dentro de la red social.

Todos estos hechos derivan en las siguientes conclusiones de cara a la planificación del desarrollo:

1. **Abstracción de componentes**: podemos abstraer el componente visual de la profile_para que pueda ser llamado tanto para mostrar la información del usuario de la sesión, como la de otros usuarios.
2. **Aprovechamiento de las variables locales**: esto es, crear nuevas variables que permitan dar o quitar visibilidad a los botones y componentes dependiendo del lugar del que se llame al componente visual.
3. **Priorización de desarrollos futuros**: ya que la pantalla en cuestión deriva en otra funcionalidad compleja, la de las conversaciones con otros usuarios. Es absurdo a nivel de negocio limitar la interacción de los usuarios a que estos puedan ver sus perfiles, sus publicaciones y ya. La gente quiere hablar entre sí.

Podemos observar la abstracción de componentes mediante el componente user_screen, que está compuesto por lo que antes era la parte visual y funcional de la profile_screen y cuyas nuevas llamadas se muestran en la siguiente figura:

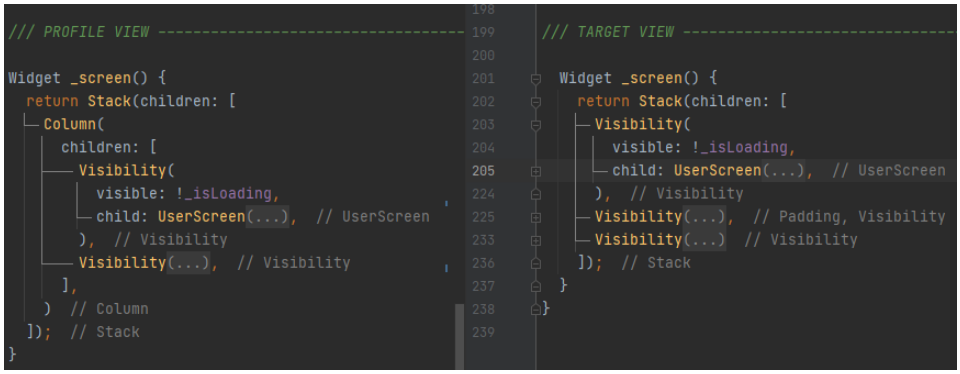


Figura 5.14: Implementación de las llamadas al componente user_screen desde la profile_screen y la target_screen. Fuente propia.

En este caso, la target_screen es la pantalla del target, que es la forma interna con la que nos referimos a los usuarios que son objetivo de una búsqueda.

Por el momento esta abstracción no conlleva cambios funcionales, nuevas variables o controles de visibilidad.

Es importarte seguir el orden de las tareas para evitar confusiones, y tener una mayor trazabilidad de los posibles fallos que surjan durante el desarrollo. Mezclar tareas o alterar su orden, como sería el caso de añadir nuevas variables de control visual mientras intentamos abstraer componentes, puede hacer que las relaciones entre lo que antes era un componente monolítico y ahora son componentes independientes sea difusa, y esto deriva en más trabajo a largo plazo.

Llevar a cabo estos cambios sería el siguiente paso a realizar dentro de este desarrollo, y podemos observarlos en la siguiente figura:

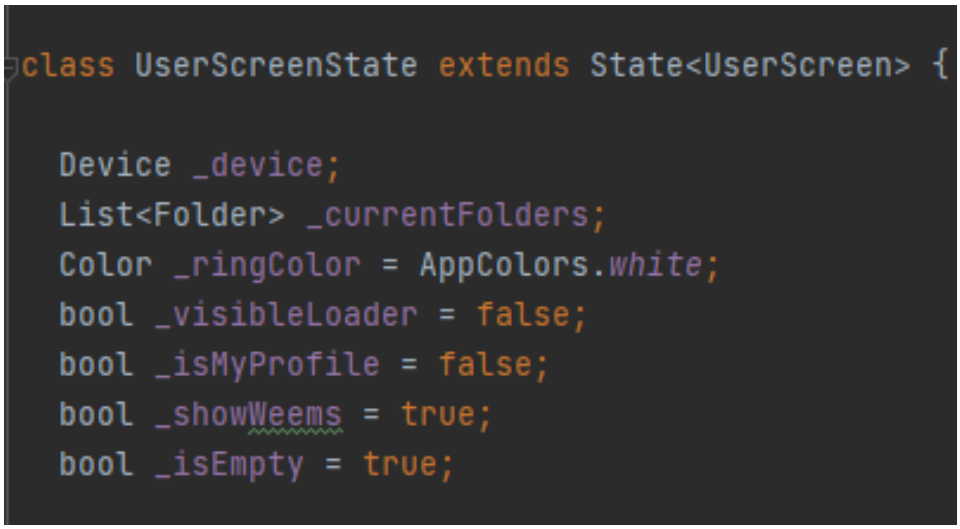


Figura 5.15: Implementación de las variables del componente user_screen. Fuente propia.

Como se observa en la figura, se ha creado una nueva variable **_isMyProfile**, cuyo valor se definirá en la función **initState()** y que es esencial para esta labor de abstracción, ya que es la variable que controla si se muestran o no varios elementos asociados bien a la pantalla del propio usuario de la sesión, o a la pantalla de otro usuario de la aplicación, como podemos ver en la siguiente figura:

```
Widget _actionButtons() {
  return Padding(
    padding: EdgeInsets.only(top: (_device.width(16.5) + Config.imageStroke / 2) + 10),
    child: Row(
      children: [
        Padding(
          padding: const EdgeInsets.only(right: 10),
          child: FloatingActionButton.small(
              heroTag: "btnActionLeft",
              child: Icon(
                _isMyProfile? AppIcons.notifications: widget.user.isFollowed? AppIcons.check: AppIcons.follow,
                color: widget.user.isFollowed? AppColors.white: AppColors.blue,
              ),  // Icon
              backgroundColor: widget.user.isFollowed? AppColors.blue: AppColors.white,
              onPressed: widget.onActionLeftPressed),  // FloatingActionButton.small
        ),  // Padding
        if (!_isMyProfile)
          Padding(
            padding: const EdgeInsets.only(right: 10),
            child: FloatingActionButton.small(
                heroTag: "btnActionCenter",
                child: Icon(...),  // Icon
                backgroundColor: AppColors.white,
                onPressed: widget.onActionCenterPressed),  // FloatingActionButton.small
          ),  // Padding
        Padding(
          padding: const EdgeInsets.only(right: 10),
          child: FloatingActionButton.small(
              heroTag: "btnActionRight",
              child: Icon(
                _isMyProfile ? AppIcons.settings : AppIcons.info,
                color: AppColors.blue,
              ),  // Icon
              backgroundColor: AppColors.white,
              onPressed: widget.onActionRightPressed),  // FloatingActionButton.small
        ),  // Padding
      ],
    ),  // Row
  );  // Padding
}
```

Figura 5.16: Ejemplo de uso de la variable _isMyProfile del componente user_screen para controlar la visibilidad de los botones de la pantalla. Fuente propia.

Ahora podemos ver cómo esa nueva variable derivada del proceso de abstracción de un componente visual, nos ayuda a marcar la visibilidad de unos u otros componentes, en este caso la visibilidad de los botones de acción sobre el perfil de un usuario. Como habíamos definido en la especificación del rediseño, los botones mostrados para el usuario sobre su propio perfil han de ser distintos de los botones que se le muestran a un usuario sobre el perfil de otro; haber definido la variable _isMyProfile nos ha ayudado a lograr la especificación de una forma eficaz.

6. Resultados

Los resultados del esfuerzo derivado de la aplicación de la metodología de desarrollo de aplicaciones multiplataforma explorada en este TFG no son otros que Wemon, la aplicación sujeto de estudio que ha guiado este TFG de principio a fin.

