



TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE



Desarrollo de una Aplicación Multiplataforma Escalable y Plataforma Analítica Complementaria para la Recopilación y Análisis de Datos de Uso

Estudiante: Francisco Lareo García

Dirección: Outro Nome Completo

A Coruña, setembro de 2024.

Dedicatoria

Agradecimentos

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Resumo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Palabras clave:

- First itemtext
- Second itemtext
- Last itemtext
- First itemtext
- Second itemtext
- Last itemtext
- First itemtext

Keywords:

- First itemtext
- Second itemtext
- Last itemtext
- First itemtext
- Second itemtext
- Last itemtext
- First itemtext

Índice Xeral

1	Introducción	1
1.1	Contexto y motivación	1
1.2	Objetivo del trabajo	2
1.2.1	Desarrollo de una aplicación OTT multiplataforma y multicliente . . .	2
1.2.2	Desarrollo e integración de una aplicación para la recogida y visualización de datos de uso	2
2	Fundamentos	4
2.1	Introducción	4
2.2	Fundamentos teóricos	4
2.2.1	OTT (Over-the-Top)	4
2.2.2	Arquitectura	6
2.2.3	Experiencia de usuario (UX)	8
2.2.4	Escalabilidad, adaptabilidad y multiplataforma	10
2.2.5	Importancia de la Analítica de Datos en Plataformas OTT	11
2.2.6	Aplicación de los Principios de UX en la Analítica de Datos	12
2.3	Fundamentos tecnológicos	12
2.3.1	Arquitectura Software: Comunicación entre componentes y desarrollo multicliente	12
2.3.2	Tecnologías Web: JavaScript, HTML y CSS	13
2.3.3	Desarrollo multiplataforma	14
2.3.4	Rendimiento y Optimización	16
2.3.5	Gestión de Versiones (Git)	17
2.3.6	Uso de APIs	17
3	Metodología	18
3.1	Introducción	18
3.1.1	Diseño del proyecto	18

3.2	Metodología de Desarrollo	19
3.2.1	Implementación progresiva e incremental	19
3.2.2	Adaptación continua y reuniones de validación	19
3.2.3	Ventajas de la metodología ágil	20
3.3	Descripción de las Etapas de Desarrollo	20
3.3.1	Fase de Análisis y Planificación	20
3.3.2	Fase de Diseño	20
3.3.3	Fase de Desarrollo	21
3.3.4	Fase de Pruebas	21
3.3.5	Conclusión	22
4	Análisis	23
4.1	Introducción	23
4.2	Requisitos	23
4.2.1	Requisitos de la plataforma OTT	24
4.2.2	Requisitos de la aplicación de análisis de datos	25
4.3	Necesidades del cliente	26
4.4	Estudio de viabilidad	27
4.4.1	Viabilidad técnica	27
4.4.2	Viabilidad económica	29
4.4.3	Conclusión final	29
4.5	Estudio de las características de los distintos sistemas operativos, dispositivos y tecnologías	29
4.5.1	Sistemas operativos	30
4.5.2	Dispositivos	31
4.5.3	Tecnologías	31
5	Diseño	32
5.1	Introducción	32
5.2	Diseño de la aplicación OTT	32
5.2.1	Introducción al diseño de la aplicación <i>ott</i>	32
5.2.2	Diseño de la arquitectura del sistema	33
5.2.3	Diseño de la interfaz de usuario	35
5.3	Diseño de la aplicación de análisis de datos	40
5.3.1	Introducción al diseño de la aplicación de análisis de datos	40
5.3.2	Estudio y diseño de la aplicación de análisis de datos	40

6 Implementación	45
6.1 Introducción a la implementación	45
6.2 Implementación de la aplicación OTT	45
6.2.1 Desarrollo de las funcionalidades básicas	45
6.2.2 Adaptación a los diferentes sistemas operativos de televisión	52
6.3 Implementación de la aplicación de análisis de datos	56
6.3.1 Instalación, configuración e integración de la API de Matomo	56
6.3.2 Desarrollo de la aplicación de análisis de datos	57
6.3.3 APIs y librerías utilizadas	65
7 Pruebas	69
7.1 Introducción	69
7.1.1 Plan de pruebas unitarias, integración y sistema	69
7.1.2 Pruebas de aceptación	70
7.1.3 Pruebas realizadas por los marketplaces	71
7.2 Resultados de las pruebas	71
8 Conclusiones	72
8.1 Logros Alcanzados	72
8.2 Desafíos y Soluciones	73
8.3 Aprendizajes	73
8.4 Conclusión Final	73
9 Futuros Objetivos	74
Bibliografía	78

Índice de Figuras

5.1	Captura de la interfaz en las primeras fases del proyecto	36
5.2	Captura de la interfaz más reciente, aún en pruebas por parte del cliente	36
5.3	Captura de la interfaz casi final	37
5.4	Ejemplos de un widget básico de tipo "banner" y "destacado"	39
5.5	Ejemplos de un widget básico de tipo "poster"	40
5.6	Página de inicio de la aplicación de análisis de datos	41
5.7	Página de comparador de la aplicación de análisis de datos	42
5.8	Ejemplo de grafico circular mostrando la distribución de los tipos de dispositivos en los que se accede a la web	43
5.9	Ejemplo de gráfico lineal mostrando la evolución del número de visitantes únicos en la web	43
6.1	Menú lateral	48
6.2	Pantalla de detalle de un contenedor	50
6.3	Pantalla de detalle	50

Índice de Táboas

Capítulo 1

Introducción

1.1 Contexto y motivación

¿Netflix? ¿HBO? ¿DAZN? ¿Amazon Prime Video? ¿Disney+? Es muy probable que casi todos los lectores de este documento hayan utilizado alguna de estas aplicaciones en los últimos meses. De hecho, suelen ser de las primeras opciones que consideramos al descargar aplicaciones en nuestros dispositivos, y en muchos casos, incluso vienen preinstaladas. Según un estudio realizado por la Comisión Nacional de los Mercados y la Competencia (CNMC), el 58% de los hogares españoles con acceso a Internet usaba algún servicio de vídeo en streaming a mediados del año 2023, frente al 37% de mediados de 2019 [1]. Otras fuentes sitúan este porcentaje en un 81% a principios de 2023 [2] e incluso en un 95% en junio de 2024 [3]. Estas cifras varían un poco dependiendo de la fuente, pero todas coinciden en algo: el consumo de contenido audiovisual a través de plataformas de streaming está en auge y ya supera a las plataformas televisivas tradicionales.

Leyendo estos estudios y observando la evolución de este mercado, se entiende el por qué de la constante aparición de nuevas plataformas de este estilo. Desde películas y series hasta deportes, pasando por documentales, cursos, conciertos, etc., estas plataformas se adaptan a cualquier estilo de contenido y a cualquier tipo de usuario.

No es únicamente el número de usuarios el que ha aumentado, sino también el tiempo que pasamos viendo contenido en estas plataformas, el número de plataformas distintas a las que accedemos, y el número de dispositivos en los que lo hacemos. La causa de esto es la facilidad de acceso a estas plataformas desde cualquier lugar y dispositivo. Y es que, ¿quién no ha empezado a ver una serie en la televisión del salón, ha continuado en la tablet de la cocina y ha terminado en el móvil de la cama? ¿O quién no ha parado la serie para cambiar de plataforma y ver un partido de fútbol? El consumo de contenido audiovisual nunca había sido tan sencillo y accesible.

Sencillo para el consumidor, claro, pero para lograr esta versatilidad y facilidad de uso,

hay un gran trabajo detrás para asegurar que estas plataformas sean funcionales sin importar el tamaño, la marca o el sistema operativo del dispositivo, ni el lugar en el que se encuentre el usuario.

Aquí es donde surge la necesidad de desarrollar soluciones tecnológicas que permitan a las empresas ofrecer experiencias de usuario consistentes y personalizadas en múltiples dispositivos y sistemas operativos. El mercado creciente de plataformas OTT ha creado una demanda significativa para soluciones que no solo se adapten a esta diversidad tecnológica, sino que también permitan a diferentes clientes configurar sus propias plataformas con facilidad.

1.2 Objetivo del trabajo

1.2.1 Desarrollo de una aplicación OTT multiplataforma y multicliente

En este contexto, el objetivo de este trabajo es desarrollar una aplicación multiplataforma para la visualización de contenido audiovisual en streaming, diseñada para adaptarse a una amplia variedad de dispositivos y sistemas operativos, así como para ser utilizada por múltiples clientes.

El proyecto abordará todas las fases de desarrollo de una aplicación OTT (Over The Top), con dos características principales: multiplataforma y multicliente. Multiplataforma porque la aplicación está orientada a poder adaptarse a cualquier dispositivo y sistema operativo. Multicliente porque el código no está pensado únicamente para un cliente en concreto, sino para ser utilizado por cualquier cliente que quiera tener su propia plataforma donde mostrar su contenido.

Estos enfoques suponen varios retos a nivel de desarrollo, los principales siendo la adaptación a las distintas necesidades de cada dispositivo donde se quiera dar soporte a la aplicación, con las características y limitaciones que conlleva utilizar las distintas plataformas, tecnologías y sistemas operativos; y la adaptación a las distintas necesidades de cada cliente, tratando de mantener un equilibrio entre la personalización de la aplicación y la reutilización del código.

1.2.2 Desarrollo e integración de una aplicación para la recogida y visualización de datos de uso

Además, se desarrollará una aplicación complementaria destinada a la recogida y visualización de datos de uso. En el entorno competitivo actual, conocer a los usuarios y entender su comportamiento es crucial para mejorar la experiencia de usuario y mantener a los usuarios activos en la plataforma. Por ello, esta aplicación permitirá a las empresas analizar en detalle cómo interactúan los usuarios con la plataforma OTT, brindando información valiosa para la

toma de decisiones.

El desarrollo incluye el análisis y selección de las métricas necesarias para conocer el comportamiento de los usuarios en este tipo de plataformas, la integración de la API de analítica en la OTT y el desarrollo de una interfaz intuitiva para la visualización de los datos recogidos.

Capítulo 2

Fundamentos

2.1 Introducción

El desarrollo de aplicaciones multiplataforma representa un desafío técnico considerable en el entorno tecnológico actual, caracterizado por una amplia variedad de dispositivos y plataformas. Para garantizar el éxito de un proyecto de estas características, es esencial comprender tanto los principios teóricos que permiten la creación de aplicaciones flexibles y escalables como los fundamentos tecnológicos que sustentan su desarrollo.

En este capítulo se exponen los fundamentos teóricos y tecnológicos que son cruciales para el desarrollo de las aplicaciones propuestas en este trabajo. Se explorarán conceptos clave como el funcionamiento de las aplicaciones OTT, la experiencia de usuario, el diseño de interfaces, la adaptabilidad y la escalabilidad. Asimismo, se analizarán las tecnologías esenciales para desarrollar una aplicación que se ajuste tanto a las necesidades del cliente como a las especificaciones de la plataforma, así como para integrar una plataforma analítica complementaria.

2.2 Fundamentos teóricos

2.2.1 OTT (Over-the-Top)

Definición, funcionamiento y clasificación de los servicios OTT

En radiodifusión, un servicio OTT (siglas en inglés de *over-the-top*) consiste en la transmisión de audio, vídeo y otros contenidos a través de internet sin la implicación de los operadores tradicionales en el control o la distribución de los mismos [4].

