UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Migración de aplicaciones nativas a desarrollo multiplataforma: estudio de caso aplicado a una aplicación para plataforma de televisión

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Sánchez García, Claudia

Tutor/a: Andrés Martínez, David de

Cotutor/a: Garrido Tejero, Antonio

Cotutor/a externo: Cuervas Maria, Javier

CURSO ACADÉMICO: 2023/2024

Resumen

El presente trabajo se centra en la migración de aplicaciones móviles desde entornos de desarrollo nativo hacia entornos multiplataforma, con un enfoque particular en la adaptación de una aplicación destinada a una plataforma de televisión. Este estudio se llevó a cabo en colaboración con el equipo de desarrollo de aplicaciones de Alten España, en el marco de unas prácticas profesionales. El objetivo principal fue analizar las alternativas tecnológicas existentes para el desarrollo móvil, evaluando la viabilidad de migrar aplicaciones nativas de Android e iOS a un entorno multiplataforma. Como parte del proyecto, se desarrolló un prototipo que integra los módulos más relevantes, garantizando la consistencia y compatibilidad entre el desarrollo multiplataforma y el desarrollo web. A lo largo del estudio, se consideraron diversos factores como la disponibilidad de bibliotecas, el rendimiento, la compatibilidad y la escalabilidad. Se espera que este trabajo sirva como guía para la empresa en futuros análisis y desarrollos multiplataforma, proporcionando una base sólida para la toma de decisiones estratégicas en este ámbito.

Palabras clave: Aplicaciones móviles, Desarrollo multiplataforma, Migración de aplicaciones, Metodologías ágiles, React Native

Resum

Aquest treball se centra en la migració d'aplicacions mòbils des d'entorns de desenvolupament natiu cap a entorns multiplataforma, amb un enfocament particular en l'adaptació d'una aplicació destinada a una plataforma de televisió. Aquest estudi es va dur a terme en col·laboració amb l'equip de desenvolupament d'aplicacions d'Alten España en el marc d'unes pràctiques professionals. L'objectiu principal va ser analitzar les alternatives tecnològiques existents per al desenvolupament mòbil, avaluant la viabilitat de migrar aplicacions natives d'Android i iOS a un entorn multiplataforma. Com a part del projecte, es va desenvolupar un prototip que integra els mòduls més rellevants, garantint la consistència i compatibilitat entre el desenvolupament multiplataforma i el desenvolupament web. Al llarg de l'estudi, es van considerar diversos factors com la disponibilitat de biblioteques, el rendiment, la compatibilitat i l'escalabilitat. Es preveu que aquest treball serveixi com a guia per a l'empresa en futures anàlisis i desenvolupaments híbrids, proporcionant una base sòlida per a la presa de decisions estratègiques en aquest àmbit.

Paraules clau: Aplicacions mòbils, Desenvolupament multiplataforma, Migració d' aplicacions, Metodologies àgils, React Native

Abstract

This work focuses on the migration of mobile applications from native development environments to cross-platform environments, with a particular focus on adapting an application intended for a television platform. This study was conducted in collaboration with the application development team at Alten España as part of a professional internship. The main objective was to analyze the existing technological alternatives for mobile development, evaluating the feasibility of migrating native Android and iOS applications to a cross-platform environment. As part of the project, a prototype was developed, integrating the most relevant modules and ensuring consistency and compatibility between cross-platform and web development. Various factors were considered throughout the study, including the availability of libraries, performance, compatibility, and scalability.

This work is intended to serve as a guide for the company in future analyses and hybrid developments, providing a solid foundation for strategic decision-making in this area.

Key words: Mobile Apps, Cross-Platform Development, App Migration, Agile Methodologies, React Native

Índice general

Índice de figuras

Índice de tablas

Acrónimos

AEP Adobe Experience Platform

AMPS Advanced Mobile Phone System

API Application Programming Interface

CDMA2000 Code Division Multiple Access 2000

CI/CD Integración Continua y Entrega Continua

CPU Unidad Central de Procesamiento

CSS Cascading Style Sheets

DAFO Debilidades, Amenazas, Fortalezas y Oportunidades

EDGE Enhanced Data Rates for GSM Evolution

GPRS General Packet Radio Service

GPS Global Positioning System

GSM Global System for Mobile Communications

HSCSD High-Speed Circuit-Switched Data

HTML HyperText Markup Language

HTML5 HyperText Markup Language version 5

IDE Integrated Development Environment

IMT International Mobile Telecommunications

IP Internet Protocol

JDK Java Development Kit

JVM Java Virtual Machine

LTE Long-Term Evolution

MCDM Multiple Criteria Decision Making

MVP Minimum Viable Product

NMT Nordic Mobile Telephone

NTT Nippon Telegraph and Telephone

PDC Personal Digital Cellular

PMI Project Management Institute

PWA Progressive Web App

QA Quality Assurance

RAM Random Access Memory

SDK Software Development Kit

SMS Short Message Service

UI User Interface

UX User Experience

UMTS Universal Mobile Telecommunications System

URL Uniform Resource Locator

UIT Unión Internacional de Telecomunicaciones (ITU en inglés)

XP Extreme Programming

CAPÍTULO 1
Introducción

1.1 Introducción

En el mundo actual de desarrollo de aplicaciones móviles, la migración de tecnologías nativas a multiplataforma (del inglés *cross-platform*) es un tema relevante por sus beneficios en cuanto a eficiencia y mantenimiento [10]. Los desarrollos nativos tienen la ventaja de brindar un rendimiento óptimo y una experiencia más fluida, al estar diseñados específicamente para la plataforma en la que se ejecutan, además de un completo acceso a funcionalidades del dispositivo. Por otro lado, los desarrollos multiplataforma, en general, aprovechan las tecnologías HyperText Markup Language (HTML), Cascading Style Sheets (CSS) y JavaScript para crear aplicaciones que se pueden ejecutar en múltiples plataformas con un único código base; lo que reduce significativamente el tiempo de desarrollo y el mantenimiento. Sin embargo, los desarrollos multiplataforma pueden presentar problemas de rendimiento y compatibilidad, sobre todo en aplicaciones de alto grado de procesamiento o interacción con el hardware.

1.2 Motivación

Este Trabajo de Máster se enmarca en un proyecto del que formo parte, en el grupo Alten España[1], donde se trata la migración de aplicaciones móviles de desarrollo nativo a multiplataforma. La problemática radica en la necesidad de mejorar la eficiencia del desarrollo y reducir los costes de mantenimiento sin comprometer la calidad y experiencia del usuario. El alcance del proyecto incluye la evaluación de las aplicaciones existentes, identificación de funcionalidades críticas y fundamentalmente la selección de la mejor estrategia de migración. Se consideran también aspectos de rendimiento, compatibilidad y escalabilidad para garantizar que la solución cumpla con los estándares requeridos. Con esto se pretende aportar a la comunidad de desarrollo nativo un caso práctico y su análisis para guiar este tipo de migraciones muy comunes en el sector, abordando el proceso de toma de decisiones para lograr que la transición sea lo más fluida posible minimizando así el impacto en la experiencia de usuario y el negocio.

1.3 Objetivos

1. Analizar alternativas tecnológicas actuales para el desarrollo móvil nativo, híbrido y web para determinar sus principales ventajas e inconvenientes.

[1] https://www.alten.es/

2. Evaluar la viabilidad de migrar a desarrollo multiplataforma las aplicaciones nativas Android e iOS del caso práctico.

3. Implementar la migración de módulos nativos, considerando la disponibilidad de bibliotecas y SDK (del inglés *Software Development Kit*).

4. Explorar opciones de arquitecturas y su aplicabilidad en el proyecto.

5. Establecer metodologías de trabajo para la ejecución del proyecto, implementación del prototipo y realización de pruebas.

6. Desarrollar un MVP (del inglés *Minimum Viable Product*) que sirva como prototipo, integrando los módulos más importantes y verificando su correcto funcionamiento, manteniendo coherencia con el diseño UX/UI (del inglés *User Experience/User Interface*) propuesto.

7. Garantizar la compatibilidad y consistencia entre el desarrollo móvil multiplataforma y el proyecto de programación web paralelo.

1.4 Estructura de la memoria

La memoria de este trabajo esta dividido en seis capítulos, siendo este, la Introducción, el primero, donde se expone el contexto de la migración y los principales objetivos del trabajo.

El contenido principal de la memoria se encuentra en los 4 capítulos siguientes:

- Capítulo 2 - Tecnologías: Detalla el contexto actual del desarrollo móvil. Dividido en secciones donde se describe la historia de los primeros dispositivos, hasta la actualidad; las herramientas y lenguajes en los que se desarrolla específicamente para dispositivos Android e iOS; y finalmente el contexto del actual del desarrollo multiplataforma, sus enfoques y frameworks más utilizados.

- Capítulo 3 - Análisis del caso práctico: Presenta el contexto real en el que se desarrolla la migración. El análisis aborda la problemática existente y los para decidir migrar a un desarrollo multiplataforma.

- Capítulo 4 - Evaluación y selección de estrategias: Se centra en la comparación de diferentes frameworks multiplataforma, como Flutter y React Native. Se abordan aspectos cualitativos, rendimiento y disponibilidad de bibliotecas. Además, se detalla la selección del framework más adecuado según los requisitos del proyecto. La segunda parte del capítulo discute las metodologías de trabajo, como la gestión de recursos y riesgos, planificación, y metodologías ágiles como Scrum.

- Capítulo 5 - Implementación y desarrollo: Se describe el proceso de implementación del prototipo multiplataforma. Se explica el uso de las tecnologías seleccionadas y el proceso de integración de las bibliotecas; incluye detalles técnicos y pruebas realizadas. Las tareas presentadas representan mi trabajo en el equipo, la integración de las bibliotecas relacionadas con el reproductor de medios. Otros aspectos como la estructura del código, los componentes de interfaz de la aplicación, que no están relacionados con el reproductor de medios, son tareas de otros miembros del equipo y no se detallan en este trabajo.

Por último el capítulo 6, Conclusiones, resume los resultados del proyecto, las dificultades encontradas durante la migración y los éxitos logrados. Se incluyen recomendaciones para futuras implementaciones y desarrollos multiplataforma.

Esta estructura asegura que cada capítulo siga una lógica clara, partiendo del contexto general, pasando por el análisis técnico y concluyendo con las lecciones aprendidas y recomendaciones.

CAPÍTULO 2
Tecnologías

En este capítulo, exploraremos el panorama actual del desarrollo móvil, centrándonos en dos enfoques principales: el desarrollo de aplicaciones nativas y el desarrollo multiplataforma. Comenzaremos examinando la evolución de la tecnología móvil desde sus inicios hasta la actualidad. Luego, nos sumergiremos en el desarrollo de aplicaciones nativas, explorando las herramientas y lenguajes utilizados en plataformas como Android e iOS. Posteriormente, en el desarrollo multiplataforma, analizando las diferentes aproximaciones y herramientas disponibles, así como sus ventajas y desafíos. A lo largo de este capítulo, se proporcionará un panorama completo de las opciones disponibles para los desarrolladores móviles en la actualidad.

2.1 Tecnología móvil

La tecnología móvil se refiere a los dispositivos que son fácilmente transportables y están diseñados para facilitar la comunicación bidireccional del usuario. Hoy en día, la tecnología móvil se caracteriza por la omnipresencia de dispositivos con acceso a Internet, como los teléfonos inteligentes y las tabletas. Además de estos dispositivos, también incluye una variedad de otros aparatos como localizadores bidireccionales, computadoras portátiles, teléfonos móviles, dispositivos de navegación con sistema de posicionamiento global (del inglés *Global Positioning System (GPS)*), entre otros. La conectividad que poseen los dispositivos gracias a las redes de comunicación inalámbrica, permiten la transmisión de llamadas, datos y ejecución de aplicaciones [25].

Con la llegada de la tecnología móvil, la sociedad experimentó una transformación radical en la forma en que se comunicaba y se relacionaba. Antes limitada por la dependencia del correo postal, telegramas y teléfonos fijos, la aparición de dispositivos móviles trajo consigo una revolución en la comunicación, permitiendo a las personas conectarse de manera instantánea y ubicua. Para comprender mejor esta evolución, es crucial explorar la historia y el desarrollo de la tecnología móvil desde sus inicios hasta la actualidad.

2.1.1. Origen

Las telecomunicaciones de voz con tecnología fija, están presentes en nuestras vidas desde su aparición en 1876 [1]. A pesar de este avance en la comunicación no fue hasta años más tarde que se incorporaba la primera forma de tecnología móvil en la sociedad.

Durante la segunda guerra mundial se evidenció esta gran necesidad de comunicaciones y movilidad, por lo que se construyeron los primeros medios inalámbricos de telecomunicaciones y los primeros terminales construidos tenían gran volumen y peso, razón

por lo que se instalaban en vehículos. Entre 1970 y 1973 Martin Cooper, considerado "El Padre" de la telefonía celular, construyó el primer teléfono celular, que era analógico de gran peso y volumen, y la batería apenas duraba 20 minutos [1].

Los sistemas de 1G, como el Advanced Mobile Phone System (AMPS) en Estados Unidos y el Nordic Mobile Telephone (NMT) en Europa, proporcionaban servicios de voz basados en transmisión de radio analógica, pero carecían de la capacidad de interconexión internacional [6].

En 1983 sale al mercado el primer teléfono móvil comercial, el Motorola DynaTAC 8000X [1]; aunque era un dispositivo pesado y costoso marca un hito en la historia.

La tecnología 1G, a pesar de sus limitaciones iniciales, representó un avance significativo en la historia de las comunicaciones móviles.

2.1.2. Evolución

La segunda generación (2G) marcó la transición a la tecnología digital en la década de 1990. Surgieron varios estándares, como Global System for Mobile Communications (GSM) en Europa, Personal Digital Cellular (PDC) en Japón y sistemas como IS-54 y IS-95 en América del Norte. Estas tecnologías permitieron la transmisión de mensajes de texto (Short Message Service (SMS)) y datos a velocidades de hasta 9.6Kbps. A finales de los 90, se introdujeron mejoras significativas, conocidas como 2.5G, estas eran General Packet Radio Service (GPRS), High-Speed Circuit-Switched Data (HSCSD) y Enhanced Data Rates for GSM Evolution (EDGE), mejorando las tasas de transmisión de datos [1].

Para converger servicios de voz y datos surgió la tercera generación (3G), brindando acceso a internet inalámbrico, aplicaciones multimedia y altas transmisiones de datos. Se estandarizó con la tecnología International Mobile Telecommunications (IMT)-2000 por la Unión Internacional de Telecomunicaciones (ITU en inglés) (UIT). Tecnologías como Universal Mobile Telecommunications System (UMTS) en Europa y Code Division Multiple Access 2000 (CDMA2000) en América del Norte fueron pioneras en esta generación. La primera red comercial 3G fue lanzada por Nippon Telegraph and Telephone (NTT) DoCoMo en Japón en 2001, en Europa y Estados Unidos se registran en 2003. Esta tecnología significó un avance en la potencia de las antenas, permitiendo una mayor velocidad, movilidad, permitiendo el roaming de voz y mensajes SMS. Así mismo se introduce la SIM Card (del inglés *Suscriber Identity Module Card*) la cual fue adoptada también en la tecnología 2G [1].

La cuarta generación (4G), iniciada en 2010, se caracteriza por ser todo Internet Protocol (IP) (All-IP), brindando una plataforma común para todas las tecnologías móviles. Con estándares como Long-Term Evolution (LTE), se ofrecen velocidades de hasta 100Mbps para usuarios móviles y hasta 1Gbps para usuarios fijos. Aunque se comercializan como 4G, algunas implementaciones son técnicamente 3.9G, como LTE-Advanced que cumple el estándar 4G [1].

2.1.3. Actualidad

La evolución de la tecnología permitió un aumento en la velocidad y transmisión de datos, propiciando la aparición de nuevas funcionalidades para estos dispositivos que ahora serían más potentes.

En la actualidad, estas tecnologías se continúan utilizando, y entonces nos presentan los siguientes tipos de redes móviles [25]:

- **Redes de telefonía celular:** Se basan en la utilización de torres de radio distribuidas geográficamente que permiten a los dispositivos móviles cambiar de frecuencia automáticamente para mantener la comunicación sin interrupciones. Este sistema facilita la movilidad del usuario y optimiza el uso de las frecuencias de radio disponibles, permitiendo así la conexión de un gran número de usuarios simultáneamente. El impacto de estas redes es notable en la expansión de la comunicación móvil a nivel global, permitiendo la interconexión de personas en áreas extensas y remotas.

- **Redes 4G:** El estándar 4G representa una evolución significativa respecto a las generaciones anteriores de redes móviles. Utiliza tecnología de conmutación de paquetes para transmitir datos de manera eficiente y a mayores velocidades. Con una mayor velocidad las redes 4G actualmente dominan el mercado ya que han mejorado considerablemente la experiencia del usuario en términos de navegación web, transmisión de videos y otras aplicaciones que requieren un mayor ancho de banda en comparación con sus predecesoras. Este avance ha sido crucial para el desarrollo de aplicaciones más sofisticadas y ha preparado el terreno para la llegada del 5G, que promete velocidades aún mayores y una conectividad más robusta.

- **Wifi:** La tecnología wifi permite la conexión a Internet a través de ondas de radio y enrutadores localizados. A diferencia de las redes móviles, el wifi no ofrece una transmisión continua sin establecer una conexión previa. Sin embargo, su capacidad para conectar múltiples dispositivos dentro de un área limitada lo hace ideal para el hogar, oficinas y espacios públicos. El wifi ha democratizado el acceso a Internet, proporcionando una alternativa económica y accesible a las redes móviles, especialmente en entornos fijos.

- **Bluetooth:** El Bluetooth es una tecnología de comunicación inalámbrica a corta distancia que utiliza ondas de radio de baja potencia. Esta tecnología permite la conexión rápida y sencilla de dispositivos como auriculares, altavoces y teléfonos móviles. Su principal ventaja radica en la simplicidad y eficiencia en la creación de redes personales de área reducida, facilitando la sincronización de dispositivos y la transferencia de datos de manera efectiva.

En resumen, la diversidad de tecnologías de redes móviles actuales refleja la continua evolución y adaptación a las necesidades cambiantes de los usuarios. Cada tipo de red ofrece ventajas específicas que complementan las demandas de conectividad en diferentes contextos, desde la movilidad y alta velocidad de las redes celulares y 4G, hasta la accesibilidad y facilidad de uso del wifi y Bluetooth. Estas tecnologías no solo han transformado la comunicación personal y profesional, sino que también han sentado las bases para innovaciones futuras en el ámbito de la conectividad inalámbrica.

La evolución en la capacidad de transmisión de datos y velocidad a la par de la evolución de los dispositivos móviles con altas prestaciones nos brinda un abanico de oportunidades en el desarrollo de aplicaciones dirigidas a usuarios de dispositivos móviles.

2.2 Desarrollo de aplicaciones nativas

El enfoque en el desarrollo de aplicaciones nativas se basa en la capacidad de aprovechar al máximo las funcionalidades y el entorno de cada sistema operativo. Esto se traduce en aplicaciones que pueden ofrecer un rendimiento superior y una mejor integración con los servicios nativos del sistema operativo en cuestión. Además, al estar diseñadas específicamente para una plataforma, las aplicaciones nativas suelen proporcionar una

experiencia de usuario más cohesiva y optimizada, adaptada a las expectativas y estándares de cada ecosistema. Sin embargo, el desarrollo nativo puede requerir un mayor esfuerzo y recursos inicialmente debido a la necesidad de gestionar múltiples códigos base para diferentes plataformas [18]. Esto hace que la toma de decisión para los desarrolladores y empresas sea compleja, por lo que se requiere un profundo análisis del proyecto a llevar a cabo y su alcance.

Para una mayor comprensión de como se realiza el desarrollo nativo y las modificaciones en esta sección trataremos el desarrollo Android, iOS.

2.2.1. Aplicaciones Android

Android es un sistema operativo diseñado para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas, aunque también se encuentra en otros dispositivos como relojes inteligentes, televisores e incluso en los sistemas multimedia de algunos modelos de coches. Este sistema operativo es desarrollado por Google LLC en colaboración con la Open Handset Alliance[1], y está basado en el kernel de Linux y otros softwares de código abierto. Android se ha convertido en el principal responsable de la popularización de muchos dispositivos inteligentes, ya que facilita el uso de una amplia variedad de aplicaciones de forma sencilla. Cabe destacar que el núcleo de Android es de código abierto, lo que permite que otros fabricantes adapten el sistema a sus propios dispositivos, aunque muchos dispositivos incluyen la versión oficial de Google, que cuenta con Google Play, Chrome y otras aplicaciones de serie [19].

Inicialmente desarrollado por Android Inc., la compañía fue adquirida por Google en 2005 y presentó su producto en 2007, marcando un hito en la evolución de los estándares abiertos en dispositivos móviles. El código fuente principal de Android, ha destacado como el sistema operativo móvil más utilizado a nivel mundial. Según Statista, en 2020, Android cuenta con una cuota de mercado del 84.1 % [32], superando significativamente a su competidor más cercano, iOS [19].

Requisitos de desarrollo

Para poder desarrollar aplicaciones Android, necesitarás cumplir con los siguientes requisitos [20]:

- Un ordenador con sistema operativo Windows, macOS o Linux.
- El kit de desarrollo de Java (del inglés *Java Development Kit (JDK)*).
- El Android Software Development Kit (SDK).
- Un entorno de desarrollo integrado IDE (del inglés *Integrated Development Enviroment*) como Android Studio[2].

Puedes programar apps Android en cualquier lenguaje que pueda compilarse y correrse en la máquina virtual de Java (del inglés *Java Virtual Machine (JVM)*), y tus usuarios finales nunca lo notarían. Y un lenguaje, compatible con la JVM, que ha llamado mucho la atención de la comunidad Android es Kotlin[3], un lenguaje de programación estáticamente tipado, hecho por JetBrains[4]. Kotlin es el lenguaje recomendado en la actualidad

[1] https://www.openhandsetalliance.com/
[2] https://developer.android.com/studio
[3] https://kotlinlang.org/
[4] https://www.jetbrains.com/

para desarrollar aplicaciones Java, ya que presenta menor curva de aprendizaje, combinación de lo mejor de la programación funcional y procedural y compatibilidad total con Android Studio (El entorno de desarrollo integrado de JetBrain para aplicaciones Android)[21].

2.2.2. Aplicaciones iOS

Las aplicaciones iOS son programas diseñados exclusivamente para dispositivos de Apple que operan con el sistema operativo iOS. Inicialmente conocido como iPhone OS, este sistema fue lanzado por Apple en conjunto con el primer iPhone en 2007. Con la introducción del iPad en 2010, que también adoptó este sistema operativo, Apple decidió renombrarlo como iOS para reflejar su expansión a múltiples dispositivos. El prefijo "i", utilizado en iOS y otros productos de Apple desde la introducción del iMac en 1998, simboliza conceptos como individuo, instruir, informar e inspirar, según explicó Steve Jobs [22].

iOS se distingue como un sistema operativo cerrado y exclusivo para dispositivos Apple, en contraste con Android, que es utilizado por diversos fabricantes de dispositivos móviles. Esta exclusividad permite una integración profunda entre hardware y *software*, asegurando una experiencia de usuario optimizada y cohesiva en todos los dispositivos de la marca Cupertino [22].