Por concretar un poco más, y centrándonos en la pantalla usada como ejemplo a lo largo de la metodología, podemos ver como resultado funcional la siguiente imagen:

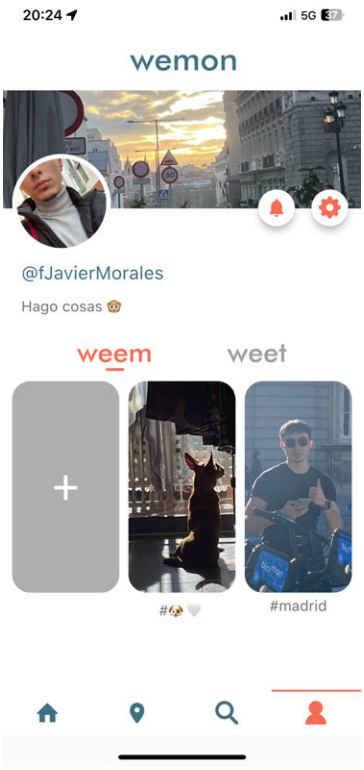


Figura 6.1: Imagen del profile_screen en la versión de desarrollo 1.0.6 de la aplicación Wemon. Fuente propia.

Esta imagen muestra la profile_screen de la aplicación en su versión 1.0.6 para desarrollo. En ella podemos ver cómo los distintos elementos mostrados en la imagen inicial 5.3 se materializan después de haber pasado por varios proceso de desarrollo y refinamiento junto al equipo de diseño, procesos propios de las metodologías ágiles.

De manera adicional, podemos ver que aplicando ciertas modificaciones y patrones a la propia profile_screen, podemos obtener lo que se muestra a continuación:

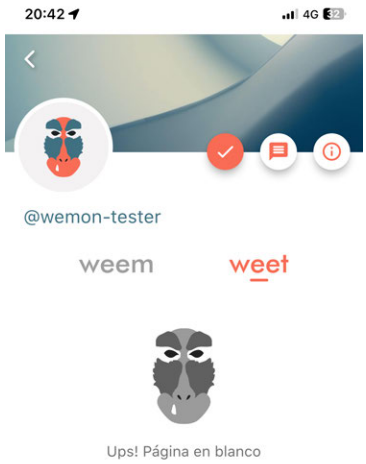


Figura 6.2: Imagen del profile_screen de un usuario ajeno al logado en la versión de desarrollo 1.0.6 de la aplicación Wemon. Fuente propia.

Como se puede ver, la aplicación de modificaciones ligeras a la profile_screen puede permitir ampliar las funcionalidades de la aplicación sin necesidad de rehacer cantidades ingentes de código. Así, reducimos tiempo de desarrollo reutilizando componentes ya existentes. En este caso, los cambios han sido los siguientes:

1. Se ha añadido un botón que indica si se sigue al usuario visualizado o no.
2. Se ha añadido un botón que permite entablar una conversación directa con el otro usuario.

3. Se ha añadido un botón de información, que permite incorporar acciones adicionales sobre el usuario visualizado como reportes en caso de un uso indebido de la aplicación, la posibilidad de bloquear al usuario o visualizar su numero de seguidores y de usuarios seguidos.

4. Se han hecho invisibles los botones de ajustes y notificaciones, ya que esas funcionalidades solo son accesibles para el propio usuario, y tiene sentido que acceda a ellas desde su perfil.

Además, entre los resultados de este trabajo se encuentra lo que pretende ser una guía básica de gestión de comunicación entre equipos y del trabajo, además de un ejemplo práctico del patrón BLoC con un contexto completo, para que quien desee leer este TFG pueda hacerlo y encontrar en él un recurso didáctico valioso.

7. Impacto Social y Medioambiental

7.1 Impacto Social

El impacto social derivado del desarrollo de Wemon pretende ser positivo, al tratarse de un modelo de red social basado en planes y que, por lo tanto, fomenta que la gente se conozca en persona y utilice la red social como el medio a un fin, que es socializar y crear conexiones relevantes para su vida social.