Estos servicios han transformado radicalmente la forma en que los usuarios consumen contenido audiovisual y cómo los proveedores lo distribuyen. La clave de esta transformación radica en la capacidad de acceder a contenidos de manera inmediata, sin necesidad de esperar

a que se emitan en televisión, radio, cine u otros medios tradicionales. Además, los servicios OTT permiten a los usuarios disfrutar de contenidos en cualquier lugar y momento, siempre que cuenten con una conexión a internet.

Su funcionamiento se basa en un catálogo de contenidos que los usuarios pueden explorar según sus preferencias. Cada catálogo incluye una variedad de películas, series, documentales y otros tipos de contenido, presentados con descripciones detalladas, géneros, calificaciones de usuarios y recomendaciones basadas en el historial de visualización. Los usuarios pueden navegar por el catálogo, seleccionar el contenido que desean ver y comenzar la transmisión de manera inmediata, beneficiándose de la capacidad de pausar, rebobinar o adelantar el contenido según su conveniencia. Este sistema flexible no solo permite a los usuarios tener control total sobre lo que ven y cuándo lo ven, sino que también facilita la entrega personalizada de contenido, adaptándose continuamente a los gustos individuales de cada usuario.

El contenido ofrecido por los servicios OTT es muy variado, abarcando desde películas y series hasta música, deportes, noticias, cursos educativos y recursos generados por los usuarios. Esta diversidad permite que los servicios OTT se adapten a diferentes preferencias y necesidades, ofreciendo una experiencia personalizada para cada tipo de usuario. Además, el acceso a datos masivos sobre los hábitos de consumo de los usuarios ha permitido a las plataformas OTT optimizar sus catálogos y estrategias de marketing, logrando una conexión más profunda y relevante con su audiencia.

En función del modelo de negocio y la forma en que se ofrece el contenido, los principales tipos de servicios OTT son:

- **SVOD (Subscription Video on Demand):** Servicios que ofrecen acceso ilimitado a un catálogo de contenido a cambio de una suscripción mensual o anual.
- **AVOD (Advertising Video on Demand):** Servicios que proporcionan contenido gratuito a los usuarios a cambio de la visualización de anuncios publicitarios.
- **TVOD (Transactional Video on Demand):** Servicios que permiten a los usuarios alquilar o comprar contenido de manera individual.

Evolución y tendencias de los servicios OTT

El crecimiento de los servicios OTT ha sido impulsado por varios factores clave, como la mejora de las conexiones a internet, que ha permitido una transmisión más rápida y de mayor calidad, y los avances tecnológicos en la compresión de video y la optimización de redes, que han mejorado significativamente la experiencia del usuario. Estas aplicaciones deben ser diseñadas para escalar eficientemente, manejando grandes volúmenes de tráfico durante eventos en vivo o lanzamientos populares, lo que plantea desafíos en términos de

rendimiento. Además, la personalización se ha convertido en un diferenciador esencial, impulsada por algoritmos de inteligencia artificial que analizan los patrones de consumo para ofrecer contenido relevante, mejorando la experiencia del usuario y fomentando la lealtad a la plataforma. La proliferación de dispositivos móviles y la preferencia por el contenido instantáneo han consolidado a los servicios OTT como la opción preferida frente a los medios tradicionales, reflejando una evolución en los hábitos de consumo hacia la conveniencia y la personalización.

El entorno competitivo también ha impulsado innovaciones en la oferta de servicios OTT. Las empresas han tenido que diferenciarse mediante la creación de contenido original exclusivo, el uso de inteligencia artificial para mejorar la personalización y la implementación de modelos de negocio híbridos que combinan suscripciones, publicidad y transacciones para maximizar el alcance y la rentabilidad. Estas estrategias no solo buscan atraer y retener a los usuarios, sino también garantizar la sostenibilidad del servicio en un mercado cada vez más fragmentado.

2.2.2 Arquitectura

La arquitectura de software se refiere a los modelos y estándares que sirven de base para el diseño e implementación de sistemas de software. Define la estructura, el funcionamiento y la interacción de los componentes de software [5].

El diseño de la arquitectura de un sistema de software para una aplicación OTT es crucial para garantizar su correcto funcionamiento, así como una entrega eficiente, escalable y confiable de los contenidos. Existen varias opciones para diseñar la arquitectura de una aplicación de estas características, como la Arquitectura Monolítica o la Arquitectura Orientada a Servicios (SOA). En este caso, se ha optado por una arquitectura basada en microservicios.

Arquitectura basada en microservicios

La arquitectura basada en microservicios es un enfoque de diseño de aplicaciones que consiste en un conjunto de pequeños servicios, los cuales se ejecutan de forma independiente y se comunican mediante mecanismos ligeros (como una API de recursos HTTP, como en este proyecto) [6]. Un microservicio se encarga de una única funcionalidad de la aplicación, lo que le permite operar de manera autónoma y comunicarse con otros microservicios según sea necesario.

En esta arquitectura, las funcionalidades de la plataforma OTT se distribuyen en diferentes microservicios, lo que permite que cada uno sea desarrollado, desplegado y escalado de forma independiente. Esto facilita la evolución de la plataforma, ya que cada microservicio puede ser mejorado sin afectar al resto del sistema. Además, una buena arquitectura de microservicios mejora la tolerancia a fallos; si uno falla, el resto de la plataforma debería continuar

funcionando correctamente o verse mínimamente afectado.

Ejemplos de microservicios comunes en una plataforma OTT son:

- **Gestión de usuarios:** Encargado de la gestión de usuarios, incluyendo registro, autenticación y autorización.
- **Gestión de contenidos:** Responsable de la gestión y almacenamiento de los contenidos, así como de los metadatos y archivos multimedia.
- **Orquestador:** Coordina las peticiones de los usuarios a los diferentes microservicios de la plataforma.
- **Recomendaciones:** Genera recomendaciones personalizadas para los usuarios.

Componentes de la arquitectura

Además de los microservicios, una plataforma OTT incluye varios componentes clave que conforman la infraestructura técnica esencial para su funcionamiento. Estos componentes soportan las operaciones críticas, desde la ingestión y distribución de contenido hasta la experiencia de usuario final. Algunos de los componentes más importantes son:

- **CDN (Content Delivery Network):** Red de distribución de contenidos que permite la entrega rápida y eficiente de contenidos multimedia a los usuarios finales. La CDN almacena copias de los contenidos en servidores distribuidos geográficamente, reduciendo la latencia y mejorando la velocidad de carga.
- **Ingesta y gestión de contenidos:** Componente que recibe, procesa y gestiona todos los contenidos que estarán disponibles en la plataforma. Permite a los administradores subir, editar, etiquetar y organizar los contenidos y sus metadatos para su distribución.
- **Gestión de usuarios:** Responsable de todas las funciones relacionadas con los usuarios, incluyendo registro, autenticación, autorización, gestión de perfiles y preferencias.
- **Interfaz de usuario:** Es la parte visible de la plataforma, donde los usuarios interactúan con ella. Recibe la información de los microservicios y la presenta de manera amigable.
- **Monitorización y análisis:** Componente encargado de monitorizar el rendimiento de la plataforma y de analizar los datos generados por los usuarios, con el fin de mejorar la experiencia de usuario y la eficiencia del sistema.
- **Otros componentes:** Incluye componentes de seguridad, monetización, publicidad, gestión de pagos, entre otros, que pueden ser necesarios según las características de la plataforma.

2.2.3 Experiencia de usuario (UX)

A diferencia de los medios tradicionales, donde el contenido se emite en un horario fijo y de una forma específica, sin permitir que el usuario interactúe con él, los servicios OTT ofrecen a los usuarios la posibilidad de navegar por un catálogo de contenidos, seleccionar lo que desean ver, consultar su información, su estructura (temporadas, episodios, etc.), y acceder al contenido en cualquier momento y lugar, sin restricciones. Esta flexibilidad y control que los usuarios desean sobre el contenido es tanto una de las principales ventajas de los servicios OTT como uno de los mayores desafíos para los diseñadores.

La experiencia de usuario (UX) [7] se define como el conjunto de factores y elementos que afectan la interacción del usuario con la interfaz de un sistema, dispositivo o aplicación. Estos factores son clave para determinar si un usuario disfruta de la experiencia de uso de un producto o servicio. Es fundamental prestar atención a estos aspectos en el diseño de cualquier aplicación, y más aún en el caso de una plataforma OTT, donde los usuarios buscan una experiencia fluida, personalizada, intuitiva y atractiva.

Principios de diseño de UX

El diseño de una experiencia de usuario efectiva se basa en una serie de principios y prácticas que deben adaptarse a cada proyecto, sus necesidades y características. En el caso de las aplicaciones OTT, dos de los principios fundamentales son la usabilidad y la simplicidad. La plataforma debe ser fácil de usar, intuitiva y accesible para todo tipo de usuarios, independientemente de su nivel de experiencia o conocimientos técnicos. Esto garantiza que cualquier usuario pueda utilizar nuestras plataformas. Otros principios importantes incluyen la consistencia y la adaptabilidad entre dispositivos y plataformas, la accesibilidad para asegurar que todos los usuarios puedan disfrutar de la plataforma, y la personalización para ofrecer una experiencia única y relevante a cada usuario.

Elementos clave de la UX en una plataforma OTT

Las plataformas OTT son aplicaciones con una serie de componentes clave que son esenciales para su funcionamiento. Estos componentes no solo deben estar presentes, sino que también deben funcionar de manera óptima para garantizar una buena experiencia de usuario. Los componentes más importantes son:

- **Contenidos:** El elemento central de la plataforma, en torno al cual gira toda la experiencia de usuario. Los contenidos deben ser fáciles de encontrar, navegar y consumir, y deben estar presentados de forma atractiva y organizada.

- **Agrupaciones de contenidos:** Ya sea en listas, series, temporadas, categorías, géneros, etc., las agrupaciones de contenidos permiten a los usuarios explorar y descubrir nuevos contenidos de forma sencilla y atractiva.
- **Metadata:** Información sobre cualquiera de los elementos que conforman la plataforma, como descripciones, fechas, títulos, actores, etc. La metadata proporciona el contexto necesario para que los usuarios entiendan y valoren los contenidos.
- **Páginas:** Cada menú, serie, temporada, episodio, etc., debe tener su propia página con la información y funcionalidades necesarias para facilitar el uso de la plataforma.
- **Botones y controles:** Los botones y controles de reproducción, pausa, adelanto, retroceso, etc., deben ser intuitivos y accesibles para que los usuarios puedan interactuar con el contenido de forma sencilla.

Elementos clave para la UX de la plataforma de análisis de datos

La base que se siguió para la creación de la plataforma de análisis de datos fue la de ofrecer una experiencia de usuario sencilla y efectiva, que permitiera a los usuarios visualizar y analizar los datos de una manera clara.

La plataforma de análisis de datos es una herramienta de gran utilidad para los usuarios, ya que les permite visualizar y analizar los datos de una manera sencilla y efectiva. Algunos de los elementos clave para la experiencia de usuario de esta plataforma son:

- **Visualización de datos:** La plataforma debe permitir a los usuarios visualizar los datos de forma clara y efectiva, utilizando gráficos, tablas y otros elementos visuales para facilitar la comprensión de la información.
- **Análisis de datos:** Los usuarios deben poder analizar los datos de manera sencilla, utilizando herramientas como filtros, agrupaciones, páginas, etc., para extraer información relevante y tomar decisiones informadas.
- **Generación de análisis:** La plataforma debe permitir a los usuarios generar análisis personalizados con los datos seleccionados, para conseguir una mayor profundidad y detalle en la información.
- **Interfaz de usuario:** La interfaz de usuario debe ser intuitiva y fácil de usar, con un diseño limpio y atractivo que facilite la navegación y el uso de la plataforma.