A continuación exploraremos el desarrollo y las ventajas de las aplicaciones iOS, enfocándonos en el entorno de desarollo y sus diferentes enfoques.

Requisitos de desarrollo

Para poder desarrollar aplicaciones iOS, necesitarás cumplir con los siguientes requisitos técnicos [23]:

- Un ordenador con sistema operativo macOS.
- Credenciales de desarrollador de Apple.
- Xcode como entorno de desarrollo integrado (Integrated Development Environment (IDE)) oficial de Apple.

Los desarrolladores iOS deben conocer los lenguajes de programación Swift[5] y Objective-C[6], así como Application Programming Interface (API) (del inglés *Application Programming Interface*) relacionadas con el sistema operativo iOS [23].

2.3 Enfoques de aplicaciones múltiplataforma

Debido a los costos y requisitos de conocimiento asociados con el enfoque de desarrollo nativo [9], existen numerosos enfoques alternativos disponibles [7]. Estos son comúnmente conocidos como desarrollo móvil multiplataforma, un término general que abarca una amplia variedad de enfoques conceptuales de desarrollo y marcos técnicos [2] para simplificar la creación de aplicaciones [12]. Típicamente, un único código base puede ser utilizado para especificar o generar aplicaciones desplegables en varias plataformas con pocas o ninguna modificación específica de plataforma, aunque la capacidad de compartir código entre plataformas difiere entre los marcos y enfoques [11].

[5] https://www.swift.org/
[6] https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC

Los primeros ejemplos de marcos de desarrollo multiplataforma incluían Adobe PhoneGap (ahora conocido como Apache Cordova[7]), Appcelerator Titanium[8] y Xamarin[9], mientras que ofertas modernas similares incluyen Ionic[10] (basado en el tiempo de ejecución Capacitor[11]), React Native[12] y Flutter[13]. Estos marcos abstraen las diferencias entre plataformas empaquetando, compilando o envolviendo el código de la aplicación independiente de la plataforma de una manera compatible con cada plataforma de destino [10].

Para una mayor comprensión de los marcos de desarrollo se introducen los contextos "basado en plataforma", "web", "híbrido", "Progressive Web App (PWA)" (del inglés *Progressive Web App*) y finalmente "Multiplataforma". Los cuales se irán describiendo en la sección actual.

2.3.1. Desarrollo basado en plataforma

Este enfoque abarca el desarrollo de aplicaciones específicas para una plataforma particular. Por ejemplo, para iOS se utilizaría el lenguaje de programación Swift y el entorno de desarrollo Xcode, mientras que para Android se utilizaría Java o Kotlin con Android Studio, como habíamos comentado en la sección anterior de desarrollo nativo, o incluso para escritorio, como Windows, Linux o macOS. Cada plataforma tiene sus propias herramientas, lenguajes de programación y estándares que deben seguirse.

2.3.2. Desarrollo web

Se conoce como desarrollo web al proceso de crear y mantener un sitio web que sea funcional en internet, a través de diferentes lenguajes de programación, según el modelo y la parte de la página que corresponda. Cada sitio tiene una Uniform Resource Locator (URL) única que lo distingue de los demás en la red informática mundial [26].

En el desarrollo web, la compatibilidad entre plataformas es esencial para garantizar que las aplicaciones y sitios web funcionen de manera consistente en diferentes dispositivos y navegadores. Los lenguajes del navegador, como HTML, CSS y JavaScript, son fundamentales para construir experiencias web interactivas y atractivas. Sin embargo, la forma en que estos lenguajes son interpretados y ejecutados puede variar según el navegador y la plataforma en la que se ejecuten.

HTML (*HyperText Markup Language*): el lenguaje de marcado estándar utilizado para estructurar y presentar contenido en la web. Es soportado por todos los navegadores modernos y es fundamental para la creación de páginas web [13].

CSS (*Cascading Style Sheets*): utilizado para definir el diseño y la apariencia visual de una página web. Aunque la mayoría de los navegadores admiten las características básicas de CSS, puede haber diferencias en la interpretación de estilos más avanzados y propiedades específicas del navegador [13].

JavaScript: lenguaje de programación utilizado para agregar interactividad y funcionalidad dinámica a las páginas web. A pesar de que la mayoría de los navegadores son

[7] https://cordova.apache.org/
[8] https://titaniumsdk.com/
[9] https://dotnet.microsoft.com/en-us/apps/xamarin
[10] https://ionicframework.com/
[11] https://capacitorjs.com/
[12] https://reactnative.dev/
[13] https://flutter.dev/

compatibles con JavaScript, pueden existir diferencias en la implementación de ciertas características y API entre los navegadores [8].

Para garantizar la compatibilidad entre plataformas, es importante realizar pruebas exhaustivas en una variedad de dispositivos y navegadores. Además, el uso de bibliotecas y *frameworks* como jQuery[14], React[15] o Angular[16] puede facilitar el desarrollo de aplicaciones web compatibles con múltiples plataformas, ya que estos *frameworks* suelen abstraer las diferencias entre los navegadores y proporcionar soluciones predefinidas para problemas comunes de compatibilidad [27].

Las aplicaciones web ofrecen varias ventajas significativas sobre las aplicaciones nativas. Una de las más destacadas es su naturaleza no descargable, lo cual elimina la necesidad de ocupar espacio de almacenamiento en los dispositivos de los usuarios. Al operar directamente desde navegadores como Google Chrome[17] o Safari[18], las aplicaciones web también simplifican el proceso de desarrollo. Utilizando tecnologías ampliamente accesibles como Javascript, HTML y CSS, los desarrolladores pueden crear soluciones rápidamente y a un costo menor en comparación con el desarrollo de aplicaciones nativas. Esta accesibilidad no solo reduce los tiempos de desarrollo, sino también los costos asociados con mantenimiento y actualizaciones, haciendo de las aplicaciones web una opción económica y eficiente para muchos proyectos[17].

Sin embargo, las aplicaciones web también enfrentan desafíos, principalmente en términos de funcionalidad y accesibilidad. A diferencia de las aplicaciones nativas, las aplicaciones web pueden tener restricciones para acceder a ciertas características del dispositivo móvil, limitando su capacidad para aprovechar plenamente hardware específico o capacidades offline. Además, la dependencia de una conexión a internet constante puede ser una barrera para usuarios en áreas con conectividad limitada o inestable. Aunque las PWA intentan mitigar algunos de estos problemas permitiendo un mayor acceso a funcionalidades del dispositivo y funcionando offline de manera limitada, aún enfrentan desafíos para ofrecer una experiencia completa y fluida comparable a las aplicaciones nativas en todas las circunstancias[17].

En resumen, el desarrollo web moderno requiere un enfoque cuidadoso en la compatibilidad entre plataformas para garantizar una experiencia consistente para todos los usuarios, independientemente del dispositivo o navegador que utilicen. A pesar de sus ventajas hemos de tener en cuenta las limitaciones en el acceso a funcionalidades asociadas a los dispositivos y el funcionamiento offline, esto puede suponer un inconveniente para algunos proyectos.

2.3.3. Desarrollo híbrido

El desarrollo híbrido combina elementos de desarrollo web y aplicaciones nativas. Se trata de la User Interface (UI) dibujada con tecnologías web a través de una ventana que, en realidad, es un navegador incrustado en un contenedor nativo. Con este enfoque, se utiliza un único código base, escrito en lenguajes web como HTML, CSS y JavaScript. Esto se envuelve en un contenedor nativo para su ejecución en diferentes plataformas. Esto permite crear una aplicación que puede ejecutarse en varios sistemas operativos, sin tener que desarrollar una versión separada para cada plataforma [28].

[14] https://jquery.com/
[15] https://es.react.dev/
[16] https://angular.dev/
[17] https://www.google.es/chrome/index.html
[18] https://www.apple.com/es/safari/

El desarrollo híbrido se realiza en muchos casos con el objetivo de crear una versión móvil para ampliar el mercado de una web ya existente, por lo que sería una adaptación más rápida que el desarrollo nativo.

Las aplicaciones híbridas presentan varias ventajas significativas, siendo una de las más destacadas su rápido desarrollo. Al utilizar una base de código única que puede desplegarse en múltiples plataformas, los desarrolladores pueden reducir considerablemente el tiempo y los recursos necesarios para lanzar la aplicación en comparación con el desarrollo nativo individual para iOS y Android. Esto no solo acelera el tiempo de comercialización, sino que también facilita la gestión y actualización de la aplicación en el futuro[17].

También podemos encontrar el concepto "webview" en algunas implementaciones, el cual es un componente de interfaz de usuario que permite a las aplicaciones mostrar contenido web dentro de la propia aplicación, se utiliza comunmente en aplicaciones móviles y de escritorio como solucion sin necesidad de abrir un navegador web.

Sin embargo, estas ventajas vienen acompañadas de desventajas importantes. Una de las más notables es el menor rendimiento en comparación con las aplicaciones nativas. Debido a que las aplicaciones híbridas dependen de un componente tipo navegador para renderizar la interfaz de usuario y acceder a las funciones del dispositivo, pueden experimentar tiempos de carga más lentos y un rendimiento general inferior, especialmente en aplicaciones que requieren un alto rendimiento gráfico o un procesamiento intensivo de datos. Además, la necesidad de superar la diferencia entre los entornos nativos y web puede aumentar la complejidad del desarrollo y el costo, especialmente para aplicaciones que necesitan una integración profunda con hardware específico o características avanzadas de la plataforma lo que hace que la User Experience (UX)/UI quede lejos de la proporcionada por aplicaciones nativas[17].

2.3.4. Desarrollo de Progresive Web Apps

Ya teniendo claro lo que es una aplicación híbrida, nativa y web podemos pasar a lo siguiente. Segun el blog de Chrome para desarrolladores [29] se define de la forma siguiente:

> "Las apps web progresivas usan capacidades web modernas para ofrecer una experiencia del usuario similar a la de las apps. Evolucionan de páginas en pestañas del navegador a apps envolventes de nivel superior, lo que mantiene la fricción de la Web en todo momento."

Las *Progressive Web Apps* se distinguen por varias características clave: son universalmente adaptables, funcionan en cualquier navegador y dispositivo, pueden operar sin conexión gracias a los service workers, ofrecen una experiencia similar a las aplicaciones nativas con actualizaciones automáticas y seguridad reforzada mediante TLS. Además, son fácilmente identificables y compartibles a través de URL, eliminando la necesidad de instalación desde una tienda de aplicaciones [29].

Es una herramienta que está programada en lenguaje web como son HyperText Markup Language version 5 (HTML5), JavaScript y CSS, pero no se queda ahí pues tiene el comportamiento como una aplicación móvil nativa y a su vez como una aplicación web ("una aplicación combinada") siendo esta multiplataforma pues se pueden utilizar en diferentes dispositivos, son menos pesadas y se actualizan rápidamente, también se puede hacer uso de los mensajes Push [30].

Entre los *frameworks* existentes para esta variante de desarrollo se encuentran Angular[19], React[20], Polymer[21], Ionic[22], Vue[23], Magento PWA Studio[24] y Svelte[25]. Cada uno tiene ventajas y desventajas, la elección de alguno de ellos será en dependencia de las necesidades teniendo en cuenta la escala del proyecto, el equipo de desarrollo, compatibilidad con *software* existente y restricciones de coste y tiempo; donde pesará mucho la existencia de alguno de estos *frameworks* en su versión web [31].

2.3.5. Desarrollo usando *frameworks*

Para acercarnos al mundo del desarrollo multiplataforma utilizaremos el estudio reciente de Andrew Barnett[14] donde realiza experimentos que comparan aplicaciones móviles multiplataforma y nativas.

Este tipo de desarrollo puede brindar una variedad de resultados positivos, incluida la reducción en el tiempo, el esfuerzo y los costos generales requeridos para el desarrollo y el mantenimiento. Esto se debe a la capacidad de compartir una única base de código para todas sus plataformas móviles, ahorrando tiempo y esfuerzo porque el código solo necesita escribirse una vez. Naturalmente, estos tiempos de desarrollo resultan en costos de desarrollo más bajos. Además, permite a los equipos unificar sus procesos de desarrollo para diferentes plataformas móviles, lo que puede reducir los gastos administrativos y optimizar las operaciones .

Aunque los enfoques multiplataforma pueden ofrecer muchos beneficios, también conllevan su parte de desafíos potenciales. Algunos investigadores y profesionales expresan preocupaciones sobre varios aspectos de las aplicaciones multiplataforma, incluido su rendimiento, la percepción del usuario, el acceso a las características del dispositivo y la experiencia de desarrollo y mantenimiento.

React Native[26] y Flutter[27], representando los enfoques interpretados y compilados cruzados respectivamente, parecen haberse convertido en líderes claros en el espacio de los marcos de trabajo multiplataforma desde 2018 y no muestran signos de perder popularidad en esta etapa. Cordova[28] y Xamarin[29] han sido jugadores importantes en el espacio, pero ambos están experimentando un declive en popularidad, Cordova aún más que Xamarin. Sin embargo, estos marcos de trabajo han visto un uso considerable y sin duda siguen siendo parte de muchos sistemas heredados. Por otro lado, el marco de trabajo Ionic, junto con su nuevo tiempo de ejecución Capacitor, está mostrando signos de aceleración y parece haber desplazado a Cordova del primer lugar entre los enfoques híbridos.

Basándonos en este estudio vamos a detallar los *frameworks* líderes React Native y Flutter

[19] https://angular.dev/
[20] https://es.react.dev/
[21] https://polymer-library.polymer-project.org/3.0/docs/devguide/feature-overview
[22] https://ionicframework.com/
[23] https://vuejs.org/
[24] https://business.adobe.com/products/magento/progressive-web-apps.html
[25] https://svelte.dev/
[26] https://reactnative.dev/
[27] https://flutter.dev/
[28] https://cordova.apache.org/
[29] https://dotnet.microsoft.com/en-us/apps/xamarin

React Native

React Native es un *framework* open source desarrollado por Facebook[30] para crear aplicaciones móviles nativas para Android, iOS y más, utilizando JavaScript y React[31] usando un único código base. Este *framework* lleva las mejores partes del desarrollo con React al desarrollo nativo, siendo una biblioteca de JavaScript de primera clase para construir interfaces de usuario [33].

Entre sus ventajas destacan que utiliza una arquitectura orientada a componentes, lo que simplifica tanto el desarrollo como el mantenimiento de interfaces de usuario complejas, asegurando al mismo tiempo una apariencia uniforme en todos los dispositivos compatibles [34]. El rendimiento es óptimo al utilizar diferentes subprocesos para API y UI nativas, aunque se puede utilizar *webview*, pero esto reduciría el rendimiento[35]. La comunidad de desarrollo es extensa dado que los componentes nativos de React son open source y a diferencia de su contraparte Flutter el lenguaje JavaScript lo hace más amigable entre sus conocedores y si vienes de React en la web aún más [36].

El *framework* presenta desventajas en cuanto a la capacidad de personalización en los módulos, ya que se pueden dar casos donde la funcionalidad a implementar no esté disponible y sea necesario crear las 3 bases de código separadas (React Native, Android e iOS) para soportar dicha funcionalidad en lugar de una; por esta misma razón otra desventaja es que se hace necesario tener conocimiento de desarrollo nativo [36].

Para desarrollar apliaciones utilizando este *framework* es necesario tener conocimientos básicos de desarrollo móvil, experiencia en lenguajes de programación Javascript, Java, Kotlin, Objective-C y Swift ya que el mayor desarrollo transcurre en entorno Javascript pero se pueden dar casos de implementación nativa en entornos Android e iOS y la necesidad de utilizar las herramientas relacionadas como Android Studio y Xcode respectivamente. Al ser un *framework* basado en React.js es necesaria además cierta familiaridad con la biblioteca, sus métodos de gestión de estados entre otras funcionalidades que recaen en React [38].

Desde la versión 0.68 de React Native se ha modificado la arquitectura del *framework* para lograr mejor rendimiento, ya que su equipo ha detectado limitaciónes en la antigua arquitectura. Aún se está trabajando para hacer de esta la experiencia por defecto para React Native. Este cambio incluye novedades en la experiencia de usuario, en cuando a sincronicidad, renderizado y velocidad en momentos de la creación de módulos. Todos estos cambios llevan a la comunidad a lentamente transicionar las bibliotecas actuales a esta nueva arquitectura y el equipo de React Native ha habilitado un equipo de trabajo[32] como espacio dedicado a brindar soporte y coordinación para la adopción de la nueva arquitectura [37].

Flutter

Flutter de acuerdo a su página oficial[33] es un *framework* open source creado por Google para construir aplicaciones multiplataforma compiladas nativamente desde una única fuente de código. Presenta una integración perfecta con servicios de Google como Firebase, Google Ads, Google Wallet entre otros, lo que le aporta un valor añadido que puede potenciar su uso en algunos proyectos. El lenguaje de programación por el que está impulsado es Dart, este lenguaje ha sido optimizado para lograr un buen rendimiento en cualquier plataforma.

[30] https://www.facebook.com/
[31] https://es.react.dev/
[32] https://github.com/reactwg/react-native-new-architecture
[33] https://flutter.dev

Este *framework* ofrece múltiples ventajas que lo destacan en el mercado de desarrollo de aplicaciones móviles. En primer lugar, actualizaciones instantáneas permite la solución de errores en tiempo real, mejorando la productividad de los desarrolladores. Además, su motor gráfico asegura un rendimiento nativo, logrando aplicaciones indistinguibles de las nativas sin intérpretes intermedios. Su capacidad multiplataforma acelera la velocidad de comercialización, garantizando lanzamientos simultáneos en iOS y Android. Su lenguaje de programación Dart es fácil de aprender, facilita la entrada de nuevos desarrolladores al desarrollo móvil [42].

A pesar de esto, presenta algunas desventajas. Al ser un *framework* relativamente nuevo, aún se están explorando sus limitaciones y problemas de compatibilidad en diferentes plataformas. Uno de los principales inconvenientes es el tamaño de las aplicaciones, lo que puede hacer que los desarrolladores busquen alternativas más ligeras. Además, aunque Flutter funciona en iOS y Android, su desarrollo por Google puede dar una ligera ventaja a Android, y las actualizaciones en iOS pueden no ser tan rápidas. Finalmente, el lenguaje de programación Dart, aunque rápido y fácil de aprender, aún no es tan popular ni tan ampliamente adoptado como otros lenguajes como Java, C#, o JavaScript, lo que podría ser una barrera para algunos desarrolladores [42].

Similar a React Native, Flutter utiliza *platform channels* para integrar bibliotecas y funcionalidades nativas de Android e iOS con Flutter (Dart)[34]. Esto permite aumentar las funcionalidades de Flutter acorde a las necesidades del desarrollador, aunque los proveedores de servicios cada vez tienen más en cuenta los desarrollos multiplataformas, se pueden encontrar muchas veces la existencia solamente de SDKs nativos.

2.3.6. Conclusión

El desarrollo móvil contemporáneo ofrece dos enfoques principales: las aplicaciones nativas y el desarrollo multiplataforma.

Las aplicaciones nativas son específicas de cada plataforma (Android e iOS), optimizadas para aprovechar al máximo las funcionalidades y el rendimiento de cada sistema operativo. Este enfoque garantiza una integración profunda con el ecosistema de cada plataforma, ofreciendo control total sobre la experiencia del usuario pero requiriendo la gestión de múltiples códigos base y recursos específicos para cada plataforma.

Por otro lado, el desarrollo multiplataforma permite crear aplicaciones que funcionen en varias plataformas con un único código base. Este enfoque ha evolucionado para abordar los desafíos de costos y la complejidad asociada con el desarrollo nativo.

Cada enfoque tiene sus propias ventajas y desafíos. La elección entre estos enfoques dependerá del tipo de aplicación, los recursos disponibles y las preferencias del equipo de desarrollo. Es fundamental evaluar cuidadosamente cada opción para determinar cuál se alinea mejor con los objetivos y requisitos del proyecto específico.

[34] https://docs.flutter.dev/platform-integration/platform-channels

CAPÍTULO 3
Análisis del caso práctico

En el contexto del desarrollo de aplicaciones móviles, el mantenimiento suele plantear desafíos significativos, razón por la cual las soluciones multiplataforma ofrecen una mejora sustancial en este aspecto. La decisión de pasar a un desarrollo multiplataforma puede darse por distintos motivos, basados fundamentalemnte en tres aspectos: económico, técnico o estratégico.

Desde el punto de vista contable se evalúa el valor de los sistemas existentes, los costes que se asocian a los mismos, si ya se encuentran amortizados y si presenta valor residual. Los costes de desarrollo incluyen todos los gastos del proceso de creación del *software*, salarios de desarrolladores, licencias de *software*, equipos y otros recursos. Según la Norma Internacional de Contabilidad No.38[40] estos gastos pueden catalogarse como activos intangibles. La amortización de este coste se espera a lo largo de su vida útil la cuál se estima generalmente entre 3 y 6 años [41]. Los costes de mantenimiento se asocian a la actualización y mejora para mantener su funcionalidad y eficiencia; aunque estos costes no se capitalizan consisten gastos en el período que se incurren.

Finalmente el valor residual según la Norma Internacional de Contabilidad No. 38, es el importe que una entidad espera obtener por un activo al final de su vida útil. Los factores que pueden afectar este valor son la obsolescencia tecnológica, los costos de disposición y, en caso de que el *software* esté diseñado específicamente para uso interno de la empresa, es probable que su valor residual sea nulo.

Hay que tener en cuenta que al hacer un cambio de desarrollo nativo a multiplataforma se está amortizando completamente el *software* antiguo y su valor residual pasa a ser un coste irrecuperable (o costo hundido).

En la parte técnica es muy común que se decida pasar a desarrollo multiplataforma cuando el sistema puede llegar a quedar obsoleto y abordar una actualización tendría un coste elevado, incrementando el valor residual prácticamente al 100 %. En este caso la decisión sería entre un sistema completamente nuevo y un sistema antiguo con valor residual de 100 %, por lo que se opta por invertir en un sistema nuevo.

Sin embargo existen casos en los que se decide el cambio por motivos estratégicos de la empresa, asumiendo el coste, estos motivos pueden ser variados y estar o no relacionado con aspectos técnicos o contables del sistema existente.

En este capítulo abordaremos un caso práctico real y el estudio realizado para fundamentar la decisión de cambiar de desarrollo nativo a desarrollo multiplataforma.

3.1 Contexto del caso práctico

Para este estudio de caso contamos con la existencia de aplicaciones de desarrollos nativos en Android e iOS; además de la existencia de un desarrollo web, el cual no se pretende modificar. Estas aplicaciones cuentan con incorporación de *webviews* en un 80 %, en el otro 20 % se encuentra la navegación y bibliotecas nativas de funcionalidades específicas, relacionadas con la estructura de la aplicación.

La problemática central del estudio reside en las dependencias de bibliotecas de terceros necesarias para las funcionalidades de la aplicación y el desarrollo de las funcionalidades que actualmente se encuentran en webviews. El proyecto está relacionado con la industria televisiva e informativa, cuya base económica se sustenta en varios pilares fundamentales.

En el caso de la publicidad los anunciantes pagan grandes sumas por espacios publicitarios durante los programas, y de esta forma aprovechan la capacidad de la televisión para llegar a un público diverso [39].