Adicionalmente, se espera que este TFG sirva como guía práctica e inspiración para que otros ingenieros, desarrolladores y gente que se quiera embarcar en el mundo del software pueda hacerlo sin sentirse especialmente perdido, bien por falta de recursos como puede ser el caso de los disponibles para tecnologías relativamente nuevas, o por dispersión de recursos, lo que dificulta el acceso a cualquier disciplina al dar una apariencia a las habilidades de entrada de muro inexpugnable.

7.2 Impacto Medioambiental

No se espera que este TFG tenga impacto medioambiental alguno, mas allá de los efectos potencialmente positivos derivados del impacto que este TFG tenga sobre el potencial desarrollo de futuras aplicaciones relacionadas con el medioambiente.

8. Líneas de Investigación

Tras realizar este Trabajo de Fin de Grado, hay ciertos conceptos que han quedado fuera del alcance del mismo, y que sería interesante desarrollar en profundidad en un futuro.

En primer lugar, sería interesante explorar la implementación de los patrones seguidos en el desarrollo del back-end, así como la forma en la que se realizan las pruebas de carga en ese lado de la aplicación.

Otro aspecto interesante a tratar es la utilización de ciertas características de Firebase como es el Remote Config, que permite configurar variables de forma remota para grupos de usuarios determinados, lo cual nos permite hacer como desarrolladores cuestiones tan interesantes como habilitar ciertas funciones experimentales a grupos muy reducidos de usuarios, y así controlar los posibles errores que estas puedan dar, y si es necesario revertir el acceso a estas nuevas características de forma parcial o global sin necesidad de los usuarios se bajen una versión específica de la aplicación que active esa característica de forma irrevocable.

Un ejemplo práctico de el uso de esta característica de Firebase fue la actualización que permite a los usuarios enviarse mensajes entre ellos. En un principio habilitamos esta nueva característica para el 10% de los usuarios de la aplicación. Tras ver que no había problemas para los usuarios que la usaban, decidimos lanzarla para el 100% de los usuarios que la estaban probando.

Por último, otra posible área a explorar sería la gestión de incidencias masivas que derivó del lanzamiento de la aplicación al público general, lo que forzó al equipo a replantearse sus métodos de organización y a forzar unos estándares más estrictos de definición de tareas basada en User Stories, concepto propio de ciertas metodologías ágiles.

9. Conclusiones

9.1 Conclusiones Generales

A nivel general, el desarrollo de este TFG permitirá expandir sobre el concepto didáctico que se le ha dado al mismo para crear trabajos que mejoren de manera iterativa un concepto que clave como es la parte práctica de una buena instrucción académica.

En primer lugar, se establecen los objetivos del proyecto, que no son otros que explorar la aplicación de conceptos de ingeniería de software a nivel de desarrollo y organización de equipos a través de Wemon, una red social enfocada a planes en la vida real. Posteriormente se exploran las restricciones que rodean a esta empresa y el razonamiento tras las decisiones del equipo en materia organizativa y tecnológica, explicando brevemente los conceptos arquitectónicos sobre los que esta se basa, y cuya implementación con el framework mutiplataforma Flutter se explica en profundidad dando un ejemplo, el de la pantalla que muestra la información de los usuarios dentro de Wemon.

Los resultados reales de todo este proceso, detallado en la metodología, pueden ser vistos en el apartado de resultados, así como en la propia aplicación descargable tanto para iOS como para Android, a través de sus respectivas plataformas oficiales de distribución de aplicaciones.

Además, al ser Flutter un framework relativamente nuevo, este trabajo sirve como marco de referencia para el desarrollo de aplicaciones multiplataforma, tanto en equipos pequeños que es en lo que se enfoca este proyecto, como en equipos más grandes que descubran este trabajo académico.

9.2 Conclusiones Personales

El desarrollo de este TFG lleva a cabo mi deseo de dotar a compañeros del sector de herramientas suficientes como para embarcarse en su propia aventura de desarrollo de una forma mínimamente

organizada.