2.2.4 Escalabilidad, adaptabilidad y multiplataforma

La escalabilidad, adaptabilidad y multiplataforma son conceptos fundamentales en el diseño y desarrollo de aplicaciones, especialmente en el caso de estudio de este proyecto: una plataforma OTT con soporte multicliente y multiplataforma y una aplicación de análisis de datos para varias aplicaciones.

Escalabilidad

La escalabilidad se refiere a la capacidad de un sistema para crecer y adaptarse a nuevas necesidades, sin comprometer su rendimiento o estabilidad. En este proyecto, la escalabilidad se centra principalmente en la capacidad de la plataforma para ampliar su catálogo de servicios y funcionalidades, adaptándose a las demandas de los usuarios y a las necesidades del mercado.

En lugar de enfocarse en la infraestructura de servidores, la escalabilidad aquí implica la posibilidad de integrar nuevas funcionalidades de manera modular, permitiendo que la plataforma evolucione y mejore continuamente. Esta capacidad de expansión es crucial para mantener la relevancia de la plataforma en un entorno en constante cambio, permitiendo la incorporación de nuevas características sin afectar las existentes.

Además, la plataforma debe estar preparada para futuras actualizaciones y mejoras, de modo que pueda adaptarse a los cambios en las preferencias de los usuarios y en las tendencias del mercado. La escalabilidad funcional es esencial para asegurar que la plataforma pueda crecer de manera sostenible y seguir siendo competitiva, al tiempo que ofrece una experiencia de usuario coherente y de alta calidad.

Adaptabilidad

La adaptabilidad es la capacidad de un sistema para ajustarse a diferentes contextos, entornos y dispositivos. En una plataforma OTT, la adaptabilidad implica ofrecer una experiencia de usuario consistente y de alta calidad en una amplia variedad de dispositivos, sistemas operativos y navegadores.

Es crucial que los usuarios puedan acceder al contenido de la plataforma desde cualquier dispositivo y en cualquier momento, sin importar las limitaciones técnicas o las preferencias personales. La plataforma debe ser capaz de adaptarse automáticamente a las características de cada dispositivo, como el tamaño de la pantalla, la resolución, la velocidad de conexión y la capacidad de procesamiento, para ofrecer una experiencia de usuario óptima.

Además, la adaptabilidad implica la capacidad de la plataforma para ajustarse a los cambios en el mercado, las tecnologías y las preferencias de los usuarios. La plataforma debe ser flexible y modular, permitiendo la integración de nuevas funcionalidades, servicios y contenidos de manera rápida y sencilla, sin comprometer la estabilidad y el rendimiento del sistema.

Multiplataforma

La multiplataforma es la capacidad de un sistema para funcionar en diferentes dispositivos, sistemas operativos y navegadores. En una plataforma OTT, la multiplataforma implica ofrecer una experiencia de usuario consistente y de alta calidad en una amplia variedad de dispositivos, como ordenadores, smartphones, tablets, smart TVs y consolas de videojuegos.

Esta característica es fundamental para garantizar que los usuarios puedan acceder al contenido de la plataforma desde cualquier dispositivo y en cualquier momento, aumentando así la cantidad de usuarios a los que se puede llegar y mejorando la visibilidad y rentabilidad de la plataforma.

Además, la multiplataforma incluye la capacidad de la plataforma para integrarse con otros sistemas y servicios, como redes sociales, sistemas de pago, motores de recomendación y herramientas de análisis, para ofrecer una experiencia de usuario completa y personalizada, adaptada a las necesidades y preferencias de cada usuario.

Conclusiones

En resumen, la escalabilidad, adaptabilidad y multiplataforma son conceptos fundamentales en el diseño y desarrollo de una plataforma OTT, ya que permiten garantizar que la plataforma pueda crecer de manera sostenible, adaptarse a los cambios en el mercado y en las tecnologías, y ofrecer una experiencia de usuario consistente y de alta calidad en una amplia variedad de dispositivos y contextos.

2.2.5 Importancia de la Analítica de Datos en Plataformas OTT

La analítica de datos juega un papel crucial en las plataformas OTT, ya que permite a las empresas obtener una comprensión profunda del comportamiento del usuario y, en consecuencia, optimizar la experiencia de usuario (UX). A través de la recopilación y análisis de datos, las plataformas pueden identificar patrones de consumo, preferencias del usuario, y áreas de mejora, lo que a su vez facilita la toma de decisiones informadas sobre la oferta de contenido y el desarrollo de nuevas funcionalidades.

Herramientas y Frameworks: Matomo

Para la analítica de datos en esta plataforma OTT, se ha utilizado Matomo, una herramienta de código abierto que permite la recopilación y análisis de datos de manera eficaz y segura. Matomo fue seleccionado por su capacidad para integrarse fácilmente con la plataforma y su flexibilidad en la configuración de métricas específicas según las necesidades del cliente.

Matomo permite la captura de datos de usuario en tiempo real, incluyendo métricas clave como el tiempo de visualización, las tasas de abandono, y el comportamiento de navegación.

Además, ofrece una interfaz intuitiva para la visualización de estos datos, facilitando la interpretación y toma de decisiones.

Integración con la Plataforma OTT

La integración de Matomo con la plataforma OTT se realiza a través de APIs que permiten la recopilación automatizada de datos de usuario desde diversos puntos de interacción dentro de la aplicación. Estos datos son procesados y almacenados para su posterior análisis, permitiendo a los administradores acceder a informes detallados y dashboards interactivos que ofrecen una visión completa del uso y rendimiento de la plataforma.

2.2.6 Aplicación de los Principios de UX en la Analítica de Datos

Al igual que en la plataforma OTT principal, los principios de UX son fundamentales en la aplicación de análisis de datos. Es esencial que la interfaz de la aplicación de analítica sea intuitiva, con una presentación clara y accesible de los datos, lo que permitirá a los administradores tomar decisiones rápidas y efectivas. La interactividad y la capacidad de personalizar informes son aspectos clave que garantizan que los usuarios puedan filtrar y segmentar los datos según sus necesidades específicas, optimizando así la utilidad de la herramienta.

2.3 Fundamentos tecnológicos

2.3.1 Arquitectura Software: Comunicación entre componentes y desarrollo multicliente

Comunicación entre componentes

Una de las claves de la arquitectura software [2.2.2](#) es la comunicación entre componentes y microservicios [2.2.2](#). En esta aplicación, dicha comunicación se realiza a través de una API REST, que es una interfaz de programación de aplicaciones que sigue los principios de diseño del estilo arquitectónico de transferencia de estado representacional (REST).

REST es un estilo de arquitectura basado en la comunicación mediante HTTP, caracterizado por su sencillez, escalabilidad y adaptabilidad a diversos entornos. En una API REST, los recursos son identificados por URLs y se accede a ellos a través de los métodos HTTP estándar (GET, POST, PUT, DELETE). Esto permite que los clientes de la API realicen operaciones sobre los recursos de manera sencilla y estandarizada.

En esta aplicación, la API REST facilita la comunicación entre los diferentes microservicios de la plataforma OTT, proporcionando a la interfaz los medios para obtener la información necesaria para su visualización, y a los microservicios una manera de recibir las peticiones de la interfaz y devolver la información correspondiente.

Desarrollo multicliente

Una de las ventajas que ofrece la API REST en esta aplicación es la flexibilidad para decidir qué información obtener. Dependiendo de los parámetros añadidos a la URL, la API devolverá la información solicitada o realizará una acción específica sobre un cliente determinado.

Esta diferenciación permite, a través de la misma API, gestionar múltiples clientes sin necesidad de desarrollar nuevas APIs o microservicios para cada caso.

2.3.2 Tecnologías Web: JavaScript, HTML y CSS

La plataforma OTT está construida sobre la base de tecnologías web fundamentales: JavaScript (JS), HTML y CSS. Estas tecnologías permiten el desarrollo de una interfaz de usuario interactiva y adaptable, que puede ser fácilmente desplegada en diferentes plataformas, incluyendo la web, smart TVs y dispositivos móviles.

JavaScript (JS) JavaScript es el motor que impulsa la interactividad en la plataforma. Permite manipular el DOM, gestionar eventos del usuario, realizar llamadas asíncronas a la API de la CDN para obtener contenido dinámico y, sobre todo, implementar la lógica de negocio de la plataforma. JavaScript es un lenguaje de programación versátil y flexible, ideal para la creación de aplicaciones web complejas y altamente interactivas.

HTML HTML define la estructura de la interfaz de usuario, organizando el contenido de manera semántica y accesible. Su compatibilidad con todos los navegadores web y dispositivos asegura que la plataforma pueda ser utilizada en una amplia gama de entornos, desde navegadores de escritorio hasta smart TVs.

CSS CSS es responsable de la presentación visual de la plataforma, permitiendo una estilización coherente y atractiva en todas las plataformas soportadas. A través de técnicas de diseño responsive y el uso de transiciones y animaciones, CSS asegura que la interfaz se adapte y ofrezca una experiencia de usuario óptima en cualquier dispositivo.

Ventajas y desventajas de las tecnologías web

Las tecnologías web como JavaScript, HTML y CSS presentan numerosas ventajas, como su amplia compatibilidad, flexibilidad y la posibilidad de desarrollar aplicaciones que pueden ejecutarse en múltiples plataformas sin necesidad de cambios significativos. Estas tecnologías son altamente versátiles, lo que permite una rápida adaptación a nuevos requisitos y la incorporación de funcionalidades avanzadas de manera eficiente. Además, el uso de estándares web asegura que la plataforma sea accesible y usable en una gran variedad de dispositivos y entornos.

Sin embargo, también existen desventajas. El rendimiento de las aplicaciones web puede no igualar al de las aplicaciones nativas en ciertos dispositivos, especialmente en entornos con recursos limitados. Además, la gestión del comportamiento en múltiples plataformas puede requerir un esfuerzo adicional para asegurar una experiencia de usuario coherente.

La elección de estas tecnologías para el desarrollo de la plataforma OTT se justifica por su capacidad de ofrecer una solución multiplataforma eficiente y adaptable, permitiendo un desarrollo más rápido y una mayor flexibilidad en la personalización y mantenimiento de la plataforma, a pesar de las limitaciones inherentes a las tecnologías web.

2.3.3 Desarrollo multiplataforma

El desarrollo de una plataforma OTT que funcione de manera eficiente en múltiples dispositivos requiere un enfoque que combine la flexibilidad de las tecnologías web con herramientas específicas para adaptar la aplicación a distintos sistemas operativos y dispositivos. La aplicación se ha desarrollado utilizando JavaScript (JS), HTML y CSS, cuyas ventajas y desventajas se han discutido en la sección 2.3.2.

Adaptación a Web

Gracias a la utilización de JavaScript, HTML y CSS, la adaptación de la aplicación a la web es sencilla y directa. La aplicación se ha desarrollado siguiendo los principios de diseño responsive, lo que permite que la interfaz se ajuste automáticamente al tamaño de la pantalla del dispositivo en el que se visualiza. Además, la aplicación se ha probado en los principales navegadores web (Chrome, Firefox, Safari, Edge) para garantizar su compatibilidad y correcto funcionamiento en distintos entornos.

Adaptación a Smart TVs

La adaptación de la aplicación a televisores inteligentes es un proceso más complejo, ya que estos dispositivos suelen tener sistemas operativos propietarios con limitaciones y peculiaridades específicas. Cada fabricante de televisores inteligentes tiene su propio sistema operativo y plataforma de desarrollo, así como su propia manera de empaquetar, ejecutar y distribuir aplicaciones. Por ello, la adaptación de la aplicación a televisores inteligentes requiere un estudio y enfoque específicos para cada plataforma.