Además, los modelos de pago por ver han ganado relevancia con la expansión del *streaming*. Los consumidores pagan tarifas para acceder a contenido específico a través de plataformas digitales, lo cual ha democratizado el acceso al entretenimiento y permitido la personalización de las experiencias televisivas[39]. Aunque en el caso específico del cliente actual es una emisora pública, por lo que este modelo no se aplicaría.

Esta relación simbiótica entre contenido y publicidad es el impulsor de la economía televisiva. Para contabilizar audiencias y publicidades en las aplicaciones es necesaria la integración de servicios de terceros que se encargan de estos marcajes; específicamente en este estudio de caso se presentan dependencias con al menos 6 servicios de terceros para distinta finalidad algunos de ellos, relevantes para el análisis, se abordarán en este capítulo.

Para facilitar la compresión de ahora en adelante vamos a definir las entidades del contexto: le llamaremos “Cliente” a la empresa dueña de las aplicaciones, la cadena de televisión; llamaremos “Sistema” a las aplicaciones (Android e iOS) o en otro contexto al sistema multiplataforma a desarrollar; llamaremos “Servicios Externos” a los servicios encargados de los marcajes y audiencias; y continuaremos llamando Alten a la empresa encargada del desarrollo, compuesta por equipo administrativo y técnico encargado de llevar a cabo el proyecto.

En el Diagrama de Contexto 3.1 se muestran las entidades que se involucran con el sistema y a groso modo sus interacciones.

El Cliente es quien se encarga de proveer la información y el sistema la muestra para que los Usuarios accedan a esta información de manera estructurada cumpliendo con lo marcado sobre UX/UI por el Cliente. El sistema le envía los datos de uso e interación de los usuarios con la aplicación a los Servicios Externos. Finalmente “Alten”, la empresa, es la encargada de desarrollar y mantener el sistema.

3.2 Entorno existente

El entorno del proyecto existente, se compone de tres desarrollos, dos nativos (Android e iOS) y uno Web.

Aquí se puede observar cómo existe un equipo para cada desarrollo 3.2, todos los desarrollos consumen los mismos servicios de terceros y las API proporcionada por el

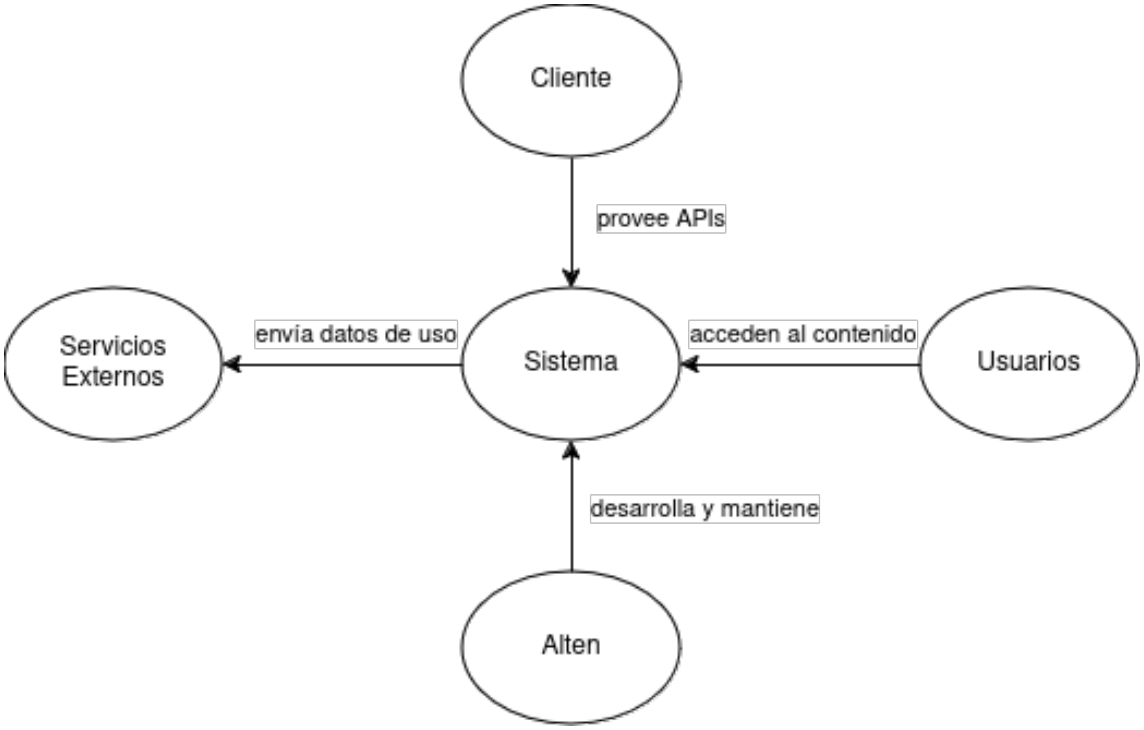


Figura 3.1: Diagrama de Contexto.

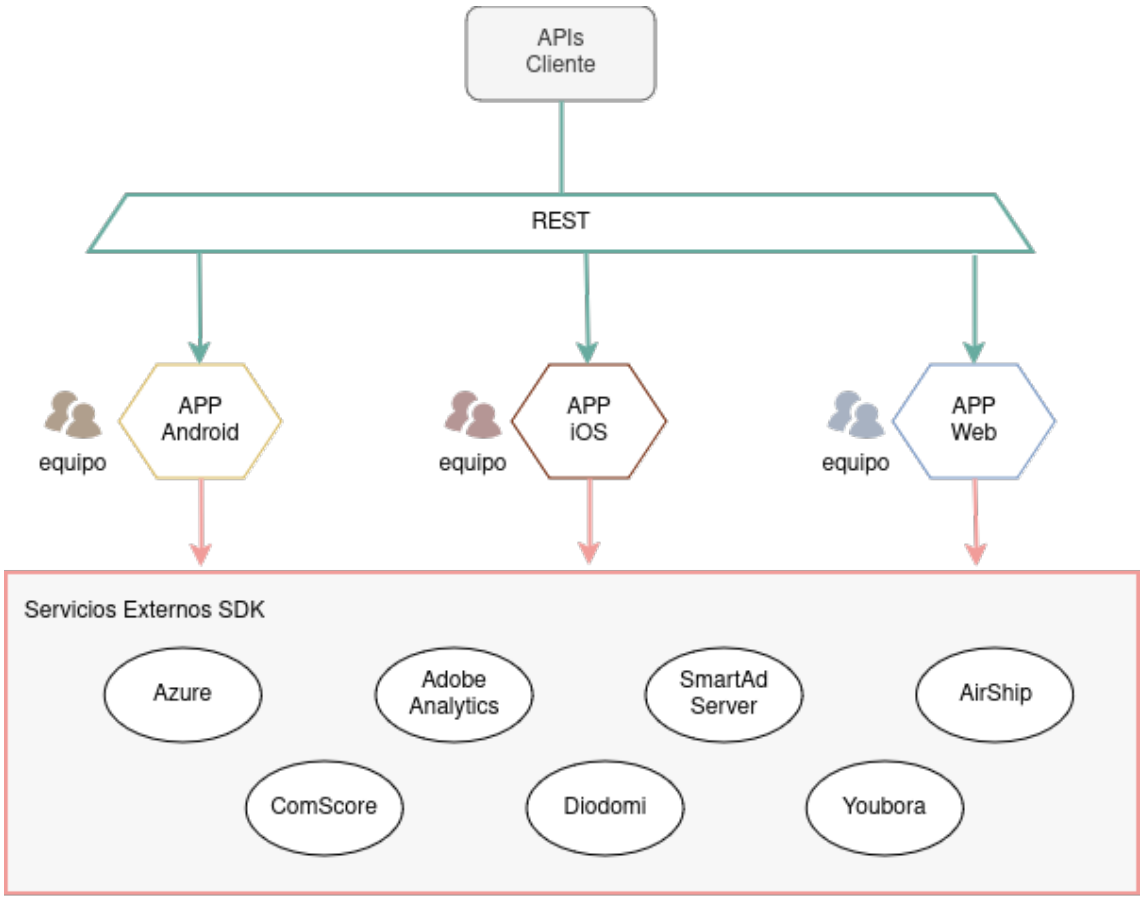


Figura 3.2: Diagrama de Entorno existente.

Cliente. La implementación del entorno de almacenamiento de datos no es relevante para nuestro análisis ya que estamos enfocados en el desarrollo con tecnologías *frontend*.

Las tecnologías y lenguajes utilizados en cada caso son:

- **Android**: lenguaje Kotlin.
- **iOS**: lenguaje Swift.
- **Web**: lenguaje Javascript, *framework* ReactJS.

Los desarrollos están unificados bajo una gestión centralizada con un único jefe de proyecto. Esta estructura busca mantener una visión cohesiva del avance y asegurar que todos los componentes trabajen hacia un objetivo común. Sin embargo, gestionar múltiples equipos de desarrollo puede presentar desafíos significativos, tales como la coordinación inter-equipos y la alineación de objetivos.

Para abordar estos retos y garantizar una gestión eficiente, se utiliza una metodología ágil basada en Scrum[1] levemente modificada a las necesidades del proyecto. Scrum, es una metodología ágil utilizada en el desarrollo de proyectos, particularmente en la industria del *software*. Se centra en la entrega incremental y repetitiva de productos, promoviendo la colaboración, la adaptabilidad y la mejora continua.

En la implementación de Scrum dentro del proyecto, se destacan varios roles clave que facilitan el proceso y aseguran el éxito del desarrollo. El Equipo de Desarrollo es un grupo autoorganizado de profesionales que trabajan juntos para entregar incrementos de producto potencialmente utilizables al final de cada *Sprint*. Dentro de este marco, se utilizan varios artefactos importantes:

- El ***Product Backlog*** es una lista priorizada de tareas, características y funcionalidades necesarias para el producto.
- El ***Sprint Backlog*** es un subconjunto de este backlog, que el equipo se compromete a completar durante el próximo *Sprint*.
- El **Incremento** es el resultado final de un *Sprint*, que debe ser un producto potencialmente entregable.

El *Product Owner* es responsable de maximizar el valor del producto y gestionar el *Product Backlog*, priorizando las tareas según su valor para el cliente. El Scrum *Master* facilita el proceso de Scrum, ayudando a eliminar impedimentos y asegurando que el equipo siga las prácticas adecuadas. Además, Scrum se estructura en torno a varios eventos. El *Sprint* es un ciclo de trabajo fijo de 1 a 4 semanas donde se realiza el desarrollo de un incremento del producto, en este proyecto se ha fijado en 2 semanas. Durante este tiempo, se llevan a cabo *Daily Scrums*, reuniones diarias cortas para coordinar las actividades del equipo y ajustar el plan de trabajo diario, en este caso de entre 10 y 20 minutos. Al final de cada *Sprint*, se realiza un *Sprint Demo* para presentar lo logrado y recibir retroalimentación. Posteriormente, se lleva a cabo una *Sprint Retrospective* para reflexionar sobre el *Sprint* pasado y mejorar continuamente el proceso. Adicionalmente a esto, en acuerdo con el Cliente se realiza un *Sprint Refinement* en mitad del *Sprint* para aclarar dudas o brindar información necesaria [43].

Scrum se sustenta en tres pilares fundamentales: Transparencia, Inspección y Adaptación. La transparencia asegura que todos los aspectos del proceso sean visibles y comprendidos por todos los involucrados. La inspección implica revisar frecuentemente los artefactos y el progreso hacia el objetivo para identificar posibles problemas. Finalmente, la adaptación se refiere al ajuste de las estrategias y planes según las observaciones y aprendizajes adquiridos durante el proceso [43].

[1] https://www.scrum.org

Se utilizan los siguientes softwares como herramientas:

- Jira[2]: Gestión de proyecto
- Microsoft Office[3]: que es un paquete de softwares de comunicación, gestión de documentos y compartidos. Las más usadas son Teams[4], Outlook[5] y One Drive[6].
- Bitbucket[7]: Controlador de versiones, gestión de código, integración continua y despliegue continuo. Posee una fuerte integración con Jira.

3.3 Problema

En este estudio de caso se busca una reducción de costes y tiempos de desarrollo y despliegue de nuevas versiones. Es por ello que una de las ventajas del desarrollo multiplataforma es la reducción de equipos, esto ayuda a optimizar las sinergias del equipo, simplifica la gestión y tiene otros beneficios indirectos como un menor riesgo de absentismo menos vacaciones a cubrir durante el desarrollo.

Además, en el caso del Cliente, sus apps nativas están hechas con *webviews*, lo que no les da ninguna ventaja por el hecho de ser nativas, y les obstaculiza personalizar el contenido. Estas aplicaciones tienen una vigencia de más de 4 años por lo que su dependencia técnica es elevada.

Aunque el uso de estas aplicaciones nativas les abrió un público y ha servido como prueba, el negocio necesita de un mejor rendimiento en este entorno. En el contexto actual se hace muy complejo mantener los desarrollos en paralelo con nuevos cambios, Cliente quiere incorporar un cambio de diseño lo que hace factible el cambio a un desarrollo multiplataforma, brindaría un sistema sin deuda tecnológica, logrando mejorar la gestión del equipo, los tiempos de despliegue y disminuyendo los costes de mantenimiento futuros.

3.4 Procedimientos

Dependiendo del proyecto a llevar a cabo, se podría incluir todos los desarrollos en un desarrollo unificado multiplataforma, incluyendo el desarrollo Web. En el estudio de caso actual se espera migrar las aplicaciones móviles completas, dejando el desarrollo web independiente. Los sistemas existentes y su implementación nativa pueden servir de ayuda para la transformación de datos y lógica, incluso en algunos casos se podría dar provecho a la implementación web.

Conociendo la organización existente, se puede contar con la misma para incluir en el desarrollo multiplataforma, con un nuevo equipo de desarrollo en paralelo a los equipos existentes ya que se continua dando mantenimiento a los productos actuales. El nuevo desarrollo comprenderá estructura, diseño de interfaz y garantizará la continuidad de la mayor cantidad de funcionalidades que contienen los desarrollos actuales. Por la complejidad de las dependencias de terceros es necesario analizar y estudiar la forma en la que se migrará cada una de estas funcionalidades teniendo en cuenta que, en este análisis, se puede llegar a desestimar funcionalidades por falta de soporte.

[2]https://www.atlassian.com/es/software/jira
[3]https://www.microsoft.com/es-es/microsoft-365
[4]https://www.microsoft.com/es-es/microsoft-teams
[5]https://www.microsoft.com/es-es/microsoft-365/outlook
[6]https://www.microsoft.com/es-es/microsoft-365/onedrive
[7]https://bitbucket.org

Se requerirá de recursos humanos y materiales para la implementación y gestión del proyecto, un plan de organización y seguimiento del mismo. La planificación se proponen tres fases, una primera fase dónde se muestre la posibilidad de implementación de las funcionalidades principales relacionadas a servicios de tercero, sin mucho detalle en UX UI ya que este aspecto será contratado a otro equipo especialista en diseño de interfaces digitales. La segunda fase, enfocada en lograr un producto, donde se cumpla con estas principales funcionalidades y se catalogue como versión 1, comenzando a integar ya el diseño. Siendo la tercera fase progresiva con puntos intermedios de despliegue hasta conseguir el producto completo.

3.5 Conclusión del capítulo

El análisis del caso práctico ha demostrado la necesidad de migrar las aplicaciones móviles del cliente a un entorno multiplataforma para mejorar la eficiencia operativa y reducir costos. Tras las problemáticas identificadas esta decisión responde tanto a las demandas técnicas actuales como a los objetivos estratégicos de la empresa; además, consolida esfuerzos, reduce costes y simplifica la gestión del desarrollo. En el próximo capítulo, se evaluarán y seleccionarán las estrategias y *frameworks* de desarrollo más adecuados para llevar a cabo esta migración, para asegurar que la transición sea fluida y efectiva.

CAPÍTULO 4
Evaluación y selección de estrategias

Para la evaluación y selección de la estrategia a seguir en durante la migración se analizarán primeramente los *frameworks* de desarrollo multiplataforma Flutter y React Native para dar contexto del panorama actual en cuanto a eficiencia y opinión de la comunidad. Además se estructurará la metodología de gestión de proyecto, los recursos humanos y materiales necesarios y las herramientas a utilizar para el desarrollo.

4.1 Evaluación de *frameworks*

En conjunto con la dirección de la empresa y el Cliente se estimó conveniente el análisis de tres aspectos fundamentales para el producto. Donde se incluye valoración cualitativa en función de la opinión de desarrolladores, estudios comparativos existentes de rendimiento y disponibilidad de las bibliotecas necesarias para la integración con el framework, se considera la implementación de módulos multiplataforma en casos en los que no exista biblioteca compatible con el framework.

4.1.1. Comparativa cualitativa

Desde el punto de vista cualitativo el estudio [15] basado en entrevistas semiestructuradas a 20 desarrolladores con experiencia práctica en Flutter y React Native brinda una mirada general a la opinión de usuarios de estas tecnologías.

En las respuestas destacan tanto los desafíos técnicos como las respuestas emocionales, algunas de las opiniones comunes de los desarrolladores hacia ambos *frameworks* son:

- El rendimiento en animaciones, aunque se han hecho avances en *frameworks* multiplataforma, no son tan suaves como en aplicaciones nativas, lo que afecta la experiencia de usuario.
- La optimización y migración: optimizar el rendimiento y gestionar la memoria es un desafío para lo que se han implementado soluciones innovadoras como subdivisión de paquetes y la carga de módulos bajo demanda. Migrar la lógica del negocio existente a nuevas arquitecturas es un reto considerable.
- Soporte y comunidad: tanto Flutter como React Native cuentan con una comunidad extensa y activa, facilitando la resolución de problemas y la adopción del *framework*.

Siendo la comunidad de React Native más extensa y con soporte adicional de la comunidad de Javascript y React.

En el estudio se presentan varias comparaciones cualitativas entre Flutter y React Native, destacando en diversos aspectos según las características y necesidades del proyecto. A continuación, se detallan los aspectos en los que cada *framework* destaca:

Ventajas de Flutter

- **Rendimiento:** Flutter compila directamente a código nativo, eliminando la necesidad de un puente de JavaScript, lo que mejora significativamente el rendimiento y la capacidad de respuesta de las aplicaciones, especialmente en aplicaciones con interfaces de usuario complejas y animaciones intensas.
- **Consistencia UI/UX:** Flutter proporciona un amplio conjunto de widgets personalizables que facilitan la creación de interfaces de usuario consistentes y altamente personalizables en todas las plataformas.
- **Optimización de Rendimiento:** La arquitectura de Flutter y su motor de renderizado personalizado permiten una renderización rápida de la UI, lo que es ventajoso para manejar animaciones complejas e interfaces detalladas.
- **Sistema de Tipado:** El lenguaje de programación Dart de Flutter tiene un sistema de tipado fuerte, lo que mejora la calidad y el mantenimiento del código en comparación con la naturaleza dinámica de JavaScript.

Ventajas de React Native

- **Ecosistema de JavaScript:** React Native se beneficia del amplio y maduro ecosistema de JavaScript, que incluye una vasta colección de bibliotecas y macros que pueden acelerar los procesos de desarrollo.
- **Accesibilidad y Familiaridad:** Utiliza JavaScript, lo que permite a los desarrolladores web con experiencia en React trasladarse más fácilmente al desarrollo móvil. Esto reduce el tiempo de aprendizaje y facilita la adopción del framework.
- **Soporte Comunitario:** React Native cuenta con una comunidad robusta y extensa, proporcionando documentación exhaustiva y recursos comunitarios que facilitan la resolución de problemas y la adopción del framework.
- **Modularidad y Reutilización de Código:** La arquitectura modular de React Native promueve la reutilización de código entre las plataformas iOS y Android, lo que simplifica la colaboración en equipos de desarrollo y la gestión del código.

Desafíos y Oportunidades

- **Curva de Aprendizaje:** Ambos *frameworks* presentan una curva de aprendizaje significativa, especialmente para equipos acostumbrados al desarrollo nativo tradicional. Sin embargo, la familiaridad con JavaScript puede facilitar la adopción de React Native para desarrolladores web.
- **Optimización y Rendimiento:** Aunque ambos *frameworks* buscan ofrecer un rendimiento cercano al nativo, cada uno enfrenta desafíos únicos. React Native puede encontrar cuellos de botella en el puente de JavaScript, mientras que Flutter puede requerir esfuerzos adicionales para optimizar su motor de renderizado personalizado.

4.1.2. Comparativa de rendimiento

Basado en el análisis de rendimiento comparativo de Flutter y React Native, se pueden observar varias diferencias significativas en términos de uso de CPU, memoria y frames entrecortados.

En pruebas que implican el uso intensivo de la CPU, como la función de búsqueda de la aplicación, Flutter mostró un uso significativamente menor de CPU en comparación con React Native. Específicamente, Flutter tuvo un promedio de uso de CPU del 11.34 %, mientras que React Native tuvo un promedio del 36.21 % .

En general, Flutter también mostró un uso de memoria ligeramente mejor que React Native. Por ejemplo, en el test de la función de búsqueda, Flutter tuvo un uso promedio de memoria del 4.91 %, comparado con el 5.77 % de React Native .

En cuanto a la fluidez de la aplicación, medida por la cantidad de frames entrecortados, Flutter también superó a React Native en varias pruebas. En la prueba de búsqueda, Flutter tuvo un promedio de 2 frames entrecortados, mientras que React Native tuvo un promedio de 6.6 frames entrecortados .

En resumen, en términos de rendimiento, Flutter generalmente muestra una mejor eficiencia en el uso de CPU y memoria, además de una mayor fluidez en las interacciones de la aplicación en comparación con React Native.

Sin embargo con la nueva arquitectura de React Native estas cifras mejorarán [44]. Con este cambio React Native promete varias mejoras significativas en términos de rendimiento y por ende experiencia de usuario:

Mejor Rendimiento de la Aplicación

- **Reducción del Tiempo de Compilación:** Se ha reducido el tiempo necesario para compilar el código, lo que mejora el tiempo de inicio de las aplicaciones.
- **Menor Uso de Memoria:** Disminución del uso de JavaScript, lo que conduce a un menor consumo de memoria y mejor rendimiento general.
- **Ejecución Asíncrona de JavaScript:** Permite animaciones más rápidas y suaves, y mejor escalabilidad de las aplicaciones sin degradación del rendimiento.

Interfaz de Usuario más Responsiva

- **Tasa de Actualización Consistente:** Los componentes se actualizan de manera más regular y predecible, lo que mejora la experiencia del usuario.
- **Reducción de Tareas que Consumen Tiempo:** Código asíncrono eficiente que asegura una mejor capacidad de respuesta de la UI.

Experiencia del Desarrollador Mejorada

- **Desarrollo más Rápido y Eficiente:** Facilita la creación, depuración y despliegue de aplicaciones rápidamente, permitiendo a los desarrolladores enfocarse más en la calidad del código.
- **Compatibilidad con Versiones Anteriores:** Permite una transición suave sin la necesidad de reescribir proyectos existentes.

En conjunto, estas mejoras contribuyen a un rendimiento más eficiente y una mejor experiencia tanto para los desarrolladores como para los usuarios finales de las aplicaciones creadas con React Native. Aunque hoy en día hay muy pocas bibliotecas existentes que soportan la nueva arquitectura, se espera que migren progresivamente.