Espero que este TFG y otros con la misma intención, inspiren un cambio en el que se promueva de forma activa la participación del alumnado en proyectos en equipo, y que así no tengan que esperar a ver algo para trabajar con ello. Además considero que hacer este TFG me ha ayudado a tener una idea más claramente estructurada de la organización del equipo con el que llevo casi tres años de mi vida, motivándome a seguir adelante con el proyecto incluso en momentos en los que mi vida personal se complicaba por cuestiones académicas y personales.

De forma personal, mi intención con el desarrollo de Wemon como aplicación es que esta pueda llevar a un escenario en el que las redes sociales sirvan como medio para unir a gente, y no como el espacio donde deciden unirse. Creo que pasar horas tras un modelo de interacciones que no se ajusta a cómo estas se dan en la realidad distorsiona la percepción de la gente y la hace evadirse de problemas que probablemente tengan solución tras una capa de anonimato que, además, nos permite filtrar la esencia que transmitimos al resto.

Bibliografía

[1] E. Ortiz-Ospina, «The rise of social media», *Our World in Data*, 2019, https://ourworldindata.org/rise-of-social-media.

[2] Statista, *Annual number of global mobile app downloads*, https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/, Accedido el 8 de junio de 2023.

[3] Statista, *Number of internet users worldwide*, https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/, Accedido el 8 de junio de 2023.

[4] Statista, *Global social networks ranked by number of users*, https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/, Accedido el 8 de junio de 2023.

[5] Statista, *Impact of social media on mental health*, https://www.statista.com/chart/19262/impact-of-social-media-on-mental-health/, Accedido el 8 de junio de 2023.

[6] C. R. Center, *2016 Cyberbullying Data*, https://cyberbullying.org/2016-cyberbullying-data, Accedido el 8 de junio de 2023.

[7] Statista, *BeReal app downloads worldwide*, https://www.statista.com/statistics/1338262/bereal-app-downloads-worldwide/, Accedido el 8 de junio de 2023.

[8] IEBSchool, *¿Qué son metodologías ágiles? Agile y Scrum*, https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/, Accedido el 8 de junio de 2023.

[9] Salesforce, *¿Qué son metodologías ágiles y cómo pueden ayudar a tus equipos de trabajo?*, https://www.salesforce.com/mx/blog/2021/12/que-son-metodologias-agiles-y-como-pueden-ayudar-a-tus-equipos-de-trabajo.html, Accedido el 8 de junio de 2023.

[10] H. Takeuchi e I. Nonaka, «The New New Product Development Game», *Harvard Business Review*, vol. 64, nº 1, págs. 137-146, 1986.

[11] P. Ágiles, *Historia de Scrum*, https://proyectosagiles.org/historia-de-scrum/, Accedido el 8 de junio de 2023.

[12] Asana, *What is Kanban?*, https://asana.com/es/resources/what-is-kanban, Accedido el 8 de junio de 2023.

[13] Asana, *ScrumBan: The perfect combination of Scrum and Kanban*, https://asana.com/es/resources/scrumban, Accedido el 8 de junio de 2023.

[14] E. B. School, *¿Qué es diseño gráfico?*, https://www.esneca.com/blog/que-es-diseno-grafico/, Acceso: 25 de junio de 2023.

[15] R. Content, *UI y UX: la guía completa para entender diseño de interfaces*, https://rockcontent.com/es/blog/ui-ux/, Accedido el 8 de junio de 2023.

[16] I. Online, *Leyes de la Gestalt: cómo influyen en el diseño gráfico*, https://www.ilerna.es/blog/fp-a-distancia/3d/leyes-de-la-gestalt/, Accedido el 8 de junio de 2023.

[17] Anasaci, *La iconografía en el diseño gráfico*, https://anasaci.com/blog/disenadores/diseno_grafico/la-iconografia-en-el-diseno-grafico.html, Accedido el 8 de junio de 2023.

[18] P. Hackers, *5 buenas prácticas de UI para mejorar el diseño de tu producto*, https://producthackers.com/es/blog/5-buenas-practicas-ui, Accedido el 8 de junio de 2023.