Tizen

Tizen [8] es un sistema operativo de código abierto desarrollado por Samsung Electronics y la Linux Foundation. Tizen utiliza tecnologías web como HTML, CSS y JavaScript para el desarrollo de aplicaciones, lo que facilita la adaptación de la aplicación a este sistema

operativo. Las peculiaridades de Tizen, como su sistema de empaquetado y distribución de aplicaciones, se han tenido en cuenta durante el desarrollo para garantizar su compatibilidad y correcto funcionamiento. La aplicación ha sido probada utilizando la plataforma Tizen Studio, que permite emular el funcionamiento en un televisor Samsung con Tizen OS e instalar y ejecutar la aplicación en un dispositivo real. El formato de empaquetado para Tizen es un archivo .wgt, que contiene todos los recursos y archivos necesarios para la ejecución y distribución de la aplicación en Tizen OS.

WebOS

WebOS [9] es un sistema operativo de código abierto utilizado por LG Electronics para sus televisores inteligentes. WebOS también utiliza tecnologías web como HTML, CSS y JavaScript para el desarrollo de aplicaciones, permitiendo una fácil adaptación a este sistema operativo. LG proporciona un CLI (Command Line Interface) para la creación, empaquetado y distribución de aplicaciones para WebOS. Además, está disponible una extensión para Visual Studio Code que facilita aún más el desarrollo. La aplicación ha sido probada en un televisor LG con WebOS utilizando ambas herramientas, garantizando su compatibilidad y funcionamiento. El formato de empaquetado para WebOS es un archivo .ipk.

Android TV

Android TV [10] es una plataforma de televisión inteligente desarrollada por Google, basada en el sistema operativo Android. La adaptación de la aplicación a este sistema operativo varía en comparación con las anteriores, ya que Android no permite el uso nativo de tecnologías como JavaScript, HTML y CSS. Por ello, es necesario un proceso de conversión para hacer la aplicación compatible con Android TV.

Conversión a Android TV: Cordova Para adaptar la aplicación a Android TV, se ha utilizado la herramienta Cordova [11], que permite utilizar tecnologías estándar en lugar de las tecnologías nativas de determinadas plataformas, como en el caso de Android.

Apache Cordova es una plataforma de desarrollo móvil que permite a los desarrolladores crear aplicaciones multiplataforma utilizando tecnologías web estándar como HTML5, CSS3 y JavaScript. Una de las principales ventajas de Cordova es que permite acceder a las funcionalidades nativas del dispositivo a través de APIs, facilitando la creación de aplicaciones que pueden ejecutarse en múltiples plataformas sin necesidad de escribir código específico para cada una. Esto no solo reduce el tiempo de desarrollo, sino que también simplifica el mantenimiento y la actualización de las aplicaciones.

Cordova cuenta con una amplia comunidad de desarrolladores y una gran cantidad de plugins que extienden sus capacidades. Estos plugins permiten integrar funcionalidades adi-

cionales como el acceso a la cámara, el almacenamiento local, las notificaciones push, y más. La flexibilidad y extensibilidad de Cordova lo convierten en una opción popular para el desarrollo de aplicaciones móviles híbridas, especialmente en proyectos donde el tiempo y los recursos son limitados. La capacidad de Cordova para adaptarse a diferentes plataformas, incluyendo Android TV, demuestra su versatilidad y potencia como herramienta de desarrollo.

La aplicación ha sido probada en un dispositivo Android TV, garantizando su compatibilidad y funcionamiento. El formato de empaquetado para Android TV es un archivo .apk.

Empaquetado y ejecución de la aplicación

Para las aplicaciones de TV, se ha creado una plantilla que contiene los archivos necesarios para la ejecución de la aplicación en los distintos sistemas operativos de televisores inteligentes. La estructura de la aplicación permite que, en el momento de empaquetar para prueba o distribución, sea sencillo añadir el código a la plantilla y ejecutar las herramientas necesarias para cada sistema operativo.

2.3.4 Rendimiento y Optimización

El rendimiento es un aspecto crítico en el desarrollo de la plataforma OTT, ya que influye directamente en la experiencia del usuario. Para garantizar un rendimiento óptimo, se han implementado diversas estrategias de optimización tanto en el frontend como en las interacciones entre el frontend y el backend.

En el frontend, se han optimizado los movimientos y transiciones utilizando técnicas avanzadas de CSS y se han creado funcionalidades a través de JavaScript para asegurar una experiencia de usuario fluida y agradable. Estas optimizaciones permiten que las animaciones, transiciones y otros efectos visuales se ejecuten de manera eficiente, mejorando la percepción de velocidad y la interactividad de la aplicación. Además, se ha trabajado en la estructura HTML y CSS para asegurar una carga eficiente de los recursos visuales, evitando bloqueos o ralentizaciones innecesarias.

Otra área clave de optimización ha sido la gestión eficiente de las llamadas a la API y el manejo de los datos recibidos. Se han implementado técnicas de concurrencia y paralelismo para acelerar las operaciones de red y minimizar el tiempo de espera al cargar o actualizar contenido dinámico. Estas optimizaciones aseguran que la interfaz de usuario sea ágil y responsive, incluso en entornos con conexiones de red variables.

En cuanto al backend, aunque no es el enfoque principal de este proyecto, se han mencionado optimizaciones generales como el uso de caching y la optimización de consultas a las bases de datos para mejorar la eficiencia. Sin embargo, estas mejoras en el backend no están directamente relacionadas con el trabajo desarrollado en este proyecto.

2.3.5 Gestión de Versiones (Git)

La gestión de versiones en este proyecto se ha llevado a cabo utilizando Git, un sistema de control de versiones distribuido que permite a los desarrolladores trabajar de manera colaborativa y gestionar el historial de cambios en el código fuente. Git facilita la creación de ramas para desarrollar nuevas funcionalidades o realizar correcciones sin afectar al código principal.

El uso de Git también permite un control preciso sobre las versiones del software, lo que es esencial para mantener la estabilidad de la plataforma durante las actualizaciones y el despliegue de nuevas funcionalidades.

2.3.6 Uso de APIs

El uso de APIs (Interfaz de Programación de Aplicaciones) es fundamental en la arquitectura de la plataforma OTT. Las APIs permiten la comunicación eficiente entre los diferentes componentes de la aplicación, facilitando la integración con servicios externos y el intercambio de datos entre microservicios.

En este proyecto, la API REST es la principal herramienta de comunicación, permitiendo a los clientes acceder a los recursos de la plataforma a través de métodos HTTP estándar. Las APIs también juegan un papel crucial en la integración de funcionalidades como la autenticación de usuarios, la gestión de contenidos y la recopilación de datos analíticos. La implementación de estas APIs ha sido diseñada para ser escalable y flexible, permitiendo la fácil incorporación de nuevas funcionalidades y servicios en el futuro.

No solo se han utilizado APIs para la comunicación entre los distintos componentes de la plataforma, sino que también se han integrado APIs de terceros para enriquecer la experiencia del usuario. Por ejemplo, se han incorporado APIs para la obtención de los datos de las aplicaciones (Matomo) o el almacenamiento de información y para la generación de análisis sobre gráficas (MongoDb Y OpenAi).

Capítulo 3

Metodología

3.1 Introducción

En esta sección se describe la metodología de trabajo seguida para el desarrollo de esta plataforma OTT. Se detallan las fases de desarrollo, las herramientas utilizadas, y las metodologías de trabajo empleadas para la implementación de la plataforma.

3.1.1 Diseño del proyecto

El objetivo principal del proyecto fue desarrollar una aplicación OTT que, a partir de un único código fuente, pudiera adaptarse a diversos dispositivos y cumplir con las necesidades de diferentes clientes. Además, se planificó el desarrollo de una aplicación complementaria para el análisis de datos de uso, destinada a mejorar la comprensión del comportamiento de los usuarios y optimizar la experiencia de uso.

La primera fase del proyecto consistió en el análisis y diseño de ambas plataformas, estableciendo los siguientes objetivos generales:

- Crear una plataforma OTT que permita a los clientes tener su propia aplicación de distribución de contenido.
- Ofrecer una alta capacidad de personalización para que los clientes adapten la aplicación a su marca y contenido.
- Desarrollar una plataforma escalable y adaptable a las diversas necesidades de cada cliente.
- Asegurar que el código funcione en la mayor cantidad de dispositivos posible.
- Desarrollar una aplicación de análisis de datos que recopile y visualice información sobre el uso de la plataforma OTT, facilitando la toma de decisiones y la mejora continua del servicio.

Meta principal: Evitar la necesidad de crear un código fuente distinto para cada cliente y dispositivo.

Con la idea general del proyecto clara, se definieron los requisitos, funcionalidades, y el alcance tanto de la plataforma OTT como de la aplicación de análisis de datos. Se realizó un análisis de mercado, priorizando el soporte inicial para los sistemas Android, Tizen y WebOS, con miras a ampliar a otros sistemas en el futuro.

También se revisaron productos similares desarrollados previamente por la empresa, para establecer una base sólida en términos de características visuales y funcionales, que luego se mejorarían y evolucionarían.

Se planificaron las tareas y funcionalidades prioritarias para lograr un producto mínimo viable lo antes posible, con el fin de comenzar las pruebas y presentaciones tanto de la plataforma OTT como de la aplicación de análisis de datos.

3.2 Metodología de Desarrollo

El desarrollo tanto de la plataforma OTT como de la aplicación de análisis de datos se llevó a cabo siguiendo una metodología ágil. Esta metodología permitió una mayor flexibilidad en el desarrollo de ambos proyectos, facilitando la adaptación a los cambios y asegurando la evolución y mejora continua de los productos finales.

3.2.1 Implementación progresiva e incremental

El proceso de desarrollo se estructuró en ciclos pequeños y manejables, cada uno enfocado en la implementación de una funcionalidad específica. Cada ciclo incluía las siguientes fases:

- **Análisis:** Se identificaban los requisitos de la funcionalidad a implementar, priorizando según su importancia y dependencia de otras funcionalidades.
- **Diseño:** Se diseñaba la arquitectura necesaria, definiendo componentes e interacciones.
- **Implementación:** La funcionalidad se desarrollaba según el diseño especificado.
- **Pruebas:** Se realizaban pruebas unitarias y de integración para asegurar que la funcionalidad cumpliera con los requisitos.

3.2.2 Adaptación continua y reuniones de validación

Una de las principales ventajas de la metodología ágil fue la capacidad de adaptación continua. Reuniones periódicas con los clientes permitieron revisar el progreso, validar las funcionalidades y ajustar las prioridades según fuera necesario. Estas sesiones garantizaron

que el proyecto se mantuviera alineado con las expectativas del cliente y permitieron una respuesta rápida a cualquier cambio o nueva necesidad.

3.2.3 Ventajas de la metodología ágil

El enfoque ágil ofreció varias ventajas clave para el desarrollo del proyecto:

- **Flexibilidad y Adaptación:** Posibilidad de ajustar el proyecto a nuevas necesidades durante su desarrollo.
- **Control y Seguimiento:** División del proyecto en ciclos pequeños facilitó el seguimiento detallado del progreso.
- **Calidad y Satisfacción del Cliente:** Pruebas continuas y participación activa del cliente aseguraron un producto final que cumplió con las expectativas y estándares de calidad.

3.3 Descripción de las Etapas de Desarrollo

En esta sección, se describen las principales etapas del desarrollo del proyecto, desde la planificación inicial hasta el despliegue final. Cada etapa jugó un papel crucial en la evolución del proyecto, permitiendo una implementación organizada y eficiente de la plataforma OTT.