4.1.3. Disponibilidad de bibliotecas de terceros

En este aspecto analizaremos la disponibilidad de SDK en cada *framework* y posibilidad de utilizar los servicios con API REST o código nativo (Android / iOS). Para React Native, también su compatibilidad con la nueva arquitectura.

Tabla 4.1: Compatibility Table

	Flutter	React Native	REST API	Android/iOS
Bitmovin player	✓	✓		
Azure active directory	✓	✓	✓	✓
ComScore			✓	✓
Adobe Analytics	✓(AEP)	✓(AEP)	✓	✓
Didomi	✓	✓	✓	✓
SmartAd Server			✓	✓
Youbora		✓*	✓	✓
Airship	✓	✓	✓	✓

* : soporta la nueva arquitectura de React Native

Se puede comprobar en 4.1 que a diferencia del soporte nativo, el soporte multiplataforma es un poco precario en cuanto a los servicios de terceros. De la misma manera la nueva arquitectura es soportado solo por una biblioteca compatible con React Native. Por este motivo la implementación de React Native con la nueva arquitectura no es viable en el estudio de caso presentado.

Sin embargo al tratarse de servicios utilizados para realizar marcajes y analíticas se considera la opción de utilizar las bibliotecas nativas mediante implementación de módulos para el *framework* multiplataforma. Para ello se realizó una prueba de desarrollo de un módulo de ejemplo utilizando la nueva arquitectura siguiendo los pasos indicados en el grupo de trabajo "*React Native New Architecture*"[1] y la sección "Integración de Plataformas"[2] de la documentación de Flutter.

En el caso de React Native, se logró hacer módulos de la nueva arquitectura y se prosiguió con realizar el mismo módulo pero esta vez utilizando la guía que soporta ambas arquitecturas brindando compatibilidad retroactiva de los módulos, en cómputo de tiempo de 256 horas trabajadas. Como estas pruebas fueron exitosas se acepta la posibilidad de realizar módulos de React Native compatibles con ambas arquitecturas en los casos donde las bibliotecas soporten los desarrollos de Android e iOS.

En caso de Flutter se consiguió igualmente soportar el módulo, aunque esta ocasión en un cómputo de 344 horas trabajadas. El motivo de la tardanza en el desarrollo del mismo se debe principalmente a la dificultad y falta de experiencia en el lenguaje Dart, contrario a Javascript/Typescript en el cual ya se presenta experiencia; además que en el caso de React Native se realizaron dos pruebas de implementación, mientras que en Flutter solo se realizó una.

El estudio llevado a cabo brinda claridad sobre las alternativas y posibles costos de mantener la mayor cantidad de funcionalidades existentes, lo cual es primordial en el producto sin comprometer la capacidad del producto desarrollado de migrar en el futuro. Se comprende también que una vez superada la curva de aprendizaje de dicho procedimiento los desarrollos serían de mayor rapidez.

[1] https://github.com/reactwg/react-native-new-architecture/tree/main
[2] https://docs.flutter.dev/platform-integration/platform-channels

Sin embargo, los casos donde sin poseer solución multiplataforma, presentan soporte iOS, Android y además API REST, se estudiará la viabilidad de utilizar la API en lugar de realizar el desarrollo multiplataforma de dicho módulo.

4.2 Selección de *frameworks*

Para cuantificar el valor de cada variable y compararlas en aspectos: cualitativo, rendimiento y compatibilidad con bibliotecas de terceros.

Las ponderaciones definidas en el análisis se logran en conjunto con el Cliente, donde distribuye el 100 % del valor entre las variables. Cada aspecto con un total de 100 %, a mayor ponderación la variable es considerada de mayor valor para el proceso de desarrollo o el producto final.

4.2.1. Valor cualitativo

El valor cualitativo del *framework* se calculará en base a los indicadores Nulo, Adecuado, Bueno y Excelente, con valores normalizados en una escala de 0 a 1. Con lo que se obtienen valores del 0-3. Para facilitar el estudio se resume en cada *framework* las variables consideradas como relevantes extraído del análisis anterior y la consulta con el Cliente.

VC-01 Rendimiento en Animaciones:
Flutter: Mejora significativa al compilar directamente a código nativo.
React Native: Puede enfrentar cuellos de botella debido al puente de JavaScript.
Ponderación: 15 %

VC-02 Optimización y Migración:
Flutter: Arquitectura y motor de renderizado personalizados, potencialmente más eficiente.
React Native: Desafíos en optimización y migración debido a la naturaleza dinámica de JavaScript.
Ponderación: 25 %

VC-03 Consistencia UI/UX:
Flutter: Ofrece un amplio conjunto de widgets para una interfaz de usuario consistente.
React Native: Depende más de componentes nativos y bibliotecas de terceros para lograr consistencia.
Ponderación: 10 %

VC-04 Ecosistema y Modularidad:
Flutter: Ecosistema más joven pero en crecimiento; modularidad y reutilización de código más limitadas.
React Native: Amplio ecosistema de JavaScript, modularidad avanzada y reutilización de código entre plataformas.
Ponderación: 20 %

VC-05 Curva de Aprendizaje y Familiaridad:
Flutter: Requiere aprendizaje del lenguaje Dart; curva de aprendizaje moderada.

React Native: Familiaridad con JavaScript y React facilita la transición.
Ponderación: 20 %

VC-06 Soporte y Comunidad:
Flutter: Comunidad activa, pero más pequeña en comparación con React Native.
React Native: Comunidad extensa y madura, con una gran cantidad de recursos y soporte.
Ponderación: 10 %

Basado en este resumen podemos evaluar cada aspecto con las ponderaciones y calificativo para llegar a un total del valor cualitativo de los *frameworks* que se considera en este proyecto.

Tabla 4.2: Comparativa cualitativa entre Flutter y React Native.

Variable	Ponderación	Flutter	React Native
VC-01	15 %	Excelente	Adecuado
VC-02	25 %	Bueno	Adecuado
VC-03	10 %	Excelente	Adecuado
VC-04	20 %	Adecuado	Excelente
VC-05	20 %	Adecuado	Excelente
VC-06	10 %	Adecuado	Excelente

Variables en escala de **0 a 1**: Excelente: 1 Bueno: 0.67 Adecuado: 0.33 Nulo: 0

La **Puntuación Total** se calcula como:

$$\text{Puntuación Total} = \sum_{i=1}^{n} (\text{P}_i \times \text{C}_i)$$

Donde:

- P_i es la ponderación o peso de la variable i (en porcentaje).
- C_i es la calificación del *framework* en la variable i, ajustada a una escala de 0 a 1.

Sustituyendo lo valores de la tabla 4.2 a escala de 0 a 1 en cada caso:

$$\begin{aligned}\text{Puntuación Total de Flutter} = {} & (0{,}15 \times 1) + (0{,}25 \times 0{,}67) + (0{,}10 \times 1) \\ & + (0{,}20 \times 0{,}33) + (0{,}20 \times 0{,}33) + (0{,}10 \times 0{,}33)\end{aligned}$$

$$\begin{aligned}\text{Puntuación Total de React Native} = {} & (0{,}15 \times 0{,}33) + (0{,}25 \times 0{,}33) + (0{,}10 \times 0{,}33) \\ & + (0{,}20 \times 1) + (0{,}20 \times 1) + (0{,}10 \times 1)\end{aligned}$$

Obteniendo como resultado:

$$\text{Puntuación Total de Flutter} = 0{,}5925$$

$$\text{Puntuación Total de React Native} = 0{,}665$$

A pesar no ser mucha la diferencia, React Native destaca por encima de Flutter en el aspecto cualitativo, principalmente por su curva de aprendizaje, familiaridad y el amplio ecosistema existente.

4.2.2. Valor rendimiento

En el aspecto del rendimiento se toman los puntos de uso de Unidad Central de Procesamiento (CPU), uso de Memoria Random Access Memory (RAM) y el promedio de frames entrecortados. Las ponderaciones de relevancia para el proyecto fueron consultadas con el cliente para su validación, obteniendo una distribución 40-30-30 respectivamente.

En la variable **VR-03 Frames entrecortados** se considera un rendimiendo fluido para una cantidad menor o igual a 3, entre 4 y 6 provocaría irregularidad en las animaciones y mayor de 6 se pierde la fluidez afectando negativamente la experiencia de usuario. Por este motivo y para simplificar la comparación se pasa a base 10 los porcentajes de las variables **VR-01 Uso de CPU y VR-02 Uso de Memoria** unificando así la escala de valores. Para mejor compresión resumido en la tabla 4.3.

Tabla 4.3: Comparativa de rendimiento entre Flutter y React Native.

Variable	Ponderación	Flutter	React Native
VR-01	40 %	1.13	3.62
VR-02	30 %	0.49	0.58
VR-03	30 %	2	6.6

Para la variable de rendimiento a menor puntuación total se considera mejor rendimiento, ya que corresponde a un menor consumo de recursos. La **Puntuación Total** se calcula como:

$$\text{Puntuación Total} = \sum_{i=1}^{n} (\text{P}_i \times \text{C}_i)$$

Donde:

- P_i es la ponderación o peso de la variable i (en porcentaje).
- C_i es la calificación del *framework* en la variable i, ajustada a base 10.

Sustituyendo en cada caso:

$$\text{Puntuación Total de Flutter} = (0{,}40 \times 1{,}13) + (0{,}30 \times 0{,}49) + (0{,}30 \times 2)$$

$$\text{Puntuación Total de React Native} = (0{,}40 \times 3{,}62) + (0{,}30 \times 0{,}58) + (0{,}30 \times 6{,}6)$$

Obteniendo como resultado:

$$\text{Puntuación Total de Flutter} = 1{,}199$$

$$\text{Puntuación Total de React Native} = 3{,}602$$

Resultado normalizado a escala de 0 a 1:

$$\text{Puntuación Normalizada de Flutter} = 0{,}1199$$

$$\text{Puntuación Normalizada de React Native} = 0{,}3602$$

Estos resultados reflejan que Flutter ofrece un rendimiento significativamente mejor en comparación con React Native en los aspectos evaluados. React Native muestra un rendimiento adecuado a la norma en algunos aspectos, aunque Flutter se destaca como un *framework* más eficiente, especialmente en términos de uso de CPU y fluidez de la interfaz de usuario.

4.2.3. Valor Compatibilidad

Para el análisis de este aspecto se contabilizarán por tipo de compatibilidad, clasificando las bibliotecas en cada aspecto por framework.

- **Total**: existencia de SDK específico para el framework. Ponderación 2 al considerarse de máxima importancia, ya que la existencia de esta compatibilidad disminuye considerablemente el tiempo de desarrollo de la misma.
- **Parcial**: existencia de API REST o mediante desarrollo nativo adaptado al framework. Ponderación 1 por brindar la posibilidad de integrar la funcionalidad, aunque no de la mejor manera.
- **No Compatible**: no es compatible para la integración. Ponderación -1, ya que este aspecto limita la implementación de funcionalidades necesarias.
- **Probable***: existencia de biblioteca Javascript. Se conoce la existencia a ciencia cierta de su posible uso; sin embargo, puede ser compatible con React Native. Ponderación 0.5 por la posibilidad.

Las bibliotecas no se pueden clasificar por más de una de estas compatibilidades, han de clasificarse en la de mayor ponderación posible. Simplificando el estudio 4.1 en la tabla 4.4.

Tabla 4.4: Comparativa de rendimiento entre Flutter y React Native.

Variable	Ponderación	Flutter	React Native
Compatibilidad Total	2	5	6
Compatibilidad Parcial	1	3	2
No compatible	-1	0	0

La **Puntuación Total** se calcula como:

$$\text{Puntuación Total} = \sum_{i=1}^{n} (P_i \times C_i)$$

Donde:

- P_i es la ponderación de la variable i.
- C_i es la calificación del *framework* en la variable i.

Sustituyendo en cada caso:

$$\text{Puntuación Total de Flutter} = (2 \times 5) + (1 \times 3) + (-1 \times 0) = 10 + 3 + 0 = 13$$

$$\text{Puntuación Total de React Native} = (2 \times 6) + (1 \times 2) + (-1 \times 0) = 12 + 2 + 0 = 14$$

Obteniendo como resultado:

$$\text{Puntuación Total de Flutter} = 13$$

$$\text{Puntuación Total de React Native} = 14$$

La comparativa en este caso da ventaja a React Native, lo que se considera un aspecto de peso, por la importancia del requisito de integración de funcionalidades existentes. De igual forma, el coste de desarrollar un módulo donde se utilicen las funcionalidades nativas es elevado y riesgoso.

Para normalizar las puntuaciones a una escala de 0 a 1, utilizamos la fórmula:

$$\text{Puntuación Normalizada} = \frac{\text{Puntuación Total} - \text{Puntuación Mínima}}{\text{Puntuación Máxima} - \text{Puntuación Mínima}}$$

Donde utilizando un enfoque de los valores mínimos y máximos respecto a la escala total de evaluación:

- Puntuación Mínima es la puntuación total más baja posible.
- Puntuación Máxima es la puntuación total más alta posible.

Para los **datos** que tenemos de 7 bibliotecas:

- Puntuación Mínima = -8 (ninguna biblioteca compatible)
- Puntuación Máxima = 16 (compatibilidad total de todas las bibliotecas)

Puntuación **Normalizada**:

$$\text{Puntuación Normalizada de Flutter} = \frac{13 - (-8)}{16 - (-8)} = \frac{21}{24} = 0{,}875$$

$$\text{Puntuación Normalizada de React Native} = \frac{14 - (-8)}{16 - (-8)} = \frac{22}{24} \approx 0{,}917$$

4.2.4. Conclusión de sección

Para realizar la comparación cuantitativa multicriterio se utiliza una metodología de toma de decisión por criterio múltiple (del inglés *Multiple Criteria Decision Making (MCDM)*) mediante el método de la media aritmética ponderada. Para ello la suma de todas las ponderaciones debe ser 1, se normalizarán los valores obtenidos en las variables a una escala de 0 a 1.

Para evaluar y comparar los *frameworks Flutter* y *React Native*, utilizamos la media aritmética ponderada con ponderaciones para cada variable y se normalizan los valores obtenidos de cada *framework* en la tabla 4.5.

- **Valor Cualitativo**: 20 %
- **Rendimiento**: 30 %
- **Compatibilidad**: 50 %

Tabla 4.5: Valores Normalizados para cada Framework

Variable	Flutter	React Native
Valor Cualitativo	0.5925	0.665
Rendimiento	0.1199	0.3602
Compatibilidad	0.875	0.917

Cálculo de la **puntuación total**

La puntuación total se calcula utilizando la fórmula de la media aritmética ponderada:

$$\text{Puntuación Total} = \sum_{i=1}^{n} (\text{Valor Normalizado}_i \times \text{Ponderación}_i)$$

Donde:

- $\text{Valor Normalizado}_i$ es el valor normalizado del *framework* en la variable i.
- Ponderación_i es la ponderación de la variable i.

Sustituyendo la **puntuación total** por framework:

$$\text{Puntuación Total}_{\text{Flutter}} = (0{,}5925 \times 0{,}20) + (0{,}1199 \times 0{,}30) + (0{,}875 \times 0{,}50)$$

$$\text{Puntuación Total}_{\text{React Native}} = (0{,}665 \times 0{,}20) + (0{,}3602 \times 0{,}30) + (0{,}917 \times 0{,}50)$$

Obteniendo como **resultados** la tabla 4.6 siguiente:

Tabla 4.6: Puntuación total por Framework

Framework	Puntuación Total
Flutter	0.59197
React Native	0.69956

Conclusión:

Según la media aritmética ponderada, **React Native** obtiene una puntuación superior a **Flutter**, destacando especialmente en la compatibilidad, que tiene una mayor ponderación en este análisis. Por lo que se selecciona este como frámework para el desarrollo multiplataforma a realizar.

Esta decisión se respalda además, con la experiencia que del equipo de desarrollo que proviene de ReactJS, lo que brinda una ventaja con el lenguaje de programación y disminuiría la curva de aprendizaje; la amplia comunidad de React y React Native facilitaría la resolución de errores en los desarrollos de integración de bibliotecas. Además, destacó en el análisis de los resultados la posibilidad de contar con apoyo del equipo web para la utilización de los SDKs en React Native, ya que el *framework* utilizado por este es ReactJS.

4.3 Metodologías de trabajo

La gestión de proyectos es una disciplina esencial en el desarrollo de *software*, especialmente en entornos de desarrollo multiplataforma donde la complejidad y el riesgo suelen ser mayores. En esta sección, se explorarán los aspectos clave y las metodologías más comunes empleadas, enfocándose en cómo estas prácticas pueden influir en el éxito o fracaso de un proyecto de desarrollo multiplataforma.

El objetivo es proporcionar una visión general de las mejores prácticas y herramientas disponibles para la gestión efectiva de proyectos, que incluyen la gestión del alcance, la planificación, comunicación, la integración y entrega continua, entre otras áreas esenciales. Aunque el caso práctico presentado en este trabajo no cumple con muchas de estas prácticas ideales, es crucial entender las diferencias entre el enfoque teórico y la realidad del proyecto analizado. Las bases teóricas de esta sección está basado en la séptima edición de la Guía de los Fundamentos para la Dirección de Proyectos (Guía del PMBOK®) del *Project Management Institute* (PMI)[3].

Comenzamos bajo la premisa de la existencia de una organización de recursos humanos donde presentamos, el *business manager* quien se encarga de garantizar la rentabilidad de los encargos y la satisfacción del Cliente, el jefe de proyecto, un líder técnico, y los equipos Android, iOS y Web; a los que se incorporará el nuevo equipo de React Native, cada uno con un jefe de equipo.

4.3.1. Gestión de alcance

La gestión del alcance se enfoca en definir y controlar lo que está incluido y excluido en el proyecto. Para lograr esto, se requiere una cuidadosa recopilación de requisitos, que se realiza mediante reuniones con los interesados para comprender las necesidades y expectativas específicas para el desarrollo en React Native. De estas reuniones se obtiene una documentación de requisitos funcionales y no funcionales, utilizando técnicas como historias de usuario y casos de uso.

Luego, se redacta una declaración de alcance que plantea lo que se incluirá y lo que se excluirá del proyecto. Esta declaración debe ser específica para cada plataforma (en este caso, React Native) y debe ser aprobada por los *stakeholders* clave. Se debe descomponer el alcance en entregables más pequeños y manejables, definiendo también el alcance para cada uno de ellos, asegurando que cada entrega se alinee con los objetivos generales del proyecto.

Al inicio del desarrollo, cada equipo debe recibir una lista detallada de los entregables y las tareas específicas relacionadas con React Native. Se deben plantear los criterios de aceptación para cada entrega, asegurando el cumplimiento de los requisitos previamente definidos. Aunque el proyecto tiene un objetivo general común, cada equipo tiene entregables y alcances independientes pero alineados.

Para el control durante el proceso de desarrollo, se realiza un seguimiento continuo del progreso con respecto a los entregables, utilizando herramientas de gestión de pro-

yectos y realizando reuniones regulares para revisar el alcance. Los cambios en el alcance pueden ocurrir, por lo que se implementa un proceso de gestión de cambios. Este proceso asegura que todos los cambios sean evaluados y aprobados a nivel de proyecto, considerando los impactos en las plataformas involucradas.

Al finalizar el proyecto, se realiza una verificación final para asegurar que todos los entregables se han completado de acuerdo con la declaración de alcance para cada desarrollo por plataforma. Posteriormente, se recolectan las lecciones aprendidas, documentando lo que funcionó correctamente y los aspectos que podrían mejorarse para futuros proyectos.

Aplicar estos principios ayudará a gestionar eficazmente el alcance en un proyecto de desarrollo multiplataforma, garantizando que cada equipo pueda contribuir al objetivo general del proyecto mientras se manejan las complejidades de las diferentes plataformas.

En el proceso actual, la gestión del alcance se realiza de manera continua, aunque no siempre con todos los documentos requeridos. Se llevan a cabo revisiones en las que participan el *business manager*, el Cliente y el líder técnico. En estas revisiones se evalúa el progreso y se gestionan los cambios generados por la incertidumbre del proyecto. Entre las dificultades destaca la falta de una documentación específica de requisitos y entregables; los entregables se definen por *sprint* y los requisitos se conocen de manera general, no específica para cada entregable.

4.3.2. Planificación del proyecto

La planificación del proyecto es esencial para garantizar el cumplimiento de los objetivos. El PMI indica la planificación no solamente como la fase de inicio del proyecto, sino como un proceso continuo que se ajusta a las necesidades. Los pilares son: el enfoque de resultados y entrega de valor al cliente o interesados; la adaptabilidad al cambio ya que cada proyecto es único y puede requerir enfoques flexibles, ya sea mediante metodologías ágiles, híbridas o predictivas; es necesaria la revisión continua al ser un ciclo de retroalimentación y adaptación a medida que el proyecto avanza.

Las formas de enfocar la metodología de planificación de un proyecto pueden ser[3]:

Predictivo (tradicional o cascada): sigue un plan detallado que se establece al inicio del proyecto y se ejecuta de forma secuencial. Cuenta con una planificación detallada al inicio, un enfoque secuencial en el proceso de diseño contrucción y pruebas, con un estricto control de alcance. Este tipo de enfoque se utiliza en proyectos con bajo nivel de incertidumbre, como contrucción o ingeniería.

Ágil: se centra en la adaptabilidad y la entrega continua de valor en ciclos cortos. Entornos donde los requisitos y soluciones evolucionan a lo largo del proyecto, permitiendo ajustar el producto en función de la retroalimentación del cliente. Se caracteriza por ser incremental e iterativo con entregas frecuentes, con una planificación adaptativa y la colaboración constante con el cliente. Este enfoque es común en desarrollo de *software* y proyectos donde los requisitos cambian rápidamente.

Híbrido: este enfoque combina los enfoques predictivo y ágil permitiendo que distintas partes del proyecto se gestionen utilizando la metodología más adecuada. Una forma común de uso es con una planificación inicial sólida para los elementos predecibles, mientras que el desarrollo utiliza una metodología ágil para áreas con alta incertidumbre.

En general para la planificación de proyectos de desarrollo de *software* se utilizan metodologías ágiles, por la incertidumbre del proceso de desarrollo. Entre estas metodologías la metodología Scrum[43] se ha consolidado como una de las herramientas más

efectivas, permitiendo a los equipos de trabajo adaptarse rápidamente a los cambios y entregar valor continuo en sus proyectos. En lo que respecta al estudio de caso se utiliza Scrum actualmente en los proyectos en curso y será la metodología a utilizar para el desarrollo multiplataforma.

Sin embargo, exiten otras alternativas como son Kanban[3] y XP (del inglés *Extreme Programming*), las cuales pueden ser beneficiosas en algunos proyectos o equipos de desarrollo.

Por ejemplo, Kanban en comparación con Scrum es más flexible y fluido, no tiene *Sprints* predefinidos ni roles específicos. En su lugar, se enfoca en la visualización del flujo de trabajo en un tablero Kanban, donde las tareas pasan por tres etapas: "Por hacer", "En progreso" y "Completado". Su objetivo es mejorar la eficiencia del proceso y reducir el tiempo de entrega, adecuado en proyectos que tienen un flujo continuo de trabajo y cambios de prioridades frecuentes, como puede ser la gestión de servicios o mantenimiento de *software*.