[19] A. Solca, *6 principios fundamentales de UX que debes conocer*, https://adriansolca.medium.com/6-principios-fundamentales-de-ux-que-debes-conocer-6ddff96b61ad, Accedido el 8 de junio de 2023.

[20] *Laws of UX*, https://lawsofux.com/, Accedido el 8 de junio de 2023.

[21] *¿Qué es un wireframe para un sitio web?*, https://www.lucidchart.com/pages/es/que-es-un-wireframe-para-un-sitio-web, Accedido el 8 de junio de 2023.

[22] Shopify, *¿Qué es un wireframe?*, https://www.shopify.com/es/blog/que-es-un-wireframe, Accedido el 8 de junio de 2023.

[23] *Wireframes: ¿Qué son y para qué sirven?*, https://formiux.com/wireframes-que-son-y-para-que-sirven/, Accedido el 8 de junio de 2023.

[24] Flutter, *Flutter Showcase*, https://flutter.dev/showcase, consultado en 2023.

[25] Flutter, *Architectural Overview*, https://docs.flutter.dev/resources/architectural-overview, consultado en 2023.

[26] Flutter, *Introduction to Widgets*, https://docs.flutter.dev/ui/widgets-intro, consultado en 2023.

[27] Flutter, *Animations*, https://docs.flutter.dev/ui/animations, consultado en 2023.

[28] Flutter, *Gestures*, https://docs.flutter.dev/ui/advanced/gestures, consultado en 2023.

[29] Flutter, *Material Widgets*, https://docs.flutter.dev/ui/widgets/material, consultado en 2023.

[30] Flutter, *Cupertino Widgets*, https://docs.flutter.dev/ui/widgets/cupertino, consultado en 2023.

[31] Google, *Introducción a Android Studio*, https://developer.android.com/studio/intro?hl=es-419, consultado en 2023.

[32] Gradle, *Gradle User Guide*, https://docs.gradle.org/current/userguide/userguide.html, consultado en 2023.

[33] JetBrains, *Android Studio Plugins*, https://plugins.jetbrains.com/androidstudio, consultado en 2023.

[34] Google, *Emulador de Android*, https://developer.android.com/studio/run/emulator?hl=es-419, consultado en 2023.

[35] Apple Inc., *Xcode Features*, https://developer.apple.com/xcode/features/, consultado en 2023.

[36] *Firebase Analytics*, https://firebase.google.com/products/analytics?authuser=0&hl=es, Accedido el 8 de junio de 2023.

[37] *Firebase Remote Config*, https://firebase.google.com/products/remote-config?authuser=0&hl=es, Accedido el 8 de junio de 2023.

[38] *Firebase Cloud Messaging*, https://firebase.google.com/products/cloud-messaging?authuser=0&hl=es, Accedido el 8 de junio de 2023.

[39] *Firebase Authentication*, https://firebase.google.com/products/auth?authuser=0&hl=es, Accedido el 8 de junio de 2023.

[40] *AWS Marketplace - Soluciones para todos los casos de uso*, https://aws.amazon.com/marketplace/solutions/awsmp-all-use-cases?linkId=71922778&sc_campaign=AWS_Marketplace&sc_category=AWS+Marketplace&sc_channel=sm&sc_content=MP_UseCasesAll&sc_country=Marketplace&sc_geo=GLOBAL&sc_outcome=awareness&sc_publisher=TWITTER&trk=NA_TWITTER&trkCampaign=MP_UseCasesAll, Accedido el 8 de junio de 2023.

[41] *AWS - Data Lakes and Analytics*, https://aws.amazon.com/es/big-data/datalakes-and-analytics/?hp=tile&tile=solutions, Accedido el 8 de junio de 2023.

[42] *AWS - Machine Learning*, https://aws.amazon.com/es/machine-learning/?hp=tile&tile=solutions, Accedido el 8 de junio de 2023.

[43] *AWS - Serverless*, https://aws.amazon.com/es/serverless/?hp=tile&tile=solutions, Accedido el 8 de junio de 2023.

[44] *AWS - Almacenamiento*, https://aws.amazon.com/es/products/storage/?hp=tile&tile=solutions, Accedido el 8 de junio de 2023.