3.3.1 Fase de Análisis y Planificación

Recolección de Requisitos

En esta etapa, se realizó un análisis exhaustivo para identificar los requisitos funcionales y no funcionales del proyecto. Esto incluyó reuniones con los clientes para entender las expectativas y necesidades de cada uno, así como un análisis de mercado para asegurar que las plataformas cumplieran con los estándares actuales.

3.3.2 Fase de Diseño

Diseño de Arquitectura

Durante esta etapa, se definió la arquitectura general de la plataforma OTT con sus componentes. Se utilizó una arquitectura basada en microservicios para garantizar la escalabilidad y flexibilidad del sistema e integración con los servicios existentes en la empresa. También se diseñó la estructura de comunicación entre los componentes y se eligieron las tecnologías más adecuadas para cada parte del sistema.

Se estudiaron los conjuntos de datos y las APIs necesarias para creación de la plataforma de análisis de datos.

Diseño de la Interfaz de Usuario

Se comenzó a trabajar sobre una interfaz básica para la plataforma OTT que permitiera el desarrollo de las funcionalidades principales de la plataforma y poco a poco se fue refinando y mejorando en función de las peticiones de los clientes y las pruebas de usabilidad realizadas. A través de reuniones con los clientes y analizando sus peticiones y prototipos, se fueron contruyendo los diseños de manera conjunta hasta llegar a un diseño final que satisfaciera las necesidades de los usuarios.

Se diseñó la interfaz de usuario para la plataforma de análisis de datos, con el objetivo de que los usuarios pudieran visualizar y analizar los datos de manera sencilla y efectiva sin necesidad de conocimientos técnicos.

3.3.3 Fase de Desarrollo

Implementación de Funcionalidades

Esta fase fue el núcleo del proyecto, donde se desarrollaron las funcionalidades clave de la plataforma OTT. Cada funcionalidad se implementó de manera incremental, siguiendo un ciclo ágil de análisis, diseño, codificación y pruebas. Se utilizaron tecnologías web como JavaScript, HTML y CSS para asegurar la compatibilidad multiplataforma.

Para la plataforma de análisis de datos, se implementaron las funcionalidades necesarias para la carga, procesamiento y visualización de los datos, así como de análisis y generación de informes.

Optimización y Refactorización

A medida que se añadían nuevas funcionalidades, el código se refactorizaba y optimizaba para mejorar el rendimiento y la mantenibilidad. Este proceso incluyó la simplificación de la lógica de negocio, la mejora de la estructura del código, y la optimización de la carga y la respuesta de la aplicación.

3.3.4 Fase de Pruebas

Pruebas Unitarias y de Integración

Se realizaron pruebas unitarias para cada componente, asegurando que las funciones individuales funcionaran correctamente. Luego, se realizaron pruebas de integración para verificar que los componentes interactuaran de manera efectiva y sin conflictos.

Pruebas de Compatibilidad

Se llevaron a cabo pruebas exhaustivas en diferentes dispositivos y sistemas operativos (web, Tizen, WebOS, Android TV) para asegurar que la aplicación funcionara correctamente en todos los entornos previstos. Esto incluyó pruebas de rendimiento para garantizar que la aplicación se ejecutara sin problemas en dispositivos con capacidades de hardware limitadas.

3.3.5 Conclusión

Estas etapas se llevaron a cabo de manera iterativa y colaborativa, con reuniones periódicas con los clientes para validar el progreso y ajustar las prioridades según fuera necesario. Así para cada funcionalidad nueva se realizaba un ciclo de análisis, diseño, implementación y pruebas para asegurar que la funcionalidad cumpliera con los requisitos y expectativas del cliente.

Capítulo 4

Análisis

4.1 Introducción

El análisis es la primera fase del ciclo de vida de un proyecto de software. En esta fase se recopilan los requisitos del sistema, se identifican los objetivos y se establecen las restricciones que debe cumplir el sistema. El análisis es una fase crucial en el desarrollo de software, ya que de ella depende el éxito o fracaso del proyecto. En esta fase se establece la base sobre la que se construirá el sistema, por lo que es importante que se realice de forma correcta y exhaustiva.

En esta sección se describirá como se ha realizado el análisis del proyecto, los objetivos que se pretenden alcanzar y las restricciones que se han establecido.

4.2 Requisitos

En software, un requisito es una condición o capacidad que debe cumplir un sistema para resolver un problema o satisfacer una necesidad, incluyendo funciones, características y restricciones.

Identificar correctamente los requisitos es crucial para el éxito del proyecto, ya que orienta el desarrollo y asegura que el sistema cumpla con las especificaciones y expectativas del cliente. Existen dos tipos principales de requisitos: funcionales, que describen las funciones que debe realizar el sistema, y no funcionales, que detallan las características que el sistema debe tener.

Según la norma ISO 9001:2000, los requisitos se obtienen de varias fuentes, incluyendo las especificaciones del cliente, las necesidades implícitas para el uso previsto, los requisitos legales y los adicionales determinados por la organización. Todos los requisitos deben revisarse para asegurar su precisión y viabilidad a lo largo del proyecto.

4.2.1 Requisitos de la plataforma OTT

Los requisitos recogidos para la plataforma OTT se enfocan en la creación de una aplicación multiplataforma y multicliente, excluyendo componentes como la CDN o la gestión de usuarios y contenidos.

Requisitos funcionales

Requisitos especificados por el cliente Los requisitos funcionales abarcan tanto los generales como los específicos de cada cliente, por ejemplo:

- **Generales:** Pantallas para buscar contenido y configurar la aplicación.
- **CyLTv Play:** Pantallas para contenido en directo, a la carta y por delegaciones.
- **Oh!Jazz:** Inicio/cierre de sesión, restricciones para usuarios no registrados, y opción de cambio de idioma.

Requisitos no especificados por el cliente Requisitos necesarios para el correcto funcionamiento de la plataforma:

- La aplicación debe determinar y adaptar la interfaz según el cliente y la plataforma.
- La interfaz debe ser intuitiva, atractiva, adaptarse a distintas resoluciones y funcionar con mandos a distancia en TVs.

Requisitos no funcionales

Rendimiento: Mostrar contenidos rápidamente, sin cortes, y asegurar una experiencia fluida.

Seguridad: Garantizar la protección de datos de usuarios y clientes, y evitar accesos no autorizados.

Usabilidad: La aplicación debe ser intuitiva, visualmente atractiva y accesible para todos los usuarios.

Mantenimiento: La aplicación debe ser fácil de mantener, escalar y desplegar.

Escalabilidad: La aplicación debe adaptarse a aumentos en usuarios, contenidos y clientes.

Disponibilidad: La aplicación debe estar disponible siempre y poder recuperarse de fallos.

Compatibilidad: La aplicación debe ser compatible con todos los sistemas operativos, navegadores y dispositivos.

Internacionalización: La aplicación debe adaptarse a diferentes idiomas, culturas y zonas horarias.

Personalización: La aplicación debe permitir la personalización con logos y colores de cada cliente y plataforma.

Adaptabilidad: La aplicación debe ajustarse a distintas resoluciones, dispositivos y navegadores.

4.2.2 Requisitos de la aplicación de análisis de datos

La aplicación de análisis de datos tiene como objetivo proporcionar herramientas para la recopilación, visualización y análisis de datos de uso. A continuación, se describen los requisitos funcionales y no funcionales de la aplicación:

Requisitos funcionales

- Permitir la visualización de estadísticas de uso en tiempo real.
- Ofrecer filtros para datos por criterios como fecha, tipo de contenido o dispositivo.
- Generar análisis detallados de métricas como visualizaciones, reproducciones y usuarios únicos.
- Permitir una integración futura datos de diferentes fuentes y consolidarlos en un formato unificado.

Requisitos no funcionales

Rendimiento

- Procesar grandes volúmenes de datos de manera eficiente, manteniendo una respuesta rápida.

Seguridad

- Proteger los datos sensibles, garantizando su confidencialidad e integridad.

Usabilidad

- Ofrecer una interfaz intuitiva y accesible para la interpretación de datos.
- Proporcionar una experiencia de usuario fluida en todos los dispositivos soportados.

Mantenimiento

- Facilitar la actualización e integración de nuevas métricas y funcionalidades.
- Escalar la aplicación para manejar un aumento en la cantidad de datos.

4.3 Necesidades del cliente

Dado que esta aplicación está destinada a múltiples clientes, cada uno con sus propias necesidades y requisitos, el proyecto se ha centrado en desarrollar una solución única capaz de satisfacer estas diversas demandas.

Para cada requisito, se ha realizado un análisis detallado para determinar la mejor forma de implementarlo. El objetivo ha sido garantizar que, siempre que sea posible, la solución desarrollada pueda ser utilizada por todos los clientes de la plataforma. Esto ha implicado diseñar funcionalidades que sean lo suficientemente flexibles para adaptarse a diversas necesidades sin requerir modificaciones adicionales.

En los casos donde un requisito es particularmente específico o esencial para un cliente, se ha buscado una implementación que no interfiera con las funcionalidades del resto de los clientes. Esto se ha logrado mediante la creación de configuraciones modulares y opciones personalizables, que permiten a cada cliente ajustar la aplicación según sus propias necesidades sin afectar la experiencia de otros usuarios de la plataforma.

Este enfoque ha permitido mantener una base de código única y consistente, evitando la necesidad de desarrollar versiones separadas para cada cliente. Al mismo tiempo, se ha asegurado que cada cliente pueda personalizar su experiencia sin comprometer la funcionalidad general de la plataforma.

Un ejemplo es la interfaz de usuario, que permite a los clientes personalizar colores, logos e imágenes. Aunque la presentación general de los contenidos sigue un diseño común en el mercado, como listas horizontales de contenidos con imágenes y texto, la personalización se ha centrado en elementos clave como las listas de destacados. Estas listas, que muestran los contenidos más importantes, pueden ser configuradas por cada cliente para incluir o excluir elementos como botones de reproducción, títulos, subtítulos, duraciones, y fechas de publicación.

Para soportar esta personalización, se ha desarrollado un sistema de "etiquetado" que clasifica tanto los contenidos como las listas, permitiendo una visualización flexible según las

preferencias del cliente. Este enfoque permite a cada cliente diferenciar su interfaz, aunque implica un esfuerzo adicional en desarrollo y gestión de contenidos.

Actualmente, el proyecto se está orientando hacia una mayor personalización sin necesidad de crear nuevos tipos de elementos cada vez que se requiere una nueva configuración o presentación, mejorando así la flexibilidad de la plataforma.

4.4 Estudio de viabilidad

Desde el inicio, uno de los principales interrogantes fue si era factible desarrollar una aplicación única que pudiera generar versiones para distintos sistemas operativos y dispositivos. Este apartado analiza la viabilidad técnica y económica del proyecto, abordando las preguntas clave sobre la posibilidad de crear una solución híbrida entre dispositivos móviles y televisores, la existencia de tecnologías adecuadas y la eficiencia en su implementación.

4.4.1 Viabilidad técnica

La **viabilidad técnica** de un proyecto se refiere a la condición que hace posible llevar a cabo una idea o proyecto desde el punto de vista tecnológico. Para evaluar la viabilidad de este proyecto, es crucial determinar si existe una herramienta o tecnología que permita trabajar con un número suficiente de sistemas operativos y dispositivos para que el proyecto sea rentable. A continuación, se detalla este estudio.

En cuanto a infraestructura, la empresa ya dispone de los componentes necesarios y completamente funcionales para soportar la aplicación. Tanto el CMS, como la base de datos y los servidores de aplicaciones están operativos y son capaces de manejar la carga de trabajo que la aplicación requerirá, lo que hace que la viabilidad técnica en este aspecto sea positiva.