Por otra parte, *Extreme Programming* (XP)[4] comparte principios fundamentales con Scrum como la entrega incremental, la retroalimentación continua y la adaptabilidad al cambio. Sin embargo, se diferencian en el enfoque y la prácticas, Scrum se enfoca a la planificación con *sprints* de 1 a 4 semanas, los roles y eventos claramente definidos, XP propone técnicas concretas para mejorar la calidad del código, como la programación en pareja, el desarrollo guiado por pruebas y la refactorización continua, poniendo el foco en la excelencia técnica y la mejora continua del producto desde el punto de vista del desarrollo. Al ser metodologías que se enfocan en aspectos distintos del proceso de desarrollo, a menudo se pueden ver combinadas fortaleciendo así tanto la gestión como la calidad técnica del producto.

Uno de los recientes desenlaces es el método Spotify [45], un enfoque ágil y flexible para la gestión de equipos de desarrollo, inspirado en la empresa Spotify[5]. Este modelo no es una metodología formal como Scrum o XP, sino más bien un conjunto de prácticas y principios que enfatizan la autonomía de equipos, la innovacion y la colaboración. Los conceptos que se implementan son:

- Escuadrones (*Squads*): Equipos pequeños y autónomos que trabajan en objetivos específicos y usan metodologías ágiles como Scrum o Kanban.
- Tribus (*Tribes*): Grupos de escuadrones que trabajan en áreas relacionadas del producto. Cada tribu tiene un líder de tribu que asegura la coordinación y alineación, sin perder la agilidad de los equipos pequeños.
- Capítulos (*Chapters*): Agrupan a personas con habilidades similares de diferentes escuadrones para mantener la coherencia técnica y el intercambio de conocimientos entre escuadrones.
- Gremios (*Guilds*): Comunidades informales que comparten intereses comunes, como prácticas de ingeniería, calidad o agilidad. Estos gremios fomentan la innovacion y colaboración a lo largo de la organización.

En la metodología Spotify la autonomía de los equipos es una ventaja, donde estos tienen la libertad de tomar decisiones sobre cómo alcanzar sus objetivos. Al mismo, tiempo el enfoque de alineación asegura que los escuadrones y tribus trabajen hacia metas compartidas para cumplir con la estrategia general de la empresa. La escalabilidad, con una

[3] https://kanbantool.com/es/metodologia-kanban
[4] http://www.extremeprogramming.org
[5] https://www.spotify.com/es

estructura adaptable para grandes organizaciónes, es otra ventaja de este método donde se promueve la innovación, fomentando que los equipos experimenten. Sin embargo, entre las desventajas de este método es la compleja coordinación entre equipos y la falta de un marco formal al no ser una metodología rígida, lo que puede generar confusión si no están bien definidos los objetivos y la cultura organizativa.

En resumen, el método escogido para los proyectos de desarrollo en el estudio de caso es correcto, ya que se alinea con las necesidades del proyecto; sin embargo se propone estudiar y analizar el surgiente método Spotify para los proyectos que requieran de innovacion y participación técnica de distintas áreas, ya que puede resultar más productivo que un método rígido como es Scrum.

4.3.3. Gestión de la comunicación

La gestión de la comunicación es clave para garantizar que toda la información relevante se transmita de manera oportuna y efectiva a los miembros del equipo, los clientes y otras partes interesadas. En el estudio de caso actual, además, se requiere la comunicación entre los equipos de Android, iOS y Web para la implementación de las integraciones nativas de las bibliotecas que no presentan soporte directo para React Native y el consumo de la API REST del cliente. Una comunicación deficiente atrasaría el desarrollo afectando directamente la planificación.

Para lograr una comunicación efectiva el PMI define estrategias para equipos multidiciplinarios a modo de guía:

- Plan de comunicación: define canales de comunicación, audiencia, expectativas de frecuencia, los medios de comunicación y responsabilidades.
- Usar tecnologías colaborativas: herramientas como Slack[6], Jira[7] o Trello[8], permiten la comunicación efectiva y ágil en el equipo además de lograr una documentación y seguimiento, independiente de la disciplina o la ubicación.
- Reuniones periódicas: las reuniones regulares, como *dailies* o *retrospectives*, permite alinear a todo el equipo y resolver cualquier duda o problema de forma colaborativa.
- Lenguaje común: en la comunicación entre disciplinas con terminologías diferentes pueden surgir conflictos de terminologías, se evita estableciendo un lenguaje común que puede incluir glosarios de términos o guías comunes de comunicación.

Una buena comunicación reduce los conflictos y permite la resolución temprana de problemas. La comunicación facilitada por un entorno seguro, es esencial para encontrar soluciones colaborativas. Un líder de proyecto o Scrum *Master* puede ayudar a mediar, asegurando que todos los puntos de vista sean considerados. Tener claras las responsabilidades y procedimientos asegura que todos conozcan lo que se espera de ellos.

En el proyecto actual, la comunicación con el Cliente, por lo general, es a través de comentarios en las tareas de Jira o por correo electrónico. A través de los comentarios se consultan dudas y de esta forma se deja constancia en las propias tareas de lo discutido. Los correos electrónicos se utilizan para comunicaciones más formales o agendar reuniones. Para agendar reuniones se ha de comentar siempre con el jefe de proyecto y añadirle

[6] https://slack.com/intl/es-es
[7] https://www.atlassian.com/es/software/jira
[8] https://trello.com

como opcional al evento, asegura la alineación y seguimiento de discusiones técnicas y estratégicas.

La comunicación interna del equipo se basa en reuniones diarias que forman parte de la metodología Scrum, donde se informa el estado actual de las tareas y su continuación. Estas reuniones también son el método de comunicación de la administración con los equipos para sincronizar el progreso del proyecto, alinear las prioridades y detectar posibles bloqueos o cuellos de botella. De esta manera, se asegura una coordinación fluida entre todos los miembros y se mantiene un enfoque claro hacia los objetivos del proyecto.

Sin embargo, se encuentran deficiencias en esta comunicación tanto por parte del cliente como del equipo:

- Falta de claridad en la información transmitida: esto afecta la velocidad de respuesta y se puede solucionar implementando una política de documentación detallada, donde se abarquen distintos tipos de incidencias y de cada una se describa la información requerida. Esta política se retroalimentaría al final de cada *sprint* para mejorar y ampliar las posibles incidencias, facilitando así la colaboración.

- Respuestas tardías: en algunos casos se ha dado la tardanza en respuestas, sobre todo en los comentarios de Jira y, en caso de urgencia, se recurre al correo electrónico para enfatizar su importancia, lo cual siempre ha sido exitoso. Aunque no son muchas las ocurrencias, puede ser de ayuda la definición de tiempos de respuesta con el Cliente desde el principio del proyecto y acciones a tomar en dependencia de la urgencia de la consulta.

- Barreras de interacciones entre proyectos: la comunicación entre equipos de otras tecnologías no tiene un procedimiento definido, por lo que en ocasiones donde se requiere consultas se encuentran las horas ocupadas en tareas de sus desarrollos. La solución que se propone para este caso es elaborar un plan de acción donde, a partir de la planificación del *sprint*, aunque en el inicio no requiera intervención, si se prevee la posibilidad de necesitarlo, planificar un porcentaje de horas del *sprint* en ambos equipos a la interación y agendar tentativamente una reunión semanal para consultas.

4.3.4. Gestión de riesgos

La gestión de riesgos, tal como lo define el PMI, abarca los procesos necesarios para planificar, identificar, analizar, responder y monitorear los riesgos durante el ciclo de vida del proyecto. Su objetivo principal es minimizar el impacto de las amenazas y maximizar las oportunidades.

La identificación y gestión temprana de los riesgos permite anticipar problemas potenciales antes de que se conviertan en realidades costosas y difíciles de manejar. Adoptar un enfoque sistemático en la gestión de riesgos, permite al gestor de proyectos planificar y tomar decisiones más informadas, basadas en datos y análisis.

En el estudio de caso actual, la gestión de riesgos ha sido un componente ausente, a pesar de su importancia, especialmente debido a la naturaleza del proyecto. El desarrollo multiplataforma conlleva un riesgo elevado, dado que depende de la integración de bibliotecas de terceros que deben ejecutarse en diversas plataformas a partir de un único código base. Adicionalmente, cuando las bibliotecas no son compatibles con el framework, es necesario desarrollar soluciones nativas, lo que añade complejidad e incertidumbre al proyecto.

La propuesta de implementación de una gestión de riesgos para este tipo de proyecto comienza con la asignación de roles y responsabilidades claras para quienes se encargarán del seguimiento de los riesgos. Asimismo, es crucial definir los métodos de evaluación que se emplearán.

Identificar los riesgos es fundamental antes de iniciar el desarrollo, participan en el proceso recursos humanos con experiencia en el sector, así como miembros técnicos del equipo. Las herramientas comúnmente utilizadas incluyen lluvias de ideas, análisis Debilidades, Amenazas, Fortalezas y Oportunidades (DAFO), entrevistas y listas de verificación[3]. Una vez identificados los riesgos, se analizan en función de su probabilidad de ocurrencia y su impacto, lo que permite priorizarlos según su criticidad. Para este análisis, pueden emplearse herramientas estadísticas como el análisis de Monte Carlo o el valor monetario esperado (EMV), que son de las más utilizadas[16].

Una vez documentados, se deben planificar las respuestas a los riesgos para estar preparados en caso de que ocurran. Dependiendo de la naturaleza del riesgo, las estrategias pueden incluir evitar, mitigar, transferir o aceptar el riesgo.

Esta preparación y documentación, junto con un enfoque proactivo y continuo, permiten a la gestión del proyecto anticiparse a las posibles afectaciones, minimizando su impacto en el coste y mejorando la toma de decisiones. Además, se maximiza la posibilidad de aprovechar oportunidades. Considero que, dado el alto nivel de incertidumbre asociado con este proyecto, la gestión de riesgos sería un componente esencial para su éxito.

4.3.5. Gestión de recursos humanos

Siguiendo las directrices del PMBOK[3], la gestión de recursos humanos no solo implica la asignación y coordinación de tareas, sino también la motivación, desarrollo profesional y bienestar del equipo de trabajo. En el contexto de un proyecto de desarrollo de *software*, donde los equipos multidisciplinarios trabajan en entornos de alta complejidad y constante cambio, la gestión efectiva de recursos humanos se convierte en un factor decisivo para el éxito. Esto incluye la selección adecuada de los miembros del equipo, la definición de roles y responsabilidades, y la creación de un ambiente que fomente la colaboración y el rendimiento óptimo.

Para el proyecto este elemento se divide en varias áreas clave:

Seguimiento y supervisión de consultores: El jefe de proyecto lleva a cabo un seguimiento diario de los consultores para evaluar su rendimiento, detectar signos de fatiga o descontento, y asegurarse de que las tareas se estén realizando correctamente. Este seguimiento es esencial para mantener la moral alta, garantizar eficiencia e identificar posibles problemas antes de que se conviertan en obstáculos significativos para el proyecto.

Gestión de relaciones con *managers*: En Alten el *business manager* es responsable de supervisar los recursos humanos necesarios para completar con éxito los proyectos, contratando y gestionando a los consultores y consultoras.

Planificación y distribución de la carga de trabajo: El proyecto se organiza en *sprints* de dos semanas, enfocados en completar la mayor cantidad posible de *Minimum Viable Product* (MVP). La planificación de estos *sprints* es clave para gestionar eficientemente la carga de trabajo del equipo. El *Product Owner* del cliente desempeña un papel vital en la negociación y distribución de tareas, utilizando el *backlog* y el dimensionamiento del equipo para asegurar un buen trabajo y el cumplimiento de los plazos establecidos.

Optimización de la planificación: La planificación inicial puede ser modificada y optimizada según las directrices del cliente, que se alinean con sus estrategias comerciales.

Esto requiere una gestión ágil y flexible del equipo, adaptándose rápidamente a los cambios en las prioridades del cliente.

Gestión de capacidad del equipo: En situaciones donde la capacidad del equipo no es suficiente para cumplir con las expectativas del *sprint*, se tiene la opción de incorporar recursos adicionales desde la dirección técnica. Esto asegura que los objetivos del *sprint* se cumplan, incluso si se requieren ajustes en la composición del equipo.

A modo general, la gestión de recursos humanos en este proyecto es esencial para asegurar que el equipo esté alineado con los objetivos y que se mantenga motivado y productivo. La supervisión diaria, la planificación cuidadosa y la flexibilidad para adaptarse a las necesidades cambiantes son la base de esta gestión.

4.3.6. Gestión de la calidad

La Gestión de la Calidad en la gestión de proyectos, según la séptima edición del PMBOK[3], se refiere a las actividades y procesos necesarios para asegurar que los entregables del proyecto cumplan con los requisitos y estándares de calidad establecidos por los *stakeholders* y la organización. El objetivo principal de la gestión de la calidad es satisfacer o superar las expectativas del cliente y otras partes interesadas mediante la entrega de un producto adecuado para su propósito.

Los principios claves de la gestión de calidad son: enfoque en el cliente, mejora continua, responsabilidad de todos y enfoque basado en procesos. Esto implica comprender las expectativas del cliente desde el inicio hasta el cierre del proyecto, la revisión y mejora constante, la concienciación de un equipo comprometido en la entrega de un trabajo de alta calidad y una correcta identificación, medición y control de los procesos. Todo esto garantiza resultados consistentes y predecibles en aras de un producto de calidad. En el caso de estudio se cuenta con un equipo de QA (del inglés *Quality Assurance*) el cual es responsable de la gestión de calidad.

Como parte de la estrategia de calidad llevada a cabo, el componente de planificación incluye la creación y diseño de planes de prueba basados en los documentos funcionales proporcionados por el cliente. El equipo de Quality Assurance (QA) analiza minuciosamente estos documentos para identificar todas las casuísticas posibles, asegurando que cada aspecto del producto sea validado adecuadamente. Se utiliza *Testlink*[9] como herramienta para diseñar y documentar los casos de prueba; *Testlink* permite la creación de *suites* de pruebas que cubren tanto las pruebas de regresión, como las pruebas para nuevos evolutivos. Además, JIRA se emplea para la gestión y seguimiento de las incidencias detectadas durante el proceso de *testing*, aportando al registro de la evolución de las tareas.

El control de calidad se logra mediante metodologías de pruebas donde se involucra el equipo de desarrollo. Las pruebas unitarias realizadas por el equipo de desarrollo se enfocan en verificar que cada unidad o componente individual del *software* funcione correctamente. Posteriormente, el equipo de QA realiza pruebas funcionales y de regresión para asegurar que las funcionalidades existentes del *software* sigan funcionando correctamente y evolutivas para validar nuevas funcionalidades o cambios introducidos. Estas pruebas, basadas en los casos diseñados en *Testlink*, se ejecutan en un entorno de preproducción, asegurando que el producto cumple con los estándares de calidad antes de su liberación.

El aseguramiento de calidad se lleva a cabo mediante revisiones periódicas de los planes de prueba y su ejecución, para verificar que se cubren todos los casos de uso rele-

[9] https://testlink.org

vantes. Cualquier incidencia detectada se documenta en JIRA y se realiza un seguimiento exhaustivo hasta su resolución. Durante todo el ciclo de vida del proyecto, el equipo de QA proporciona *feedback* continuo al equipo de desarrollo, lo que permite realizar ajustes y correcciones de manera proactiva.

Los defectos detectados durante las pruebas se documentan en JIRA, donde se priorizan y se realiza un seguimiento. Como parte de la gestión de incidentes, el flujo de trabajo en JIRA permite el seguimiento eficiente de las incidencias, garantizando que todos los "KO" (errores) se resuelvan antes de la entrega final, en colaboración con el equipo de desarrollo.

Al finalizar cada *sprint*, el equipo de QA revisa las lecciones aprendidas del proceso de pruebas y las documenta para mejorar la planificación y ejecución en futuras iteraciones, garantizando la mejora continua de la gestión de calidad.

La correcta implementación de la gestión de la calidad es vital para el éxito de cualquier proyecto. Asegura que los entregables no solo cumplan con las expectativas del cliente, sino que también se adhieran a los estándares y regulaciones aplicables. Además, una fuerte gestión de la calidad contribuye a reducir costos, evitar retrabajos, y mejorar la satisfacción del cliente, lo que en última instancia puede llevar a la repetición de negocios y una reputación positiva para la organización.

4.3.7. Integración Continua y Entrega Continua (CI/CD)

La gestión de CI/CD (del inglés *Continuous Integration/Continuous Delivery*) se refiere a una serie de prácticas y herramientas utilizadas para automatizar y mejorar el ciclo de desarrollo y despliegue de *software*, asegurando que los cambios en el código sean integrados, probados y desplegados de manera rápida, eficiente y fiable.

En el proyecto actual para este componente se sigue el estándar de mensajes de confirmación de cambios definido por *Conventional Commits*[10]. Bitbucket[11] se utiliza para la gestión del código y el control de versiones. Está integrado con Jira, lo que permite una visibilidad completa de las tareas y el seguimiento del progreso del proyecto directamente desde la plataforma de gestión de proyectos. Jenkins[12] es la herramienta empleada por el Cliente para la automatización de la integración continua y el despliegue continuo. Jenkins ejecuta las *pipelines* necesarias para la integración de nuevas versiones y el despliegue automatizado en producción, asegurando un flujo de trabajo eficiente y continuo; de esto se encarga el cliente por contrato.

Para el versionado, se sigue la convención establecida en Versionado Semántico[13]. Tras cada *sprint*, se actualiza la versión *MINOR* para reflejar las nuevas funcionalidades y mejoras implementadas, manteniendo un historial claro y organizado de las versiones del producto.

La comunicación interna del equipo se realiza principalmente a través de **Microsoft Teams**, donde se notifican las solicitudes de *Pull Requests* y los nuevos *merges* en la rama *develop*. Esto asegura una rápida colaboración entre los desarrolladores y mantiene a todos los interesados al tanto de los cambios importantes en el código base.

Gestión de entornos

[10] https://www.conventionalcommits.org/en/v1.0.0/
[11] https://bitbucket.org
[12] https://www.jenkins.io
[13] https://semver.org/lang/es/

En el proyecto se utilizan tres entornos diferenciados: Producción, Pre-producción y Pruebas. Cada entorno cumple un rol específico en el ciclo de vida del desarrollo y despliegue de *software*, y cada uno está asociado con licencias de *software* y API particulares.

El entorno de Producción (Pro) es el entorno final donde la aplicación está disponible para los usuarios finales. Aquí se implementan las versiones estables y completas del *software*, y cualquier error o problema puede impactar a los usuarios finales.

El entorno de Pre-producción (Pre) se utiliza para realizar pruebas finales antes del despliegue en Producción. Es un entorno que simula el entorno de Producción lo más fielmente posible, con el fin de validar que el *software* funciona correctamente bajo condiciones similares a las reales.

Por último el entorno de Pruebas es utilizado por los desarrolladores y *testers* para validar nuevas funcionalidades, realizar pruebas unitarias, de integración y de sistema. Este entorno es crítico para detectar y corregir errores antes de pasar a los entornos de Pre-producción y Producción.

Los despliegues de la aplicación en entornos de producción y pre-producción son responsabilidad del cliente y están automatizados a través de Jenkins, lo que garantiza una integración y entrega continua más fluida. Mientras que en entorno de pruebas es responsabilidad del equipo y se realizan utilizando Applivery[14], aunque no están automatizados, este servicio facilita la distribución y gestión de versiones de prueba a los *testers* y otros interesados.

Cada uno de estos entornos juega un papel crucial en asegurar que el *software* se despliegue de manera efectiva y sin problemas. La correcta gestión de licencias y API en cada entorno es esencial para garantizar el cumplimiento de los requisitos y la estabilidad del *software* a lo largo de su ciclo de vida.

Automatización de pruebas y calidad de código

En cuanto a las pruebas automatizadas, el equipo utiliza Jest[15] para realizar pruebas unitarias y de integración, asegurando que cada nueva funcionalidad y cambio de código mantenga la estabilidad del sistema y no introduzca nuevos errores. Las pruebas funcionales, por otro lado, son realizadas manualmente por el especialista de QA. Este especialista también utiliza herramientas como *Charles Proxy*[16] para verificar audiencias y monitorear el tráfico de red durante las pruebas.

Para garantizar la calidad del código, se implementa ESLint[17] con la siguiente configuración, que sigue las mejores prácticas para proyectos React Native:

```
module.exports = {
  root: true,
  extends: '@react-native',
};
```

Además, se emplea Prettier[18] para formateo automático del código con la configuración siguiente, manteniendo un estilo de código consistente en todo el equipo:

```
module.exports = {
  arrowParens: 'avoid',
```

[14] https://www.applivery.com/es/
[15] https://jestjs.io/es-ES/
[16] https://www.charlesproxy.com
[17] https://eslint.org
[18] https://prettier.io

```
  bracketSameLine: true,
  bracketSpacing: false,
  singleQuote: true,
  trailingComma: 'all',
  semi: true,
  singleAttributePerLine: true,
  printWidth: 100,
  endOfLine: "auto"
};
```

Este enfoque exhaustivo asegura que cada etapa del ciclo de vida del *software* esté controlada, automatizada y gestionada de manera eficiente, reduciendo los riesgos y optimizando el proceso de entrega de nuevas funcionalidades.

4.4 Conclusiones

En este capítulo, se ha realizado un exhaustivo análisis de las estrategias y *frameworks* disponibles para la migración a un entorno multiplataforma, con un enfoque particular en Flutter y React Native. Se ha evaluado cada opción en términos de rendimiento, compatibilidad, y adecuación a los requerimientos del proyecto, así como se han presentado las metodologías de gestión y los recursos necesarios para llevar a cabo la implementación.

Este análisis fue necesario para seleccionar las herramientas y el enfoque correctos que no solo aseguren una migración exitosa, sino que también proporcionen una base sólida para el desarrollo futuro y la optimización continua del sistema. Las decisiones y procedimientos que se incluyen impactan directamente en la eficiencia, escalabilidad y sostenibilidad del proyecto.

Con estos conceptos claramente establecidos, estamos preparados para avanzar hacia la fase de implementación, donde se pondrán en práctica las estrategias seleccionadas. En el siguiente capítulo, se detallará la ejecución del proyecto de las tareas relacionadas con integraciones de bibliotecas, incluidas en el desarrollo del prototipo, las pruebas realizadas, y los ajustes necesarios para alcanzar los objetivos planteados.

CAPÍTULO 5
Implementación y desarrollo

En este capítulo se describen las funcionalidades desarrolladas para el prototipo en la aplicación multiplataforma, especialmente la integración de bibliotecas de terceros.

El equipo de desarrollo multiplataforma para la aplicación de React Native se conforma por tres desarrolladores y un líder de equipo; con la supervisión del líder técnico. Este equipo se incluye dentro del proyecto de aplicaciones móviles que se lleva a cabo para el Cliente, donde se incluyen tambien los equipos existentes de desarrollo Android e iOS. Además, existe un encargado de calidad para el proyecto, el cual valida los resultados de cada *sprint*.

El proyecto se comienza a realizar a partir de los requisitos del cliente y se cuenta con un *wireframe* proporcionado por el equipo de producto. En este trabajo nos enfocaremos en las tareas que me fueron asignadas como parte del equipo de desarrolladores de React Native. Mis tareas en esté período, abarcaron las integraciones de bibliotecas de terceros en el *framework* multiplataforma.