[45] *Heroku Managed Data Services*, https://www.heroku.com/managed-data-services, Accedido el 8 de junio de 2023.

[46] *Heroku Platform*, https://www.heroku.com/platform, Accedido el 8 de junio de 2023.

[47] *GitHub Docs - About Projects*, https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects, Accedido el 8 de junio de 2023.

[48] *GitHub Docs - Customizing views in your project*, https://docs.github.com/en/issues/planning-and-tracking-with-projects/customizing-views-in-your-project, Accedido el 8 de junio de 2023.

[49] *¿Qué es Slack?*, https://slack.com/intl/es-es/help/articles/115004071768, Accedido el 8 de junio de 2023.

[50] *Slack Apps*, https://socios-programadores.slack.com/intl/es-es/apps, Accedido el 8 de junio de 2023.

[51] *Google Meet - Centro de ayuda*, https://support.google.com/meet/?hl=es#topic=7306097, Accedido el 8 de junio de 2023.

[52] *Telegram - FAQ*, https://telegram.org/faq, Accedido el 8 de junio de 2023.

[53] *WhatsApp*, https://www.whatsapp.com/?lang=es, Accedido el 8 de junio de 2023.

[54] F. Community, *Client-Server vs Peer-to-Peer Networks: What's the Difference?*, https://community.fs.com/es/blog/client-server-vs-peer-to-peer-networks.html, Accedido el 8 de junio de 2023.

[55] E. Britannica, *Client-Server Architecture*, https://www.britannica.com/technology/client-server-architecture, Accedido el 8 de junio de 2023.

[56] IBM, *Benefits of Using a Client-Server Architecture*, https://www.ibm.com/docs/en/rsas/7.5.0?topic=patterns-benefits-using, Accedido el 8 de junio de 2023.

[57] L. Mind, *Bloc Pattern: ¿Qué es y cómo funciona?*, https://leanmind.es/es/blog/bloc-pattern/, Accedido el 8 de junio de 2023.

[58] F. Angelov, *flutter$_b$loc*, https://pub.dev/packages/flutter_bloc, Accedido el 8 de junio de 2023.

[59] *AWS Caching*, https://aws.amazon.com/es/caching/, Accedido el 8 de junio de 2023.

[60] *Preguntas frecuentes sobre internacionalización (i18n)*, https://www.w3.org/International/questions/qa-i18n.es, Accedido el 8 de junio de 2023.

[61] R. Prathiba y S. Suresh, *A Study on Software Metrics*, https://airccse.org/journal/ijsea/papers/3112ijsea16.pdf, Accedido el 8 de junio de 2023, 2016.

[62] R. Guru, *Factory Method*, https://refactoring.guru/design-patterns/factory-method, Accedido el 8 de junio de 2023.

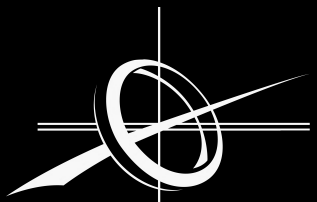
[63] Flutter, *Stream class - dart:async library - Dart API*, https://api.flutter.dev/flutter/dart-async/Stream-class.html, Accedido el 8 de junio de 2023.

[64] Mozilla, *HTTP/1.1: Status Code Definitions - 200 OK*, https://developer.mozilla.org/es/docs/Web/HTTP/Status/200, Accedido el 8 de junio de 2023.

[65] Flutter, *Stack class - widgets library - Flutter API*, https://api.flutter.dev/flutter/widgets/Stack-class.html, Accedido el 8 de junio de 2023.

[66] Flutter, *Inspector - Flutter DevTools*, https://docs.flutter.dev/tools/devtools/inspector, Accedido el 8 de junio de 2023.

[67] Flutter, *Hot Reload - Flutter Documentation*, https://docs.flutter.dev/tools/hot-reload, Accedido el 8 de junio de 2023.


Universidad
Politécnica
de Madrid
ETSI SISTEMAS
INFORMÁTICOS