Adaptación a web

Las aplicaciones web ofrecen flexibilidad y portabilidad, permitiendo el uso de diversas tecnologías y lenguajes de programación. Este entorno facilita el desarrollo, ya que la aplicación puede aprovechar tecnologías comunes para adaptarse a otros dispositivos, sin que la viabilidad técnica sea un obstáculo significativo.

Adaptación a televisores

El caso de los televisores es muy distinto. Aunque los televisores inteligentes actuales cuentan con un navegador web, estos no son ni tan potentes ni tan cómodos de utilizar como los de los dispositivos móviles o los ordenadores, por lo que, aunque una persona pueda

acceder a la aplicación desde su televisor, no es la opción ideal para estos dispositivos. La opción que aplica para este proyecto es la de crear una aplicación nativa para televisores. Estas aplicaciones se ejecutan directamente en el sistema operativo del televisor, lo que requiere compatibilidad con la tecnología utilizada para desarrollar la aplicación.

Los sistemas operativos que se estudiaron en un primer momento para este proyecto fueron los más utilizados en televisores inteligentes: Android TV, Tizen, webOS y Apple TV. La primera opción fue utilizar tecnologías como JavaScript, HTML y CSS. Tras analizar Tizen y webOS, se confirmó que ambas opciones permiten el desarrollo con estas tecnologías, lo que hacía viable la implementación técnica. Sin embargo, Android TV y Apple TV no permiten el desarrollo con estas tecnologías de manera nativa, ya que utilizan sus propios entornos de desarrollo. Para superar esta limitación en Android TV, se optó por [Apache Cordova](#), una herramienta que permite crear aplicaciones híbridas usando HTML, CSS y JavaScript. En el caso de Apple TV, no se encontró una solución viable en esta etapa para adaptar el código a sus tecnologías propietarias.

Aunque no se encontró una solución para Apple TV, la capacidad de desarrollar para Android TV, junto con Tizen y webOS, cubre aproximadamente un 27% del mercado mundial de televisores inteligentes [12]. Aunque no es un porcentaje muy alto, es suficiente para considerar el proyecto como viable.

Con el proyecto ya en marcha y versiones funcionales de la aplicación disponibles, se están estudiando nuevas implementaciones para otros sistemas operativos y dispositivos. Entre las opciones futuras está Vidaa OS (Hisense), que actualmente posee el 7.8% del mercado mundial, posicionándose como el segundo sistema operativo más utilizado. Los primeros estudios sobre Vidaa OS han sido positivos, indicando que es compatible con las tecnologías elegidas, por lo que se planea continuar con el análisis y eventualmente integrar este sistema operativo.

En cuanto a Apple TV, se planifica desarrollarlo por separado y explorar cómo adaptar el código de la aplicación a su tecnología en el futuro, si es posible.

Adaptación a dispositivos móviles

Inicialmente, el enfoque no está en generar aplicaciones móviles nativas, sino en asegurar que la página web sea responsive para ofrecer una buena experiencia en dispositivos móviles. A medio plazo, se contempla la posibilidad de desarrollar para Android, con una probable exclusión inicial de iOS debido a las limitaciones tecnológicas.

Conclusión

El proyecto es técnicamente viable. A pesar de la limitación con Apple TV, el soporte para Android TV, Tizen y webOS, junto con la flexibilidad del desarrollo web, asegura que el proyecto puede avanzar con éxito.

4.4.2 Viabilidad económica

La viabilidad económica se centra en evaluar los costos y beneficios del proyecto para determinar su rentabilidad.

Costes

Los costes que conlleva el desarrollo de este proyecto son los siguientes:

- **Costes de desarrollo:** Incluyen el desarrollo de la aplicación, donde el único gasto significativo ha sido el del desarrollador, ya que muchas herramientas y sistemas de prueba son gratuitos o ya existentes. Otros gastos, como diseñadores o licencias, han sido cubiertos por los clientes.
- **Costes de mantenimiento:** La aplicación requiere un mantenimiento continuo, que incluye la solución de problemas y la incorporación de nuevas funcionalidades para adaptarse a nuevos clientes y plataformas. Estos costos se alinean con los costos de desarrollo.

Beneficios

Los ingresos provendrán de la creación y mantenimiento de la aplicación, con la posibilidad de cobrar por funcionalidades adicionales según lo requiera el cliente.

Conclusión

El proyecto promete ser económicamente viable gracias a la reducción de costos asociada a la reutilización del código, el tiempo de desarrollo optimizado y la menor necesidad de personal, lo que puede reflejarse en una rentabilidad favorable.

4.4.3 Conclusión final

En resumen, el proyecto es viable tanto técnica como económicamente. La adaptación a diferentes plataformas ha sido probada con éxito, y los costos previstos son manejables frente a los beneficios esperados, lo que justifica la continuación del desarrollo.

4.5 Estudio de las características de los distintos sistemas operativos, dispositivos y tecnologías

Cuando se realizó el primer estudio del proyecto y los distintos estudios de viabilidad, se analizaron las características de los distintos sistemas operativos, dispositivos y tecnologías

que se podían utilizar para el desarrollo de la aplicación. En este apartado se realizará un estudio de las características de estos elementos y cómo se integran para el desarrollo de la aplicación.

4.5.1 Sistemas operativos

Al inicio del proyecto, el objetivo principal era enfocarse en televisores, sin perder de vista otros dispositivos. Se tenía una idea clara de los sistemas operativos con los que empezar: Android TV, Tizen, webOS y Apple TV. Como se mencionó en la sección 4.4.1, Apple TV fue descartado, pero los otros tres sistemas operativos siguieron siendo viables. Se estudiaron las características de estos sistemas a través de la documentación oficial y la información disponible en la red. Las características tecnológicas básicas ya han sido comentadas en la sección de fundamentos tecnológicos 2.3.3. En este apartado se ampliará el análisis realizado para el proyecto.

Características de los sistemas operativos Las tecnologías seleccionadas para el desarrollo de la aplicación fueron JavaScript, HTML y CSS. El primer análisis fue el de la compatibilidad de estas tecnologías con los sistemas operativos. Tanto Tizen como webOS son compatibles con estas tecnologías, aunque algunas funcionalidades avanzadas aún no están disponibles de forma nativa. Android TV presentó un reto, ya que no es compatible de forma nativa con estas tecnologías, pero se encontró una solución mediante Apache Cordova (ver ??).

Empaquetado y distribución de aplicaciones Otro aspecto importante fue cómo empaquetar y distribuir las aplicaciones. Tizen y webOS son similares, aunque tienen formatos de empaquetado diferentes (wgt e ipk, respectivamente). Ambos requieren un archivo HTML principal, un archivo de configuración (config.xml y package.json), el código de la aplicación (JavaScript, HTML y CSS) y los recursos necesarios (imágenes, fuentes, etc.). Android TV utiliza la estructura de Cordova, que incluye los siguientes directorios:

- **www**: Contiene el código de la aplicación.
- **config.xml**: Archivo de configuración.
- **platforms**: Directorio con las plataformas en las que se ha empaquetado la aplicación.
- **plugins**: Contiene los plugins de Cordova utilizados.

Herramientas de desarrollo Se estudiaron las herramientas de desarrollo para cada sistema operativo. El desarrollo principal se realizó en Visual Studio Code, con plantillas específicas para cada SO. Las herramientas utilizadas fueron:

- **Tizen:** Tizen Studio de Samsung permite la gestión de dispositivos, emuladores y el empaquetado de aplicaciones.
- **webOS:** LG ofrece una CLI y una extensión para Visual Studio Code, facilitando el desarrollo, empaquetado y distribución.
- **Android TV:** Aunque se podría usar Android Studio, la adaptación con Cordova requirió utilizar comandos en el terminal para empaquetar e instalar las aplicaciones.

4.5.2 Dispositivos

Se analizaron dos aspectos principales: la capacidad de los dispositivos para ejecutar la aplicación y la resolución de pantalla.

El estudio de resoluciones mostró la necesidad de que la aplicación fuera adaptativa, capaz de ajustarse a cualquier resolución de pantalla. Se utilizó un diseño responsive, empleando unidades relativas (vh, vw, porcentajes) en lugar de px, lo que permitió que la aplicación se adaptara a diferentes dispositivos y resoluciones, como 1920x1080 (Samsung y LG) o 920x540 (TCL Android TV).

El rendimiento varía entre dispositivos. Las pruebas iniciales revelaron que, en televisores con menos capacidad, los movimientos resultaban lentos. Para mejorar la eficiencia, se optimizaron funciones como el uso de `transform: translate()` en lugar de `top` o `left`, y la utilización de `will-change` para mejorar el renderizado.

4.5.3 Tecnologías

El estudio de las tecnologías fue un proceso constante durante el desarrollo del proyecto. La elección de las mejores funciones de JavaScript, el análisis de bibliotecas de React y el uso eficiente de HTML y CSS fueron clave para mejorar tanto la eficiencia como la presentación de la interfaz.

En cuanto a la aplicación de análisis de datos, se estudiaron diversas librerías disponibles en React, lo que permitió mejorar exponencialmente el desarrollo y rendimiento de la plataforma de análisis. La combinación de estas tecnologías permitió optimizar la calidad y la eficiencia de la aplicación OTT y su plataforma analítica complementaria.

Capítulo 5

Diseño

5.1 Introducción

Una vez completada una fase de análisis y claros los requisitos y objetivos del proyecto, vamos a trabajar sobre estos para definir las estructuras de datos, las funciones y los comportamientos del sistema. Se trata de una fase previa a la implementación, en la que se definen los elementos que compondrán el sistema y cómo se relacionan entre sí.

En este apartado se describirá como fueron las primeras fases de diseño de las aplicaciones *Ott* y *Ott Data* y como se trabaja en general en las fases de diseño de nuevas funcionalidades e iteracciones en el proyecto.

5.2 Diseño de la aplicación OTT

5.2.1 Introducción al diseño de la aplicación *ott*

En este capítulo se describirá el diseño de la aplicación *OTT*. Se comenzará con una descripción de la arquitectura del sistema, seguido de la interfaz de usuario y los diagramas UML utilizados en el desarrollo de la aplicación.

El proceso de diseño de la aplicación *OTT* ha sido particular, ya que en un principio se trabajó sobre diseños de interfaz ya utilizados en otras plataformas para otros clientes. El objetivo inicial era crear una base sólida sobre la que desarrollar las funcionalidades, sabiendo de antemano que este no sería el diseño final. Posteriormente, una vez que la aplicación estaba en una fase avanzada con una gran cantidad de funcionalidades implementadas, comenzaron las reuniones con los distintos clientes. En estas reuniones, además de recoger sus impresiones y sugerencias, se estudiaron nuevas funcionalidades a implementar, y se ajustaron tanto la interfaz como la experiencia de usuario para adaptarlas a las preferencias de cada cliente.

Durante el desarrollo de la aplicación, se han modificado y añadido funcionalidades conforme a las necesidades que fueron surgiendo. Gracias a la flexibilidad de la metodología ágil,

fue posible adaptar la aplicación a estas nuevas funcionalidades, así como a las demandas de los clientes y las nuevas tendencias del mercado. Cada vez que se completaba una funcionalidad o una iteración, se llevaban a cabo reuniones internas o con los clientes para validar el trabajo y planificar las siguientes etapas. Una vez definido el objetivo de la nueva iteración, se iniciaba el ciclo de análisis, diseño, implementación y pruebas, lo que permitía una adaptación continua y mayor flexibilidad en el desarrollo del proyecto.