5.1 Prototipo

Para el desarrollo del prototipo la integración de bibiotecas se separa en dos tipos, relacionados con el **reproductor de medios** (de aquí en adelante *"player"*) o con la **estructura** de la aplicación.

Tabla 5.1: Bibliotecas a integrar por tipo

Player	**Estructura**
Bitmovin *player*	ComScore
ComScore	*Adobe Analytics*
Adobe Analytics	*SmartAd Server*
Youbora	*Azure Active Directory*
SmartAd Server	Didomi
StatsAPI*	

* : API del cliente

Algunas bibliotecas se repiten en ambos tipos de integraciones, esto se debe que se utilizan para rastreos de eventos del *player* y de la actividad en la aplicación. Dado que la reproducción de medios es una funcionalidad clave en el producto, se prioriza en la planificación la integración de bibliotecas relacionadas con este. Además de integrar bibliotecas de terceros, también se realiza la integración de un servicio proporcionado por el cliente para el seguimiento de uso que llamaremos StatsAPI. El anexo A contiene el

diagrama de Gantt de las tareas estimadas por cada integración requerida para el *player*,correspondiente a la primera fase.

En el desarrollo se combinan dos patrones de diseño, el patrón Observador y el patrón Mediador. Al ser mayormente bibliotecas de seguimiento y marcaje, se ejecutan cuando se detonan eventos en el reproductor y cada biblioteca es manejada por un servicio que encapsula su lógica. Los elementos encontrados en el patrón Observador son el *player* (*subject*) quien notifica a los servicios (*observers*) sobre los eventos; aquí es donde se incluye el patrón Mediador con un *handler* de eventos (*mediator*) que centraliza la comunicación entre el *player* y los servicios, gestionando qué acciones se toman cuando se recibe un evento (figura 5.1).

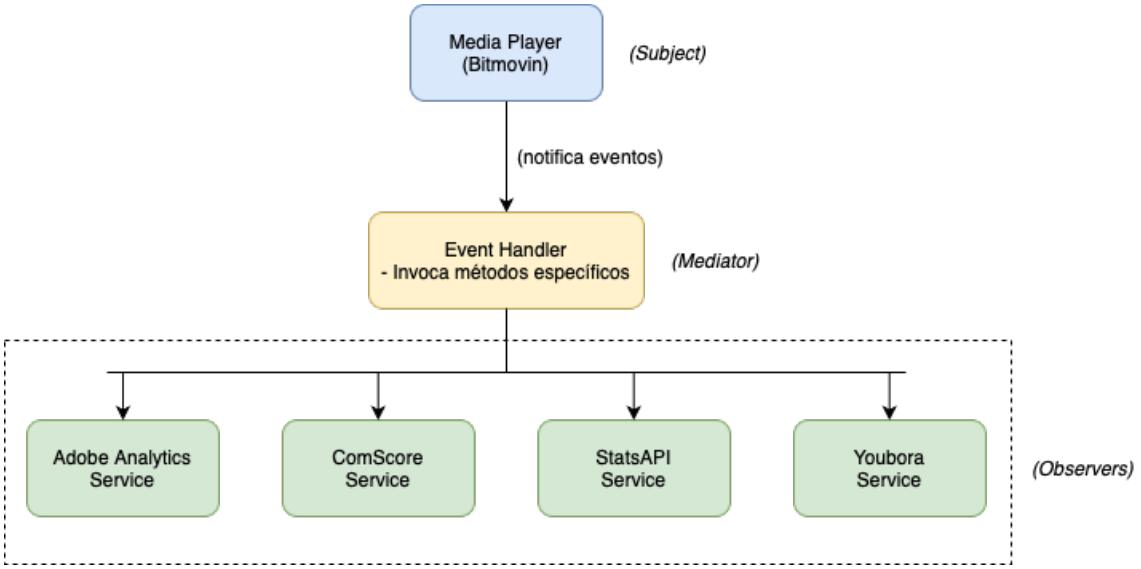


Figura 5.1: Patrón Observador y Medidador combinados.

Antes de entrar en el detalle de cada integración, es importante señalar que se consideran tres métodos principales para llevar a cabo la integración de bibliotecas. El primero y más eficiente es la integración directa, que utiliza un SDK específico para el *framework* React Native. El segundo método implica el uso de API REST, que permite la integración mediante servicios web. Finalmente, el tercer enfoque es la implementación de un *Turbo Module*, que actúa como un puente para utilizar SDK nativos de Android e iOS. Este enfoque se emplea cuando la biblioteca no dispone de un SDK para React Native y debe ser compatible tanto con la arquitectura antigua como con la nueva de módulos de React Native, asegurando así la compatibilidad hacia atrás y evitando la deuda técnica. Esta estrategia sigue las directrices del grupo de trabajo "*React Native New Architecture*"[1], garantizando una integración robusta y mantenible.

5.1.1. Bitmovin player

Reproductor de medios avanzado que soporta una amplia gama de formatos de video y audio. Permite reproducción en pantalla completa, *Picture-in-Picture*[2] y en segundo plano. Diseñado para proporcionar una experiencia de usuario de alta calidad en la reproducción de contenido multimedia. Se utilizará la última versión de la biblioteca "0.28.0".

[1] https://github.com/reactwg/react-native-new-architecture/tree/main

[2] Picture-in-picture es una función que permite ver un video en una ventana pequeña sobre otras aplicaciones mientras se realiza multitarea en un dispositivo.

El reproductor de video esencial en el producto se implementa mediante la SDK del proveedor, como funcionalidades básicas se espera que cumpla:

- Reproducción de medios bajo demanda.
- Reproducción de medios en vivo.
- Manejo de errores de carga de medios.
- Reproducción en pantalla completa.
- Funcionamiento en segundo plano.
- *Picture-in-Picture*.
- Diseño de interfaz correspondiente al realizado en Web.

La información relacionada con la reproducción de medios se obtiene a través de una API REST proporcionada por el cliente. Con esta información, se configura todo lo necesario para la reproducción de video y audio. Este proceso se realizó sin inconvenientes, tanto para formatos en vivo como bajo demanda y el manejo de errores en la carga de medios.

Sin embargo, las funcionalidades relacionadas con pantalla completa, audio en segundo plano y *picture-in-picture* presentaron dificultades. El SDK de React Native proporcionado por Bitmovin aún no soluciona estas funciones adecuadamente. Esto llevó a modificaciones en la planificación y creación de nuevas tareas, teniendo que atrasar las siguientes integraciones. Para implementar estas características, se utilizaron los eventos que proporciona Bitmovin y se buscaron soluciones estructurales en la aplicación. Aunque se logró la reproducción en pantalla completa y *picture-in-picture* (la cual funcionaba correctamente en dispositivos iOS), la funcionalidad de audio en segundo plano resultó más complicada.

Se consigue la reproducción de audio cuando la aplicación pasa a segundo plano; sin embargo, la interfaz de control que debería aparecer en las notificaciones o cuando la pantalla está apagada no se logra. La acción tomada en este caso fue la consulta a soporte de Bitmovin para encontrar una solución y en el tiempo de espera. Para continuar el desarrollo se inicia la preparación de datos para la integración de *Adobe*. Una vez recibida la respuesta negativa, ya que el reproductor no ofrece esta funcionalidad en la actualidad, se intentó implementar un *Turbo Module* para solventarlo, creando nuevas tareas para cada desarrollo nativo, utilizando *Media Session* para Android y *Remote Command Center* para iOS, que son bibliotecas que permiten la función de interfaz de controlador de medios en dispositivos, similar a como se utiliza en los desarrollos nativos. No obstante, el intento fracasó debido a la necesidad de una instancia de *player* en el módulo nativo de Android, lo cual es inviable dado que Bitmovin está desarrollado en React Native y duplicar esta implementación iría en contra de los principios de desarrollo multiplataforma. Esta tarea se desestimó y se estudiarán posibles soluciones en el futuro.

Para la personalización del diseño de la interfaz de usuario, se utilizó el proyecto "Bitmovin Player UI"[3] con el apoyo del equipo de desarrollo web en varias sesiones coordinadas. Este equipo, con su conocimiento en el diseño y los métodos de estilos de Bitmovin, contribuyó a la implementación exitosa de un diseño personalizado.

Esta biblioteca resultó de compleja, ya que se encuentra aún en desarrollo el soporte a React Native, el cliente es conciente de estas limitaciones y se aceptan las soluciones temporales brindadas y las negativas de algunos funcionamientos.

[3] https://github.com/bitmovin/bitmovin-player-ui

5.1.2. *Adobe Analytics*

Adobe Analytics[4] es una herramienta robusta de análisis de datos que permite a las empresas obtener información detallada sobre el comportamiento de sus usuarios a través de múltiples canales. Dentro de este entorno, la arquitectura *Adobe Experience Platform* (AEP) y la *Edge Network* juegan un papel crucial.

AEP: Es una plataforma de experiencia del cliente en tiempo real que unifica los datos de múltiples fuentes para ofrecer experiencias personalizadas en todos los canales.

Edge Network: Es una red distribuida que permite el procesamiento y la entrega de datos a baja latencia, asegurando que las interacciones del usuario se analicen y respondan casi en tiempo real.

La documentación de Analytics[5] define dos métodos de implementación, y presenta un diagrama representativo del flujo de trabajo:

Métodos de implementación de Edge (recomendados): Este método incluye el SDK de *Media* para *Edge Network* y la API de *Media Edge*. Este enfoque se recomienda tanto para *Customer Journey Analytics* como para *Adobe Analytics*, especialmente para las nuevas implementaciones (figura 5.2).

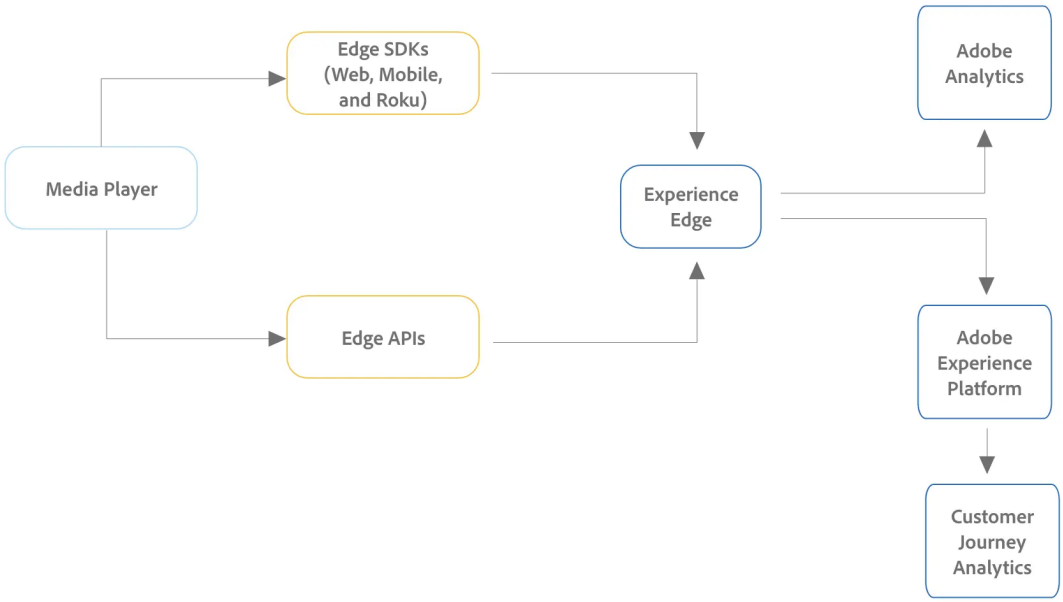


Figura 5.2: Flujo de implementación *Media* utilizando *Edge*.

Métodos de implementación solo de Adobe Analytics: Estos métodos fueron diseñados para su uso con *Adobe Analytics* y aún pueden integrar sus datos en *Customer Journey Analytics* mediante un *Analytics Source Connector*. Entre ellos se encuentra el Media SDK que se utiliza en las implementaciones actuales de Android e iOS, por lo que puede ser una solución a valorar aunque estaría agregando una deuda técnica al proyecto (figura 5.3).

Esta integración llega con retraso en la planificación y con tareas aisladas, ya que la preparación de los datos y el uso del servicio se planificaron en tareas separadas, debido a los inconvenientes encontrados con Bitmovin, el resultado es la finalización de la tarea

[4]https://business.adobe.com/es/products/analytics/adobe-analytics.html
[5]https://experienceleague.adobe.com/es/docs/media-analytics/using/implementation/overview

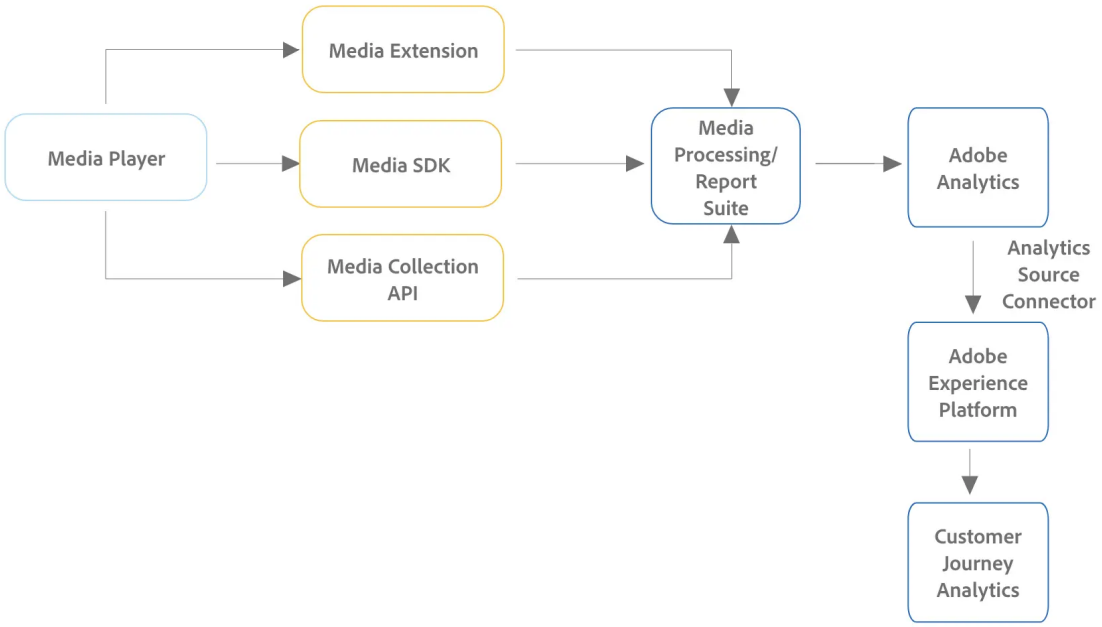


Figura 5.3: Flujo de implementación Media utilizando solo Analytics.

de preparación de datos que no presentó inconvenientes y seguido a ello se retomó una tarea de Bitmovin, para luego volver a la integración de *Adobe* esto influye negativamente en el tiempo por cambio de contexto.

Para la integración de *Adobe Analytics* en el reproductor, se evaluaron estas opciones y se optó por utilizar el SDK de React Native para *Edge*, siguiendo las recomendaciones de *Adobe* para nuevas implementaciones. Sin embargo, la migración a esta nueva implementación requiere reconfigurar parámetros en el servidor, ya que las implementaciones actuales utilizan el *Media* SDK de versiones anteriores. Esto presenta un desafío adicional, dado que la gestión de *Adobe Analytics* del cliente es manejada por un tercero. Tras consultar con dicho tercero la viabilidad de implementar *Adobe Analytics* en React Native mediante el SDK de *Edge*, se recibió una respuesta positiva, y se procedió con la implementación.

A pesar de que se logra inicializar el marcaje de la apliación y la conexión con *Adobe*, el principal desafío encontrado fue la ausencia del módulo *Media* en el SDK para el marcaje de medios, lo que lleva a considerar la integración mediante eventos de *Edge*. Sin embargo, este enfoque resultó en grandes retrasos y falta de avances, lo que llevó a abandonar la implementación con *Edge*. En su lugar, se propuso al cliente desarrollar un módulo puente en React Native para utilizar el Media SDK que ya funciona en las aplicaciones nativas. El cliente bloqueó esta tarea, esperando una consulta adicional con el equipo encargado de las analíticas de *Adobe*, en busca de una solución más concreta y documentada que permita la integración con la tecnología ofrecida por *Adobe*. Todos estos cambios e incertidumbre provocan muchas modificaciones en la planificación y retrasos para el proyecto, se crean nuevas tareas para abordar los problemas, la tarea de integración con *Edge* mantiene bloqueada y una vez se conozca la posible solución se incluirá en algún *Sprint* para abordarla.

5.1.3. ComScore

Plataforma de medición y análisis que proporciona información sobre la audiencia y el rendimiento, se enfoca en medir el alcance y la eficacia de las campañas publicitarias, así como el comportamiento de los usuarios en medios digitales.

ComScore, al no poseer soporte directo para React Native, requiere el uso de *Turbo Modules* para crear un puente entre el código TypeScript de React y el código nativo de Android e iOS, respectivamente. Siguiendo la guía del grupo de trabajo para la nueva arquitectura de React Native, se creará un módulo llamado `react-native-client-comscore`.

Inicialización del módulo

Para inicializar el módulo, se utiliza el siguiente comando, con las opciones necesarias para asegurar la compatibilidad con la nueva y antigua arquitectura de React Native:

```
npx create-react-native-library@latest react-native-client-comscore
```

Durante la ejecución de este comando, se seleccionan las siguientes opciones:

Pregunta: Looks like you're under a project folder. Do you want to create a local library?
Respuesta: `Yes`

Pregunta: Where do you want to create the library?
Respuesta: `modules/react-native-client-comscore`

Pregunta: What is the name of the npm package?
Respuesta: `react-native-client-comscore`

Pregunta: What is the description for the package?
Respuesta: `react native bridge for client com score`

Pregunta: What type of library do you want to develop?
Respuesta: `Turbo module with backward compat`

Pregunta: Which languages do you want to use?
Respuesta: `Kotlin & Objective-C`

Estructura del módulo

Al utilizar el comando de creación de módulo se genera una carpeta `modules/react-native-client-comscore` en la raíz del proyecto. Esta contiene tres carpetas: `android`, `ios` y `src`; esto conforma el puente para React Native, donde `src/index.ts` es la interfaz Typescript que se expone a React, mientras que las implementaciones nativas se desarrollan en las otras carpetas.

En la figura 5.4 se muestran únicamente los ficheros relevantes para la implementación. Además de estos, existen archivos de configuración como: `package.json`, `turbo.json`, `react-native-client-comscore.podspec`, `android/gradle.properties` y `android/build.gradle`.

Para comprender el funcionamiento del módulo, la comunicación comienza con las funciones exportadas del módulo en `index.tsx`. Allí se ejecutan funciones definidas en `NativeClientComscore.ts`, que contiene una interfaz donde se declaran las funciones

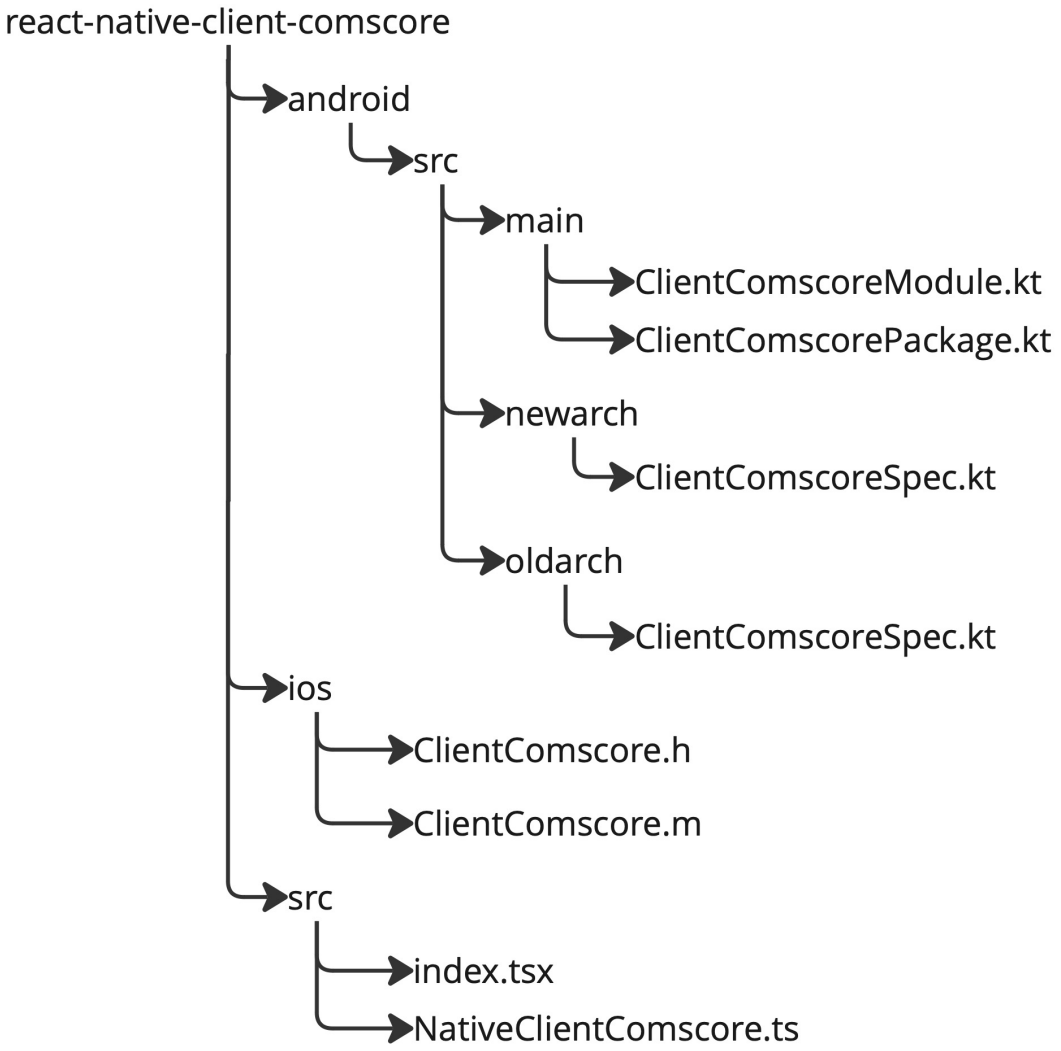


Figura 5.4: Estructura de ficheros relacionados al nuevo módulo.

que se implementarán en código nativo y se registra como un módulo. Los *Spec* en Android funcionan como interfaces para cada arquitectura: en la nueva arquitectura no es necesaria la definición de las funciones de `NativeClientComscore.ts`, mientras que en la antigua arquitectura sí es necesario definirlas y actualizarlas en consecuencia. Por otra parte, en iOS, el archivo `ClientComscore.h` se encarga de gestionar el manejo de ambas arquitecturas, sin necesidad de modificaciones en su definición.

En Android, el archivo `ClientComscoreModule.kt` implementa las funcionalidades del módulo nativo, utilizando el decorador "`@ReactMethod`" para indicar los métodos definidos en la interfaz, mientras que `ClientComscorePackage.kt` registra el módulo en la aplicación. Por otro lado, en iOS, `ClientComscore.m` se encarga de registrar e implementar las funcionalidades del módulo.

Implementación

Además de los archivos base, se crearon múltiples archivos auxiliares durante la implementación, incluyendo clases, enumeraciones, y tipos. Estos archivos adicionales con-

tribuyen a organizar mejor el código, facilitando una estructura más modular y ordenada, lo que resulta en un código más limpio, legible y fácil de mantener.

El desarrollo del módulo se benefició del soporte de los equipos de Android e iOS, quienes aportaron su experiencia y conocimientos en la implementación de funciones específicas en sus respectivas plataformas. Esto permitió minimizar la incertidumbre durante la integración de la biblioteca.

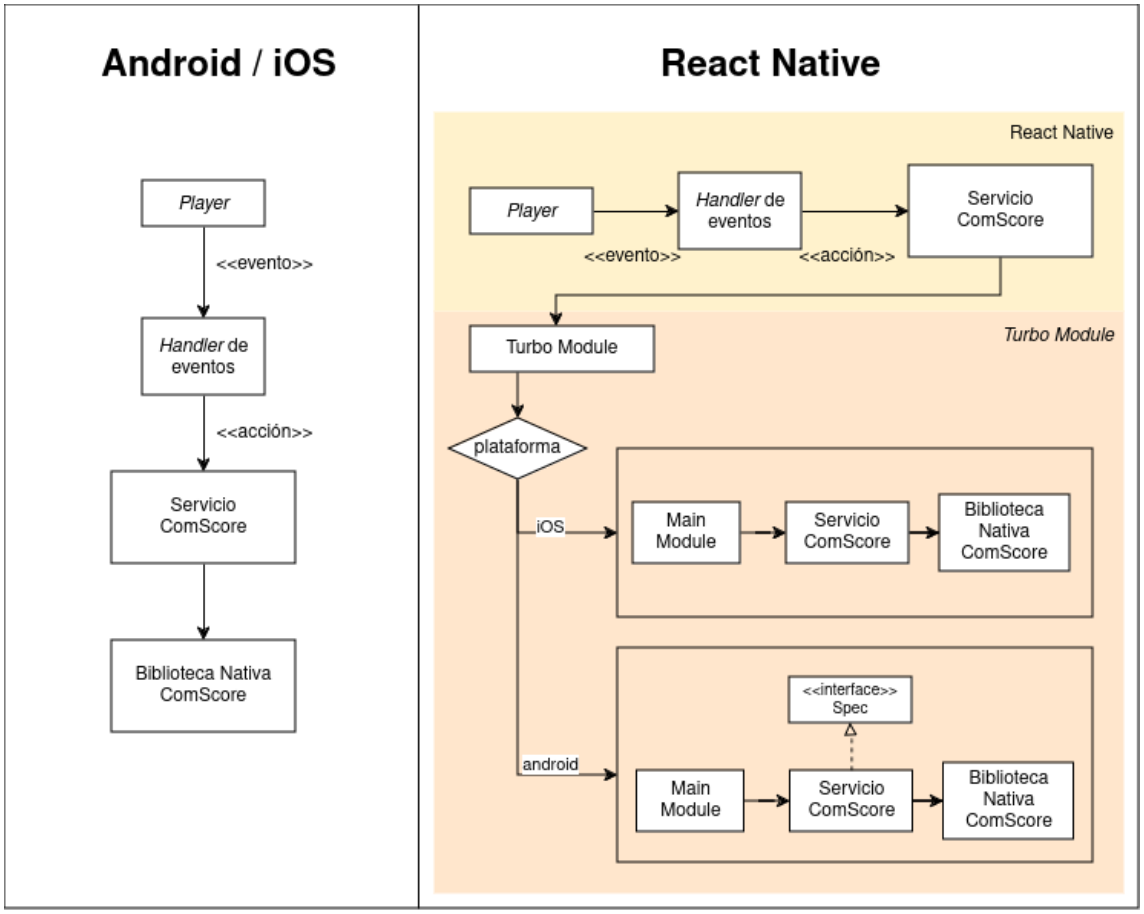


Figura 5.5: Comparación de implementaciones Nativas y usando *Turbo Modules*.

Finalmente, se logró una comunicación efectiva entre el módulo y el código de React Native. En la figura 5.5 se muestra el flujo de información en la implementación nativa (android/ios) y el nuevo flujo requerido para la implementación mediante *Turbo Module* en React Native con ligeros cambios. Al reutilizar satisfactoriamente las funciones definidas en el desarrollo nativo existente, se asegura un funcionamiento y uso correcto de la biblioteca y se obtiene un código de calidad y fácil mantenimiento por equipos de desarrollo nativo. A pesar de la complejidad esperada para este desarrollo resulto estar correctamente estimado y se cumplieron los tiempos con una variación no significativa.

5.1.4. Youbora

Solución de análisis de rendimiento y calidad para la transmisión de video en tiempo real. Proporciona datos sobre la experiencia del usuario, la calidad de transmisión y los problemas técnicos, ayudando a optimizar la entrega de contenido y mejorar la experiencia del espectador.

La biblioteca de Youbora ofrece soporte para React Native, permitiendo una integración directa. Se utilizan `npaw-plugin-react-native` y `npaw-plugin-adapters` (versión:

7.2.34). Sin embargo, al seguir los pasos descritos en la documentación, surgió una problemática relacionada con inconsistencias entre la documentación y la implementación de la biblioteca. Las funciones especificadas en la documentación como disponibles desde TypeScript no están accesibles, aunque se descubrió que están definidas en el código JavaScript. Al parecer, estas funciones no se exponen en el tipo del objeto, lo que puede deberse a una decisión del equipo de desarrollo de reemplazar funciones antiguas con nuevas versiones que no están reflejadas en la documentación, o a un error en la versión actual de la biblioteca.

Dado que la causa de esta discrepancia es incierta, se ha enviado una consulta al soporte técnico para resolver la inconsistencia. Hasta recibir una respuesta, no se puede proceder con el uso de la biblioteca. Si se confirma que las funcionalidades requeridas no están soportadas por la biblioteca, se considerará la implementación alternativa mediante API REST.

El inconveniente encontrado en esta tarea no tuvo un gran impacto en la planificación por la rápida indentificación, por lo que queda en espera de más información pero con tiempo de implementación restante que se considera suficiente para cumplir con lo planificado y no afectar a la planificación global.

5.1.5. *StatsAPI*

StatsAPI es una colección de API proporcionada por el cliente para la recopilación de información sobre el uso de la aplicación y la actividad del reproductor de medios. Los datos recopilados se utilizan posteriormente para generar informes y apoyar la toma de decisiones estratégicas dentro de la empresa.

Dado que *StatsAPI* es un servicio REST, su integración con la aplicación es directa y eficiente. Los datos se recopilan y se envían a la API correspondiente sin complicaciones. Después de realizar las pruebas necesarias, se confirma que la funcionalidad está correctamente integrada y operativa.

Esta tarea se finaliza en casi la mitad del tiempo estimado, beneficiando el proyecto al contrarestar los retrasos ocasionados por integraciones anteriores.

5.1.6. *SmartAd Server*

Plataforma de gestión de anuncios que permite la integración y entrega de anuncios en aplicaciones y sitios web. Utiliza el formato VAST (del inglés *Video Ad Serving Template*)[6] para la inserción y seguimiento de anuncios en el contenido de video.

SmartAd Server no ofrece soporte directo para React Native. Sin embargo, el reproductor de medios Bitmovin sí es compatible con el formato VAST para la reproducción de anuncios integrados en el reproductor. La configuración de los anuncios en Bitmovin se realiza correctamente, procesando la información recibida de la API de medios y registrando los anuncios de manera adecuada. Al igual que la tarea de *StatsAPI* se logra ganar tiempo al finalizar la tarea en menor tiempo del esperado, aportando positivamente a la planificación.

Para una integración completa en la estructura de la aplicación, se anticipa la necesidad de desarrollar un *Turbo Module*. Esto es particularmente relevante, ya que la misma biblioteca se utiliza para mostrar anuncios en pantalla completa y durante los segmentos informativos.

[6] https://www.iab.com/guidelines/vast/

5.1.7. Integraciones pendientes

Actualmente, las integraciones pendientes son las siguientes:

- ***Adobe Analytics***: La integración está en espera de respuestas del equipo técnico para resolver problemas pendientes.
- **Youbora**: La integración está pendiente de una resolución sobre la inconsistencia entre la biblioteca y la documentación proporcionada.

Además de las integraciones pendientes del *player*, la integración de las bibliotecas relacionadas con la estructura están en su mayoría sin abordar, excepto para **ComScore**. La inicialización de ComScore en la estructura fue necesaria para el seguimiento del reproductor de medios, por lo que se encuentra completada.

5.2 Pruebas

Se realizaron pruebas unitarias a cada servicio, al *handler* de eventos y al *hook* encargado de la lógica relacionada al *player*. Se espera como estándard de calidad una cobertura mayor del 90 % en declaraciones y 80 % en ramas, funciones y líneas (90 | 80 | 80 | 80). Los resultados de los distintos servicios desarrollados para cada integración se muestran en la tabla 5.2 siguiente.

Tabla 5.2: Resultados de tests unitarios

Servicio	**Declaraciones (90 %)**	**Ramas (80 %)**	**Funciones (80 %)**	**Líneas (80 %)**
Comscore	100	81.25	100	100
Adobe Analytics	100	92.1	100	100
StatsAPI	100	81.57	100	100
Youbora	91.66	100	84.61	91.66
SmartAd Server	100	100	100	100

***Events Handler*:** 97.05 | 83.33 | 100 | 97.05

***Player Hook*:** 94.87 | 85.76 | 89.34 | 92.45

Durante el proceso de pruebas, el equipo de QA llevaron a cabo pruebas de funcionalidad, rendimiento y estabilidad del prototipo desarrollado. Estas pruebas se se realizaron progresivamente, centradas en varios aspectos críticos, incluyendo la compatibilidad con las bibliotecas de terceros, la eficiencia en la carga de datos y la fluidez en la interfaz de usuario.

Los resultados obtenidos fueron en general satisfactorios. Las bibliotecas integradas, realizan su función correctamente. En cuanto a la interfaz de usuario del *player*, las pruebas revelaron que la experiencia de navegación es fluida y coherente, con tiempos de carga reducidos,transiciones suaves y manejo de errores.

No obstante, se identificaron algunas áreas que requieren ajustes, como la ejecución de múltiples tareas simultáneas donde se detecta que el sistema tiende a experimentar ligeras caídas de rendimiento, principalmente al cambiar de modos de visualización. Estos hallazgos subrayan la necesidad de optimizar ciertos procesos y de considerar posibles mejoras en la gestión de recursos para lograr una mejor experiencia de usuario.

En conclusión, los resultados de las pruebas confirman la viabilidad del proyecto, aunque también señalan áreas de mejora que serán abordadas en las siguientes fases de

desarrollo. Estos resultados y el cumplimiento de los parámetros de cobertura, proporcionan una base sólida para continuar con la implementación, garantizando que el producto final cumpla con las expectativas de calidad y rendimiento establecidas.

CAPÍTULO 6
Conclusiones y recomendaciones

En este capítulo se presentan las conclusiones finales del proyecto de migración de aplicaciones nativas a un entorno de desarrollo multiplataforma. A lo largo del proceso, se han evaluado diferentes alternativas tecnológicas, se ha implementado un prototipo y se han realizado pruebas para garantizar la viabilidad y efectividad de la migración. Este capítulo sintetiza los resultados obtenidos, discute las dificultades encontradas durante la ejecución del proyecto y propone recomendaciones para futuras implementaciones similares. Además, se reflexionará sobre el impacto del proyecto en el contexto empresarial del cliente y se evaluará la consecución de los objetivos planteados al inicio del trabajo.

6.1 Resultados

A la fecha de entrega de este documento, se ha logrado un prototipo con las siguientes funcionalidades de integración de bibliotecas:

Tabla 6.1: Comparación de métodos de integración para *Player* y Estructura.

Método de integración	**Player**	**Estructura**
Directa	2	1
Turbo Module	1	0
API REST	1	0
Bloqueada	2	0
No abordada	0	4

En cuanto a las bibliotecas relacionadas con el player, se han integrado con éxito las siguientes:

- **Bitmovin Player** y **SmartAd**: Se integraron de forma directa, sin necesidad de nuevos desarrollos.
- **ComScore**: Se integró utilizando un *Turbo Module*, el cual también incluyó la estructura necesaria.
- **StatsAPI**: Se integró mediante la API REST del cliente.

Las integraciones bloqueadas son:

- **Adobe Analytics**: Aún no se ha definido cómo proceder con su integración.

- **Youbora**: Se sabe que su integración será directa, pero está bloqueada debido a una consulta pendiente con el proveedor.

En cuanto a las bibliotecas relacionadas con la estructura, se integró ComScore como consecuencia de la integración del player. Las demás bibliotecas aún no se han abordado, pero se conoce lo siguiente:

- **Adobe Analytics**: Se podrá utilizar el SDK de Edge.
- **SmartAd**: No presenta SDK para React Native, por lo que es muy probable que sea necesario desarrollar un *Turbo Module*.
- **Azure Active Directory**: Presenta SDK, por lo que se espera una integración directa.
- **Didomi**: Presenta SDK, por lo que se espera una integración directa.

Como resultados relevantes destacar el éxito la implementación de ComScore mediante *Turbo Modules*, que presentaba un reto técnico para el equipo. Desde el inicio, los desarrolladores trabajaron en estrecha coordinación siguiendo la documentación de React Native, destacándose una fluida comunicación y colaboración entre los equipos involucrados. Gracias a un enfoque proactivo, se lograron resolver rápidamente los problemas surgidos, garantizando que esta funcionalidad de medición y análisis de audiencias se integrara perfectamente en la nueva plataforma multiplataforma. Este logro no solo refleja la capacidad técnica, sino también la eficacia de la comunicación y el trabajo en equipo.

6.1.1. Dificultades encontradas

Durante el desarrollo varios aspectos amenazaron la planificación del proyecto, la más destacada en la integración del reproductor de video, dado que la documentación de las bibliotecas en varias ocasiones se encontraba desactualizada por lo que se tuvo que recurrir al soporte técnico en 4 ocasiones distintas bloqueando tareas.

Desde el punto de vista estadístico de la aplicación de JIRA podemos obtener que de las 14 tareas relacionadas con integraciones de bibliotecas externas se han bloqueado un 50 % mientras que en las 109 tareas restantes del proyecto solamente el 8.26 % ha estado bloqueada. Esto evidencia que este aspecto fue de riesgo para el proyecto, siendo la integración de *Adobe* lo que ha generado un impacto en la planificación. Sin embargo se comprobó que las primeras integraciones realizadas fueron las de mayor riesgo para el proyecto quedando al final las que se esperaba fueran tareas sencillas (sin mucho riesgo).

En el proceso de desarrollo las principales dificultades se encontraron al integrar las funcionalidades de Adobe Analytics, que influyó en la planificación del proyecto y aún se encuentra bloqueado, a falta de documentación para el correcto desarrollo. Esta tarea se estima que continúe siendo complicada, no solo por el desarrollo sino también por la comunicación que será necesaria con otro equipo fuera del proyecto.

6.2 Conclusiones

A lo largo del desarrollo del proyecto, se lograron cumplir los objetivos planteados inicialmente en la introducción. El objetivo principal, que consistía en analizar y evaluar la viabilidad de migrar aplicaciones nativas de Android e iOS a un entorno multiplataforma, se alcanzó mediante el desarrollo de un prototipo funcional a pesar de la

incertidumbre existente. Este prototipo integró varias funcionalidades, garantizando la consistencia y compatibilidad entre el desarrollo multiplataforma y el desarrollo web. Además, se consideraron factores clave como la disponibilidad de bibliotecas, el rendimiento, la compatibilidad y la escalabilidad, cumpliendo con las expectativas del cliente y proporcionando una base sólida para futuras decisiones estratégicas.

En el contexto empresarial del cliente aún no el perceptible un impacto, sin embargo, se encuentra satisfecho con los resultados actuales y las metodologías empleadas. Expresan un gran interés en esta etapa de desarrollo multiplataforma y valoran positivamente el prototipo obtenido coherente con los desarrollos existentes.

Además, a nivel de recursos humanos, Alten se logra avanzar en la curva de aprendizaje relacionada con React Native y las integraciones mediante *Turbo Modules*, lo que permitirá que futuros desarrollos sean más ágiles y eficientes. De este modo, no solo se cumplen los objetivos técnicos, sino que también sienta bases para futuras migraciones exitosas hacia entornos de desarrollo multiplataforma en la organización.

La integración de funcionalidades en *frameworks* multiplataforma, pueden llevar desarrollos simples o complejos, esto va a depender del soporte que el creador brinde. Por lo que, cuando se analiza la viabiliad se requiere de la realización de un estudio cuidadoso de cada aspecto a integrar. Una base sólida permite estimar de mejor manera el coste y alcance, ya que como se ha comprobado existen diversas formas de solventarlo; sin embargo, la selección depedenderá de la particularidad de cada proyecto.

6.2.1. Recomendaciones

A modo de recomendación se puede resaltar la necesidad de una estrategia de gestión de riesgos sistemática en proyectos de desarrollo de *software* y su implementación.

Dado que la metodología comúnmente utilizada en los proyectos desarrollados en la empresa es Scrum, se dispone de un equipo con roles claramente definidos, incluyendo el jefe de proyecto, jefes de equipo e integrantes de los equipos técnicos. Adicionalmente, el proyecto se beneficia del apoyo de un líder técnico con experiencia en las áreas pertinentes, como desarrollo web, multiplataforma, Android o iOS.

Para asegurar una gestión de riesgos efectiva, es esencial que al menos un miembro del equipo tenga conocimientos especializados en la gestión de riesgos. Esta persona será responsable de implementar y supervisar las estrategias de gestión de riesgos a lo largo del proyecto. Dado que la gestión de riesgos debe estar coordinada con la planificación del proyecto, se recomienda que, en caso de no asignar un miembro nuevo con el rol específico para esta tarea, el jefe de proyecto asuma la responsabilidad de la gestión de riesgos; en coherencia con sus responsabilidades de asignación de recursos y planificación. Así mismo, los líderes técnicos y el cliente formarían parte de la etapa de identificación y planificación de riesgos, ya que sus conocimientos aportan valor al análisis.

El Apéndice B proporciona un ejemplo sencillo de los contenidos de un Plan de Gestión de Riesgos, que puede servir como referencia para establecer una estructura de gestión de riesgos en futuros proyectos. Además, para el manejo de esta información se recomienda el uso de herramientas como Microsoft Excel y *RiskWatch*[1], que, con la correcta configuración, permiten visualizar los datos y obtener rápidamente el estado del riesgo del proyecto.

Por otra parte luego de investigar el método Spotify recomendaría su estudio a fondo y valoración de utilización, ya que esta ganando popularidad y su estructura puede adaptarse a las necesidades y objetivos de la empresa.

[1] https://www.riskwatch.com

Para mejorar el proceso de integración de Adobe Analytics se recomienda la creación de un plan de comunicación entre el equipo de React Native y el encargado de la gestión de Adobe, ya que la complejidad del mismo requerirá el intercambio de conocimientos y consultas periódicas.

6.3 Trabajo futuro

A pesar de los avances significativos logrados en el desarrollo e implementación del prototipo, existen varias áreas que aún requieren atención y que se abordarán en las próximas iteraciones del proyecto. Estos aspectos son cruciales para asegurar que el producto final cumpla con los estándares de calidad y rendimiento esperados, y que se adapte plenamente a las necesidades del cliente.

Las pruebas iniciales han revelado áreas donde el sistema experimenta ligeras caídas de rendimiento bajo condiciones de alta demanda. En futuras iteraciones, se implementarán estrategias de optimización para mejorar la gestión de recursos y la eficiencia del sistema. Esto incluirá una revisión detallada de los procesos más exigentes y la implementación de mejoras en la arquitectura del software para asegurar una operación óptima en todos los escenarios.

El prototipo actual cubre las funcionalidades básicas y algunas críticas del sistema, pero aún quedan pendientes varias integraciones, especialmente aquellas relacionadas con servicios de terceros que no fueron abordadas en esta fase y aquellas que han quedado bloqueadas por diversos motivos. Estas integraciones serán prioritarias en las siguientes iteraciones para garantizar que el sistema ofrezca un conjunto completo de funcionalidades y sea coherente con el entorno que utilizan actualmente los usuarios. La implementación de estas, como hasta ahora, es parte de mis tareas en el equipo, por lo que será la continuación de lo expresado en este trabajo y seguimiento de las tareas bloqueadas.

Aunque la interfaz de usuario ha mostrado ser fluida y coherente, se llevará a cabo un refinamiento adicional para mejorar la experiencia del usuario ya que el diseño aún no está terminado y pasa por una etapa de revisión. Concluido esto el maquetado que se ha realizado hasta ahora es altamente propable que sufra cambios.

Hasta ahora, las pruebas se han centrado en la funcionalidad y el rendimiento, pero en futuras iteraciones se realizarán pruebas exhaustivas de escalabilidad y seguridad. Estas pruebas son esenciales para garantizar que el sistema pueda manejar un crecimiento en la base de usuarios y que esté protegido contra posibles vulnerabilidades y amenazas de seguridad.

Una vez finalizado, se desarrollará una documentación completa del sistema, incluyendo guías de usuario, manuales técnicos y procesos de despliegue.

En resumen, aunque se ha avanzado significativamente, el proyecto continuará evolucionando en futuras iteraciones para completar las funcionalidades restantes, optimizar el sistema y asegurar que el producto final cumpla con las expectativas tanto del cliente como de los usuarios finales.

Bibliografía

[1] D. Naranjo, D. Buenaño, & I. T. Mejía. Evolución de la tecnología móvil. Camino a 5G. *Revista Contribuciones a las Ciencias Sociales*, 2016, pp. 1-13.

[2] Rieger, C., & Majchrzak, T. A. Towards the definitive evaluation framework for *cross-platform* app development approaches. *Journal of Systems and Software (JSS)*, 153, 2019, pp. 175–199. doi:10.1016/j.jss.2019.04.001.

[3] Project Management Institute. Guía de los Fundamentos para la Dirección de Proyectos (Guía del PMBOK®), 7ª ed. *Project Management Institute*, 2021. ISBN: 978-1-62825-664-2.

[4] Mascarell Palau, D. Del teléfono móvil al smartphone: Un recorrido evolutivo del dispositivo móvil hacia implicaciones educativas y artísticas con la imagen. H-ART. Revista De Historia, Teoría Y Crítica De Arte, 12:258: , 2022. doi:10.25025/hart12.2022.10.

[5] Delía, L. N. Desarrollo de aplicaciones móviles multiplataforma. *Doctoral dissertation*, Universidad Nacional de La Plata, Argentina, 2017.

[6] International Telecommunication Union (ITU). *World Telecommunication Development Report: Mobile cellular*. 5th edition, 1999. Enlace persistente: http://handle.itu.int/11.1002/pub/800c8182-en.

[7] El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., & Wahba, A. M. Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Engineering Journal*, 8(2), pp. 163-190, 2017.

[8] Flanagan, D. *JavaScript: The Definitive Guide*. O'Reilly Media, 2018.

[9] Heitkötter, H., & Majchrzak, T. A. Cross-platform development of business apps with MD2. *Design science at the intersection of physical and virtual design*, pp. 405-411. Lecture notes in computer science. Springer, Berlín, 2013.