5.2.2 Diseño de la arquitectura del sistema

Como ya se ha mencionado en la sección 2.2.2, la arquitectura de esta aplicación está diseñada para integrarse en la arquitectura general de la empresa y se basa en una arquitectura de microservicios.

Arquitectura basada en microservicios

La arquitectura basada en microservicios es un enfoque de diseño que organiza una aplicación en un conjunto de pequeños servicios independientes, cada uno especializado en una funcionalidad específica y ejecutado en su propio proceso. Estos servicios se comunican a través de mecanismos ligeros como APIs REST [6].

Cada microservicio está desacoplado del resto, lo que permite que se desarrolle, desplíguen y escalen de manera independiente. Este enfoque facilita la evolución de la plataforma, ya que cada microservicio puede ser mejorado sin afectar al resto de los componentes. Además, la arquitectura proporciona una mayor tolerancia a fallos, ya que un error en un microservicio no debería afectar a toda la plataforma. Este diseño modular también permite la reutilización de microservicios en otros proyectos, simplificando la creación de nuevas aplicaciones y su integración con otros sistemas.

En el caso de la plataforma OTT, se han utilizado los siguientes microservicios creados por la empresa:

- **IDEN - Identificación de usuarios:** encargado de la gestión de los usuarios de la plataforma, incluyendo el registro, autenticación y autorización, así como la gestión de perfiles, intereses y preferencias.
- **Directus - CMS:** responsable de la gestión y almacenamiento de contenidos, metadatos y archivos multimedia.
- **CAS - Servicio de acceso condicional:** encargado de gestionar accesos condicionales, incluyendo licencias, DRM, cifrado y protección de contenidos.
- **Orquestador:** encargado de realizar las comprobaciones y procesos que requieren la interacción de múltiples microservicios.

- **Importer:** utilizado cuando el cliente necesita importar contenido desde su base de datos externa a la plataforma interna de la OTT.
- **Player:** encargado de proporcionar la URL de reproducción de vídeo. Aunque no se utiliza el reproductor nativo debido a las limitaciones del backend, este microservicio genera una URL para reproducir el contenido en cada SO.

Estos microservicios se comunican entre sí a través de una API REST, enviando y recibiendo datos en formato JSON. El frontend interactúa continuamente con el backend mediante esta API, solicitando y enviando datos a las rutas definidas para cada microservicio. Este modelo de comunicación permite una integración fluida entre los distintos componentes del sistema.

Objetivos de la arquitectura

La arquitectura utilizada en la aplicación está diseñada para ser escalable, flexible y fácil de mantener. Uno de los pilares fundamentales es el uso de microservicios, que permite aislar las funcionalidades de la aplicación y desarrollar cada una de ellas de forma independiente. No obstante, la correcta utilización de los microservicios y la forma en que se comunican entre ellos es clave para garantizar que la aplicación mantenga estos principios. A continuación, se detallan los principales objetivos de la arquitectura:

- **Escalabilidad:** Desde el diseño, la escalabilidad fue un objetivo prioritario. La arquitectura permite un crecimiento tanto en la capacidad de usuarios como en la adición de nuevas funcionalidades, mediante el uso de un diseño modular y el desacoplamiento de componentes. Esto facilita que cada parte del sistema pueda escalar horizontalmente (añadiendo más instancias) o verticalmente (optimizando recursos) según sea necesario.
- **Flexibilidad:** La plataforma está diseñada para adaptarse a las necesidades específicas de cada cliente, permitiendo personalizar la interfaz y ajustar funcionalidades sin modificar el código base. La flexibilidad se logra mediante una arquitectura basada en componentes, que permite activar o desactivar módulos según los requisitos del cliente. Además, se garantiza la interoperabilidad con otros sistemas y servicios, facilitando la integración con APIs de terceros.
- **Interoperabilidad:** Se priorizó la capacidad del sistema para integrarse eficazmente con otros sistemas y servicios externos, utilizando estándares de la industria que aseguren una integración sin problemas. Esto permite que la plataforma sea compatible con una amplia variedad de servicios, desde APIs hasta sistemas de gestión de contenido y análisis de datos, facilitando su futura expansión.

- **Adaptación Multiplataforma:** Se diseñó la arquitectura para asegurar que la plataforma OTT pueda adaptarse a una gran variedad de dispositivos y sistemas operativos, como navegadores web, dispositivos móviles y smart TVs. El diseño responsive y la reutilización de gran parte del código base aseguran una experiencia de usuario coherente en todos los dispositivos.
- **Mantenibilidad:** La arquitectura modular facilita que los componentes se actualicen o reemplacen sin afectar al resto del sistema. Las mejores prácticas en diseño y la automatización de pruebas y despliegues aseguran que el sistema sea fácil de mantener y actualizar a lo largo del tiempo.
- **Rendimiento y Optimización:** La plataforma está diseñada para manejar grandes volúmenes de tráfico y datos sin degradar la experiencia del usuario. Se han implementado estrategias como la optimización de la comunicación entre componentes, el uso de caché y la distribución de la carga de trabajo mediante CDNs para garantizar un rendimiento óptimo.
- **Experiencia de Usuario (UX):** El diseño de la plataforma se centra en ofrecer una experiencia de usuario excepcional, con una interfaz intuitiva, accesible y coherente en todos los dispositivos. Se realizaron iteraciones de pruebas de usabilidad para garantizar una interfaz funcional y agradable, que pueda personalizarse según las preferencias individuales de los usuarios.

5.2.3 Diseño de la interfaz de usuario

La interfaz de usuario es la parte de la aplicación que permite la interacción con el sistema. En *Ott*, se encarga de mostrar la información al usuario y permitirle navegar y utilizar las funcionalidades. Es esencial que la interfaz sea clara, sencilla e intuitiva, para garantizar una experiencia de uso eficiente.

El diseño inicial no partió de un esquema definitivo, ya que debía adaptarse a las necesidades de los clientes. Se desarrolló una interfaz base que facilitara la incorporación de nuevas funcionalidades y la personalización según los requisitos de cada cliente.

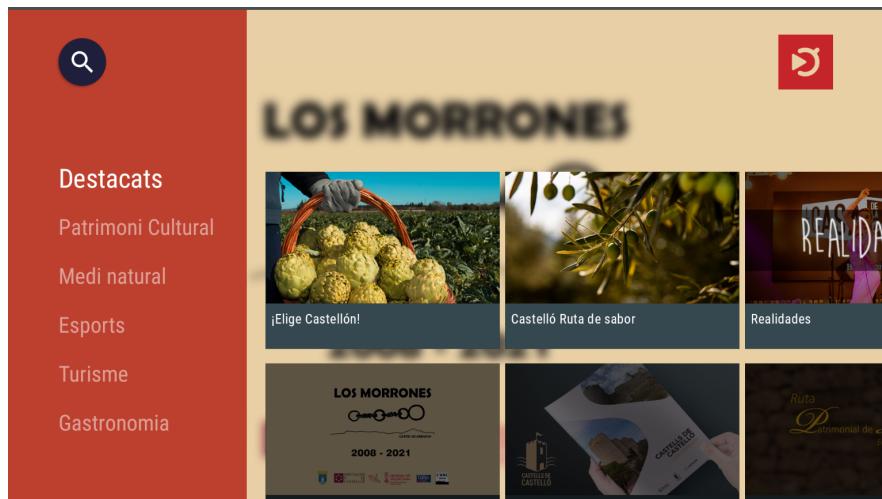


Figura 5.1: Captura de la interfaz en las primeras fases del proyecto

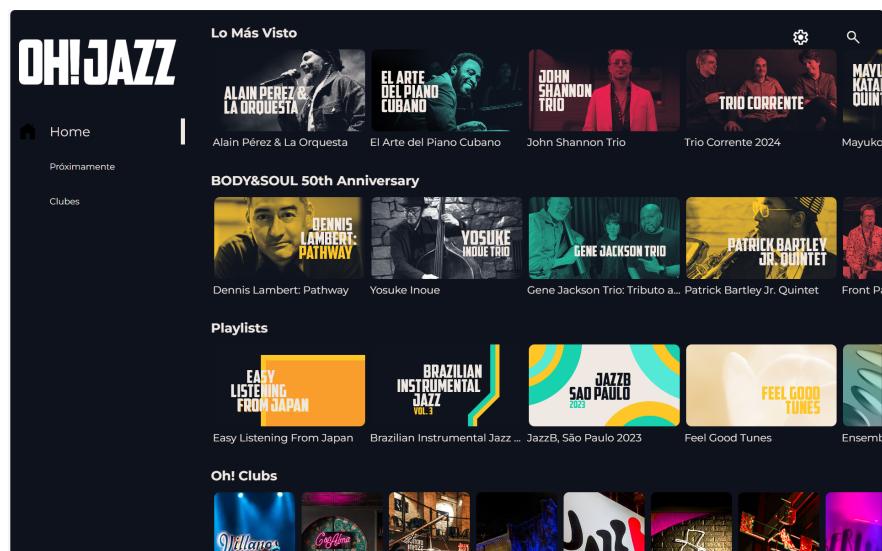


Figura 5.2: Captura de la interfaz más reciente, aún en pruebas por parte del cliente

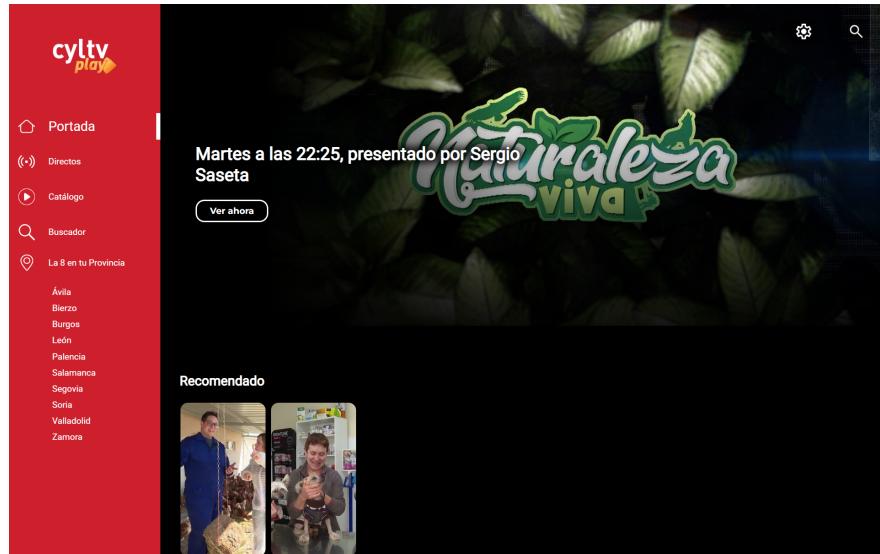


Figura 5.3: Captura de la interfaz casi final

Estas imágenes muestran la evolución de la interfaz de usuario de *Ott*. La primera imagen refleja el diseño básico inicial, sin apenas funcionalidades. En la segunda, se observa un diseño más avanzado, aunque el cliente aún estaba ajustando parámetros (por ejemplo, la falta de visibilidad de los iconos de los menús o el contraste del fondo). Finalmente, la tercera imagen muestra la interfaz casi final, con un diseño pulido y todas las funcionalidades implementadas.