[10] Heitkötter, H., Hanschke, S., & Majchrzak, T. A. Evaluating cross-platform development approaches for mobile applications. *International Conference on Web Information Systems and Technologies*, pp. 120-138. Springer, Berlin, Heidelberg. April, 2012.

[11] Biørn-Hansen, A., Rieger, C., Grønli, T. M., Majchrzak, T. A., & Ghinea, G. An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering*, 25(4), pp. 2997-3040. 2020.

[12] Biørn-Hansen, A., Grønli, T. M., & Ghinea, G. A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *ACM Computing Surveys (CSUR)*, 51(5), pp. 108:1–108:34. 2018.

[13] Duckett, J. *HTML and CSS: Design and Build Websites*. John Wiley & Sons, 2018.

[14] Barnett, A. *Experiments Comparing Cross-Platform and Native Mobile Applications – A Systematic Literature Review*. *Master's Thesis*, Tampere University, Master's Degree in Computing Sciences, April 2023.

[15] Zhou, Changkong. Challenges and solutions in cross-platform mobile development: a qualitative study of Flutter and React Native. *Master's Thesis*, Master's Programme in Computer, Communication and Information Sciences, Aalto University, 2024.

[16] Hillson, D., & Simon, P. Practical Project Risk Management: The ATOM Methodology, 3rd Edition. *Management Concepts*, 2020.

[17] Yapiko.com. Tipos de aplicaciones móviles: nativas, híbridas y web apps. Marzo 23, 2022. Consultado el 25 de abril de 2024 en: https://yapiko.com/es/blog/aplicaciones-moviles-nativas-hibridas-y-web-apps/.

[18] Universidad Europea. Qué son las apps nativas. Consultado el 25 de abril de 2024 en: https://universidadeuropea.com/blog/apps-nativas/.

[19] Roberto Adeva. Qué es Android: todo sobre el sistema operativo de Google. Actualizado el 2 de febrero de 2023, 10:51. Consultado el 8 de mayo de 2024 en: https://www.adslzone.net/reportajes/software/que-es-android/.

[20] Santiago Jiménez. Guía completa para crear aplicaciones Android con Java. Publicado el 20 de agosto de 2023. Consultado el 8 de mayo de 2024 en: https://aprenderjava.net/base/guia-completa-para-crear-aplicaciones-android-con-java/#Requisitos_para_desarrollar_aplicaciones_Android.

[21] Jessica Thornsby. Java vs. Kotlin: ¿Deberías Usar Kotlin en Desarrollo Android? Publicado el 12 de diciembre de 2016. Consultado el 8 de mayo de 2024 en: https://code.tutsplus.com/es/java-vs-kotlin-should-you-be-using-kotlin-for-android-development--cms-27846a.

[22] Rocío GR. ¿Qué es iOS? Todo sobre el sistema operativo de Apple. Actualizado el 15 de marzo de 2023, 17:01. Consultado el 10 de mayo de 2024 en: https://www.adslzone.net/reportajes/software/que-es-ios/.

[23] Conde. Desarrollo iOS – Qué es, Características y su Desarrollo. Publicado el 29 de junio de 2023. Consultado el 10 de mayo de 2024 en: https://aprendeinformaticas.com/desarrollo-ios/.

[24] Juan F. Samaniego. Adiós a una década revolucionaria: ¿cómo ha cambiado la tecnología móvil en los últimos 10 años? Publicado el 23 de marzo de 2021. Consultado el 15 de mayo de 2024 en: https://blog.orange.es/innovacion/tecnologia-movil-de-la-decada/.

[25] IBM. ¿Qué es la tecnología móvil? Consultado el 15 de mayo de 2024 en: https://www.ibm.com/mx-es/topics/mobile-technology.

[26] Coppola, Maria. Desarrollo web: qué es, etapas y principales lenguajes. Publicado el 2 de mayo de 2023. Actualizado el 4 de mayo de 2023. Consultado el 16 de mayo de 2024 en: https://blog.hubspot.es/website/que-es-desarrollo-web.

[27] Mozilla Developer Network (MDN). *Web technology for developers*. Consultado el 16 de mayo de 2024 en: https://developer.mozilla.org/en-US/.

[28] Apple Coding academy. Conoce las diferencias entre desarrollo híbrido, multiplataforma y nativo. Consultado el 22 de mayo de 2024 en: https://acoding.academy/conoce-diferencias-desarrollo-hibrido-multiplataforma-nativo/.

[29] Osmani, Addy. Extraído de: *Comienza a usar las apps web progresivas*. Consultado el 22 de mayo de 2024 en: https://developer.chrome.com/blog/getting-started-pwa.

[30] Imagine Apps. Nativo vs Híbrido vs Multiplataforma: Cómo y Qué Elegir. Publicado el 15 de septiembre de 2023. Consultado el 2 de abril de 2024 en: https://www.imagineapps.co/blog-posts-es/nativo-vs-hibrido-vs-multiplataforma-como-y-que-elegir.

[31] Tigren. Top 7 Progressive Web App (PWA) frameworks for Developers. Última actualización el 30 de enero de 2024. Consultado el 2 de abril de 2024 en: https://www.tigren.com/blog/pwa-framework/.

[32] Mónica Mena Roa. Android e iOS dominan el mercado de los smartphones. 30 de agosto de 2021. Consultado el 7 de abril de 2024 en: https://es.statista.com/grafico/18920/cuota-de-mercado-mundial-de-smartphones-por-sistema-operativo/.

[33] React Native. Mayo 23, 2024. Consultado el 23 de abril de 2024 en: https://reactnative.dev/.

[34] May Sanders. How to Build a High-Quality React Native App. 9 de agosto de 2023. Consultado el 23 de abril de 2024 en: https://www.ejemplo.com/articulo-react-native.

[35] Back4App. ¿Qué es React Native Framework? Consultado el 23 de abril de 2024 en: https://blog.back4app.com/es/que-es-react-native/.

[36] lmartinez. React Native: Ventajas y desventajas de este framework. 25 de agosto de 2021. Consultado el 24 de abril de 2024 en: https://cms.rootstack.com/es/es/blog/react-native-ventajas-y-desventajas-de-este-framework.

[37] React Native new Architecture. Mayo 28, 2024. Consultado el 24 de abril de 2024 en: https://reactnative.dev/docs/the-new-architecture/landing-page.

[38] Jose Luis Bustos. ¿Cómo convertirte en React Native Developer? Última modificación: 24 de abril de 2024. Consultado el 24 de abril de 2024 en: https://keepcoding.io/blog/convertirte-en-react-native-developer.

[39] Conceptos de la Historia. La economía detrás de la televisión: de la publicidad a las producciones originales. Consultado el 17 de junio de 2024 en: https://conceptosdelahistoria.com/conceptos-politicos/capitalismo/economia-de-la-television/.

[40] Norma Internacional de Contabilidad nº 38: Activos intangibles Fecha: diciembre 2023 Consultado el 17 de junio de 2024 en: https://www.icac.gob.es/node/761

[41] Amortización de inmovilizado. Sistemas y programas informáticos Consultado el 2 de julio de 2024 en: https://www.supercontable.com/informacion/impuesto_sociedades/Amortizacion_de_inmovilizado.Sistemas_y_programas_.html.

[42] Conoce las ventajas y desventajas del desarrollo de aplicaciones con Flutter. GRUPO EBIM. Última modificación: 28 de junio de 2023. Consultado el 2 de julio de 2024 en: https://www.grupoebim.com/blog/ventajas-desventajas-flutter/.

[43] Ken Schwaber and Jeff Sutherland. The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game. Consultado el 23 de julio de 2024 en: https://scrumguides.org/.

[44] Jain, Pramesh. A Detailed Guide On New React Native Architecture. Publicado el 17 de enero de 2023. Consultado el 14 de junio de 2024 en: https://www.pontikis.net/blog/a-detailed-guide-on-new-react-native-architecture.

[45] Cruth, Mark. Descubre el modelo de Spotify. Atlassian, consultado el 10 de agosto de 2024 en: https://www.atlassian.com/es/agile/agile-at-scale/spotify.

APÉNDICE A

Diagrama de Gantt

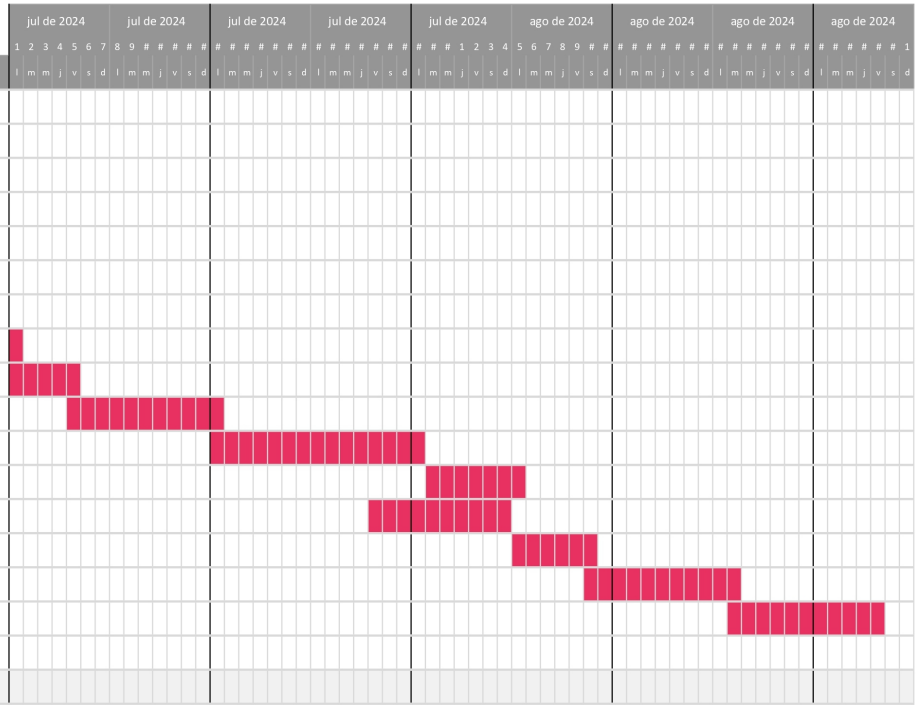
Migración a React Native

DIAGRAMA DE GANTT

Inicio del proyecto: lu, 06/05/2024

Semana para mostrar: 9

TAREA	ASIGNADO A	PROGRESO	INICIO	FIN
Player				
Bitmovin	Claudia	0 %	6-5-24	26-5-24
Lectura API media videos	Claudia	0 %	27-5-24	30-5-24
Lectura API media audios	Claudia	0%	30-5-24	2-6-24
Subtítulos	Claudia	0%	3-6-24	9-6-24
Adobe Información	Claudia	0%	10-6-24	18-6-24
Adobe SDK	Claudia	0%	11-6-24	24-6-24
ComScore Turbo Module	Claudia	0%	24-6-24	1-7-24
ComScore Turbo Module - iOS	Desarrollador iOS	0%	28-6-24	5-7-24
StatsAPI	Desarrollador RN	0%	5-7-24	15-7-24
Youbora	Claudia	0%	15-7-24	29-7-24
Publicidad Video	Claudia	0%	30-7-24	5-8-24
Player audio UI	Desarrollador RN	0%	26-7-24	4-8-24
Publicidad Audio	Claudia	0%	5-8-24	10-8-24
Publicidad audio UI	Claudia	0%	10-8-24	20-8-24
Manejo de errores	Claudia	0%	20-8-24	30-8-24
Inserte nuevas filas ENCIMA de ésta				



APÉNDICE B
Plan de Riesgos

Este apéndice pretende servir de asistencia para los elementos necesarios en una planificación de riesgos. El formato de presentación de la información puede variar, en usos de tablas o plantillas definidas por la empresa.

B.1 Identificación de Riesgos

B.1.1. Riesgos Técnicos

RIESGO: Incompatibilidad entre plataformas:

Impacto: Alto
Probabilidad: Alta
Descripción: El *software* puede no funcionar uniformemente en plataformas como iOS, Android y Windows.
Consecuencia: Retrasos en el proyecto debido a ajustes y pruebas adicionales.
Área Afectada: Desarrollo y pruebas.
Responsable: Equipo de Desarrollo Multiplataforma

RIESGO: Fallos en las integraciones de terceros:

Impacto: Alto
Probabilidad: Media
Descripción: Servicios de terceros (API, sistemas externos) pueden tener fallos o cambios inesperados.
Consecuencia: Interrupciones en las funcionalidades del *software* y posibles retrasos.
Área Afectada: Integraciones y funcionalidades del sistema.
Responsable: Equipo de Integraciones y soporte de API

B.1.2. Riesgos de Diseño

RIESGO: Cambios en los requisitos de la interfaz de usuario:

Impacto: Medio
Probabilidad: Alta
Descripción: Cambios en el diseño de la interfaz pueden afectar la usabilidad y estética.
Consecuencia: Ajustes en el desarrollo y potenciales retrasos en la entrega.
Área Afectada: Diseño y desarrollo de la interfaz de usuario.
Responsable: Equipo de Diseño y Gestión de Cambios

RIESGO: Dificultades en la implementación del diseño:

Impacto: Medio
Probabilidad: Media
Descripción: Problemas en la implementación del diseño pueden resultar en inconsistencias.
Consecuencia: Rediseño o ajustes que pueden afectar la calidad y el tiempo de entrega.
Área Afectada: Desarrollo de la interfaz y calidad del producto.
Responsable: Equipo de Desarrollo Frontend

B.1.3. Riesgos del Proyecto

RIESGO: Retrasos en la entrega de requisitos:

Impacto: Alto
Probabilidad: Media
Descripción: Retrasos en la entrega de requisitos o especificaciones pueden afectar el cronograma.
Consecuencia: Ajuste en el cronograma y posible sobrecarga de trabajo.
Área Afectada: Planificación y cronograma del proyecto.
Responsable: Gerente de Proyecto

RIESGO: Escalabilidad del equipo:

Impacto: Medio
Probabilidad: Baja
Descripción: Dificultades para ajustar el tamaño del equipo según las necesidades del proyecto.
Consecuencia: Retrasos en las tareas y posible aumento en el coste del proyecto.
Área Afectada: Recursos humanos y gestión del equipo.
Responsable: Recursos Humanos y Gerente de Proyecto

B.2 Evaluación de Riesgos

Tabla B.1: Matriz de Riesgos

Riesgo	Probabilidad	Impacto	Nivel de Riesgo
Incompatibilidad entre plataformas	Alta	Alto	Alto
Fallos en las integraciones de terceros	Media	Alto	Medio
Cambios en los requisitos de la interfaz	Alta	Medio	Alto
Dificultades en la implementación del diseño	Media	Medio	Medio
Retrasos en la entrega de requisitos	Media	Alto	Alto
Escalabilidad del equipo	Baja	Medio	Bajo

B.3 Plan de Mitigación de Riesgos

B.3.1. Riesgos Técnicos

RIESGO: Incompatibilidad entre plataformas:

Estrategia de Mitigación: Realizar pruebas continuas en todas las plataformas desde las primeras etapas del desarrollo. Utilizar herramientas y frameworks multiplataforma.
Frecuencia de Seguimiento: Semanal
Responsable: Equipo de Desarrollo Multiplataforma

RIESGO: Fallos en las integraciones de terceros:

Estrategia de Mitigación: Implementar pruebas automatizadas y de integración continua. Establecer acuerdos de nivel de servicio (SLAs) con proveedores externos.
Frecuencia de Seguimiento: Mensual
Responsable: Equipo de Integraciones y soporte de API

B.3.2. Riesgos de Diseño

RIESGO: Cambios en los requisitos de la interfaz de usuario:

Estrategia de Mitigación: Implementar un proceso claro de gestión de cambios y mantener comunicación constante con diseñadores. Establecer revisiones periódicas.
Frecuencia de Seguimiento: Quincenal
Responsable: Equipo de Diseño y Gestión de Cambios

RIESGO: Dificultades en la implementación del diseño:

Estrategia de Mitigación: Incluir prototipos y pruebas de usabilidad durante el desarrollo para identificar problemas tempranamente.
Frecuencia de Seguimiento: Quincenal
Responsable: Equipo de Desarrollo Frontend

B.3.3. Riesgos del Proyecto

RIESGO: Retrasos en la entrega de requisitos:

Estrategia de Mitigación: Establecer un cronograma detallado con hitos claros y mantener comunicación regular con todas las partes interesadas.
Frecuencia de Seguimiento: Semanal
Responsable: Gerente de Proyecto

RIESGO: Escalabilidad del equipo:

Estrategia de Mitigación: Planificar con anticipación la necesidad de recursos y mantener una lista de candidatos potenciales para ampliar el equipo rápidamente si es necesario.
Frecuencia de Seguimiento: Mensual
Responsable: Recursos Humanos y Gerente de Proyecto

B.4 Plan de Contingencia

B.4.1. Riesgos Técnicos

RIESGO: Incompatibilidad entre plataformas:

Estrategia de Mitigación: Considerar el ajuste del alcance o la adopción de tecnologías alternativas si surgen problemas críticos.
Acción: Transferir / Aceptar
Responsable: Equipo de Desarrollo Multiplataforma

RIESGO: Fallos en las integraciones de terceros:

Estrategia de Mitigación: Implementar soluciones alternativas o de respaldo en caso de fallos prolongados.
Acción: Transferir
Responsable: Equipo de Integraciones y soporte de API

B.4.2. Riesgos de Diseño

RIESGO: Cambios en los requisitos de la interfaz de usuario:

Estrategia de Mitigación: Priorizar los cambios en función del impacto y ajustar el cronograma y recursos.
Acción: Mitigar
Responsable: Equipo de Diseño y Gestión de Cambios

RIESGO: Dificultades en la implementación del diseño:

Estrategia de Mitigación: Revisar y ajustar el diseño basado en los resultados de pruebas y buscar apoyo externo si es necesario.
Acción: Mitigar
Responsable: Equipo de Desarrollo Frontend

B.4.3. Riesgos del Proyecto

RIESGO: Retrasos en la entrega de requisitos:

Estrategia de Mitigación: Ajustar plazos y alcance del proyecto. Reprogramar hitos si es necesario.
Acción: Mitigar
Responsable: Gerente de Proyecto

RIESGO: Escalabilidad del equipo:

Estrategia de Mitigación: Contratar personal temporal o subcontratar tareas específicas para manejar fluctuaciones en la carga de trabajo.
Acción: Mitigar
Responsable: Recursos Humanos y Gerente de Proyecto

B.5 Revisión y Seguimiento de Riesgos

Frecuencia de Revisión: Revisión del Plan de Riesgos Quincenal y Revisión del Estado de Riesgos durante reuniones quincenales del equipo de proyecto.

Responsables de la Gestión de Riesgos:

- Responsable Principal: [Nombre del Responsable].
- Equipo de Riesgos: [Nombres de los miembros del equipo responsables de la gestión de riesgos].

Informes y Actualizaciones: Mantener un informe actualizado y distribuirlo a todas las partes interesadas del proyecto.

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**			X	
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**			X	
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En el mundo de la tecnología, el tema abordado en este TFM, la migración de desarrollo nativo a multiplataforma, se está convirtiendo en una tendencia impulsada por la necesidad de crear aplicaciones más eficientes, accesibles y sostenibles. Este cambio está estrechamente vinculado con varios Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas. El ODS 9 (Industria, Innovación e Infraestructura) es el que más se alinea con este enfoque, además del ODS 8 (Trabajo Decente y Crecimiento Económico) y el ODS 10 (Reducción de las Desigualdades) en menor medida.

ODS 9: Industria, Innovación e Infraestructura

El ODS 9 se centra en la construcción de infraestructuras resilientes, la promoción de la industrialización inclusiva y sostenible, y el fomento de la innovación. La migración a tecnologías multiplataforma está muy alineada con este objetivo. El desarrollo multiplataforma permite la reutilización de código, reduciendo significativamente los costos y el tiempo de desarrollo. Esta eficiencia en el proceso de desarrollo no solo acelera el tiempo de llegada al mercado de nuevos productos, sino que también reduce el impacto ambiental asociado con el desarrollo de software y contribuye a una infraestructura tecnológica más integrada y sostenible. Además, la adopción de tecnologías multiplataforma impulsa la innovación, al ofrecer a los desarrolladores flexibilidad para crear aplicaciones aprovechando las capacidades de diferentes dispositivos. Esto da paso a creación de nuevas soluciones tecnológicas que pueden influir en la vida de las personas. Así, la migración a plataformas multiplataforma no solo mejora la infraestructura tecnológica, sino que también impulsa el avance tecnológico y la innovación en el sector.

ODS 8: Trabajo Decente y Crecimiento Económico

Este ODS se enfoca en el crecimiento económico inclusivo y el trabajo decente. La migración a desarrollos multiplataforma puede tener un impacto significativo en este ámbito. Al permitir una mayor eficiencia en el desarrollo y mantenimiento de aplicaciones, las empresas pueden reducir costos y redistribuir recursos de manera más efectiva. Esta reducción de costos puede traducirse en precios más bajos para los consumidores y en la capacidad de las empresas para invertir en otras áreas de crecimiento e innovación.

Además, el desarrollo de tecnologías multiplataforma puede generar nuevas oportunidades de empleo en el sector tecnológico. A medida que más empresas adoptan estas tecnologías, se necesita una fuerza laboral con habilidades específicas en desarrollo multiplataforma, lo que puede abrir nuevas oportunidades para los profesionales en tecnología. Sin embargo, aunque esta tecnología puede promover el crecimiento económico y la creación de empleo, el impacto en la mejora del trabajo decente y la inclusión laboral puede ser más indirecto. La relación no es tan directa como con el ODS 9, ya que el impacto puede variar según la implementación y el contexto.

ODS 10: Reducción de las Desigualdades

El ODS 10 busca reducir las desigualdades en y entre los países. La tecnología multiplataforma puede contribuir a este objetivo al facilitar el acceso a aplicaciones y servicios digitales desde una variedad de dispositivos y sistemas operativos. Se puede llegar a una audiencia más amplia, incluyendo personas que no tienen acceso a dispositivos de gama alta o sistemas operativos específicos. Sin embargo, el impacto en la reducción de desigualdades tecnológicas puede ser limitado. Aunque facilita el acceso a herramientas y servicios digitales, no aborda directamente otras formas de desigualdad, como la falta de infraestructura básica en ciertas regiones o las barreras económicas y sanciones que impiden a algunos paises puedan acceder a la tecnología.

La contribución a la reducción de desigualdades es real, pero no tan significativa como en el caso del ODS 9.

En conclusión, la migración de desarrollo nativo a multiplataforma tiene una relación especialmente fuerte con el ODS 9, al promover la innovación y mejorar la infraestructura tecnológica de manera sostenible. También tiene un impacto relevante en el ODS 8, al fomentar la eficiencia económica y la creación de empleo, aunque el efecto en la calidad del trabajo puede ser más indirecto. Finalmente, aunque la tecnología multiplataforma puede ayudar a reducir algunas desigualdades tecnológicas, el impacto en el ODS 10 es más limitado comparado con los otros objetivos.

En resumen, la transición hacia tecnologías multiplataforma no solo es una tendencia técnica, sino una estrategia que puede tener un impacto positivo significativo en varios ODS, especialmente en términos de innovación y eficiencia. Es importante seguir explorando y apoyando estas tecnologías para maximizar sus beneficios y contribuir a un desarrollo más sostenible.

ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es