Más que enfocarse en un diseño único, el proyecto se ha centrado en el diseño de componentes y sus variantes. Algunos elementos de la interfaz tienen una estructura fija, aunque permiten personalizar colores y textos:

- **Menú lateral:** Permite navegar por las secciones de la aplicación y se encuentra a la izquierda de la pantalla.
- **Menú superior:** Proporciona acceso a las opciones de configuración. Los iconos y colores de fondo son personalizables, pero la posición depende del número de opciones del menú.
- **Barra de búsqueda:** Situada en la pantalla de búsqueda, su posición y tamaño son fijos, pero utiliza los colores base del cliente para los efectos de foco y difuminado.
- **Lista de elementos:** Muestra los elementos disponibles en la aplicación. La estructura y tamaños son fijos, aunque el contenido y número de listas pueden personalizarse.
- **Detalle de elemento:** Muestra la información detallada de un elemento. Aunque la metadata y el contenido son personalizables, la estructura y disposición de los elemen-

tos son fijos. Se está trabajando para permitir mayor personalización desde el gestor de contenidos.

El cliente puede personalizar la interfaz ajustando colores de fondo, texto, botones, foco y añadiendo sus logos e imágenes, adaptando la aplicación a su identidad visual.

Personalización de los componentes

Como se ha mencionado anteriormente, el diseño de la interfaz de usuario se ha centrado en la creación de componentes y sus múltiples variantes. Estos componentes se dividen en dos tipos principales: *contenidos* y *contenedores*, cada uno de los cuales tiene un enfoque de diseño diferente dependiendo del tipo de *widget* en el que se utilicen. Un *widget* es un componente de la interfaz que almacena y muestra uno o varios contenidos de manera específica.

Cada tipo de *widget* determina la presentación tanto de los contenidos como de los contenedores, lo que implica que se necesita un diseño y un código específico para asegurar que se muestren correctamente en la interfaz. A continuación, se detallan los aspectos clave que cambian según el tipo de *widget*:

- **Disposición de los contenidos:** Según el tipo de *widget*, los contenidos se distribuyen de formas diversas. Si se trata de un *widget* de **destacados**, los contenidos se mostrarán con un tamaño más grande y ocuparán la parte superior de la pantalla, siendo más llamativos. Estos *widgets* pueden tener distintas configuraciones, como la inclusión de botones, un único contenido destacado o varios en formato *slider*, con opciones de distribución de texto e iconos. En contraste, un **mosaico** organizará los contenidos de manera compacta en el centro de la pantalla. Para otros tipos de *widgets*, como las listas horizontales, los contenidos se dispondrán en filas secuenciales. Estos formatos son personalizables según las necesidades del cliente y el tipo de contenido que se deseé resaltar.
- **Disposición de la información:** La cantidad y el tipo de información que se muestra también dependen del *widget*. Por ejemplo, en un **destacado**, el sistema puede mostrar el título, una descripción detallada y un botón de acceso al contenido. En un **banner**, solo se mostrará el título del contenido. Esta flexibilidad en la información permite que los *widgets* se ajusten a diferentes escenarios y preferencias del cliente, lo que es clave para mantener una interfaz adaptable y eficaz.
- **Forma de las imágenes:** La forma y el tamaño de las imágenes asociadas a los contenidos varían según el tipo de *widget*. En los **destacados**, las imágenes suelen ser grandes, ocupando una buena parte de la pantalla, y pueden cambiar la distribución de la información según el caso específico. En los **banners**, las imágenes tienen una orientación

horizontal con un ratio de 16:9, mientras que en los **posters**, las imágenes son verticales, con un ratio de 9:16. En los **posters**, las imágenes aparecen en un panel lateral cuando el foco se mantiene en el contenido durante más de un segundo. Este enfoque garantiza una presentación visual coherente que varía según el tipo de contenido y el diseño del *widget*.

- **Otras características:** Algunos *widgets* permiten funcionalidades adicionales que dependen del tipo de contenido que se esté presentando. Por ejemplo, en los *widgets* para contenidos en directo, puede aparecer una barra de progreso y el nombre del programa actual. En un **widget banner-click**, diseñado para mostrar anuncios, se presenta un banner horizontal que ocupa casi todo el ancho de la pantalla, y al hacer clic en él (excepto en televisores, donde esta opción está desactivada), redirige al usuario a una URL externa.

Estas personalizaciones, junto con otras más específicas o en desarrollo, deben estar correctamente implementadas y soportadas por la aplicación para garantizar que los *widgets* seleccionados por el cliente funcionen sin problemas y no interfieran con el resto de la interfaz.

Adicionalmente, es importante tener en cuenta ciertas limitaciones en la personalización. Solo puede haber un *widget destacado* por pantalla, los *widgets* tipo **mosaico** con muchos elementos pueden sobrecargar la interfaz, y los **posters** se utilizan generalmente para destacar ciertos contenidos, mostrando información detallada sobre estos.

Ejemplos de personalización

A continuación se muestran algunos ejemplos de personalización de los widgets de la interfaz de usuario de *Ott*.



(a) Ejemplo de un widget básico de tipo "banner"

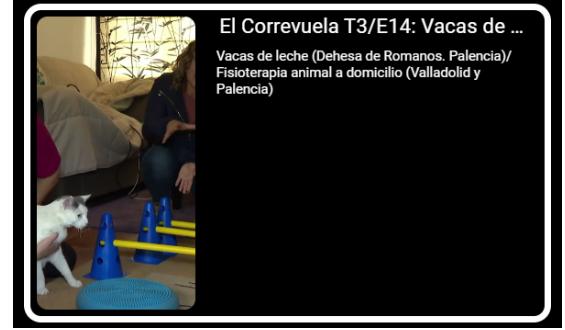


(b) Ejemplo de un widget básico de tipo "destacado"

Figura 5.4: Ejemplos de un widget básico de tipo "banner" y "destacado"



(a) Ejemplo de un widget básico de tipo "poster"



(b) Ejemplo de un widget básico de tipo "poster" con la información mostrada

Figura 5.5: Ejemplos de un widget básico de tipo "poster"

5.3 Diseño de la aplicación de análisis de datos

5.3.1 Introducción al diseño de la aplicación de análisis de datos

En esta sección se presenta el diseño de la aplicación de análisis de datos. Se describen los diferentes componentes que conforman la aplicación y cómo se relacionan entre sí. Además, se explica cómo se ha estructurado el código para facilitar su mantenimiento y escalabilidad.

5.3.2 Estudio y diseño de la aplicación de análisis de datos

El diseño de la aplicación de análisis de datos se ha realizado siguiendo una metodología centrada en el usuario. Para ello, se llevó a cabo un análisis de las necesidades de los usuarios, se definieron los requisitos de la aplicación y se diseñó la interfaz de usuario.

Entre las principales necesidades y requisitos de la aplicación destaca la importancia de que la visualización de los datos sea clara y sencilla, evitando la necesidad de conocimientos avanzados en análisis de datos para su uso.

El diseño de la interfaz de usuario sigue un enfoque sencillo y minimalista, con colores suaves y tipografía clara y legible. Se ha diseñado para ser intuitiva y fácil de usar, permitiendo al usuario acceder rápidamente a todas las funcionalidades. El foco está en la visualización de gráficos y datos, presentando la información de forma ordenada y consistente para facilitar su comprensión. Todas las gráficas y tablas comparten un estilo uniforme y limpio, evitando elementos visualmente recargados.

De forma similar a la OTT, este diseño sigue un enfoque modular, dividiendo la aplicación en componentes independientes que se comunican entre sí. Esto facilita el mantenimiento y la extensibilidad de la aplicación, ya que cada componente puede ser modificado o reemplazado sin afectar al resto. Además, el diseño modular permite la reutilización de código y la

integración de nuevas funcionalidades.

Estudio de las agrupaciones de datos

Una vez definidos los requisitos de la aplicación y el diseño general, era importante estudiar qué agrupaciones de datos se podían lograr con las funcionalidades que ofrece Matomo y qué opciones de visualización se podían implementar.

Lo primero fue identificar qué tipos de gráficos eran necesarios soportar y diseñar. Tras analizar la entrega de datos que realiza Matomo, se decidió comenzar con el soporte para gráficos lineales, de barras, circulares y tablas. De esta manera, en función de la utilidad, periodo y enfoque de los datos, se podrán seleccionar diferentes tipos de gráficos para visualizar la información.

Páginas y secciones Posteriormente, se estudió qué páginas o apartados de la aplicación de análisis de datos podían implementarse. La primera página en la que se pensó fue el dashboard o página de inicio, donde se mostrarán gráficos que presenten información más general y resumida de los datos.

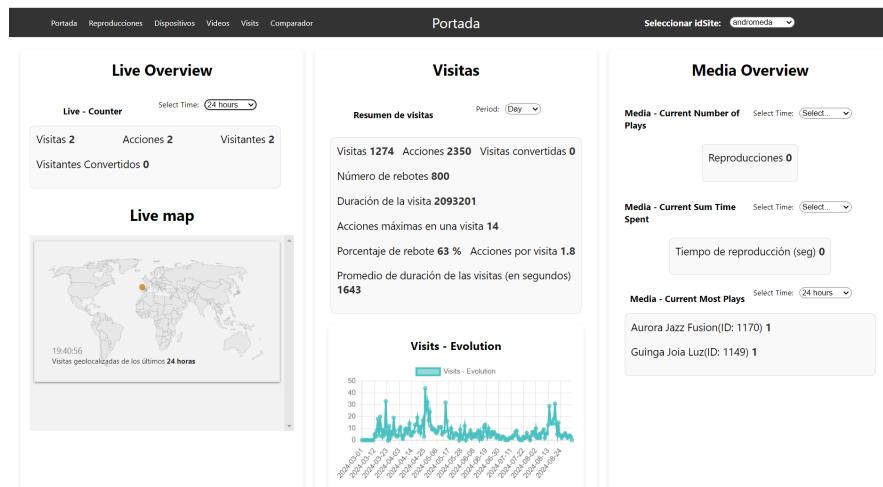


Figura 5.6: Página de inicio de la aplicación de análisis de datos

Otra página considerada durante el análisis fue una que permitiera al usuario comparar gráficas de diferentes módulos. La idea era ofrecer al cliente la posibilidad de visualizar varias gráficas en una misma página, facilitando así el análisis de información en busca de patrones, tendencias o correlaciones entre los datos.

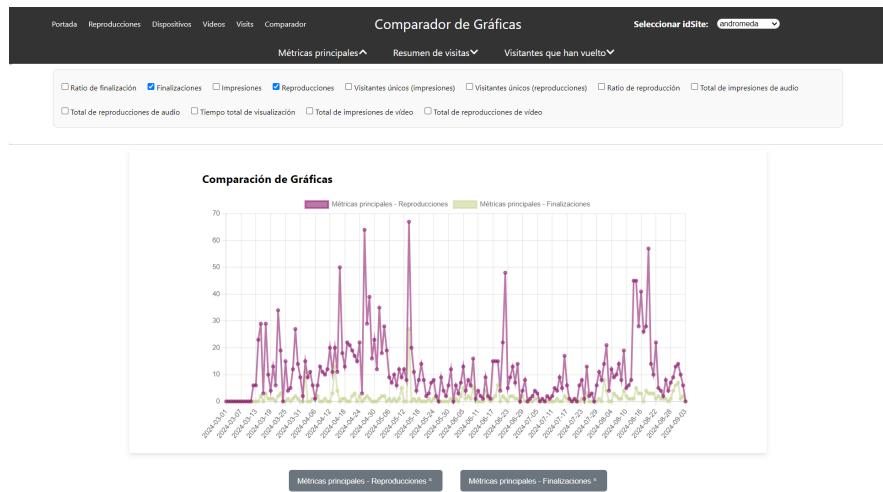


Figura 5.7: Página de comparador de la aplicación de análisis de datos

El resto de las páginas se dedican a agrupaciones de datos que tienen relación entre sí. La creación de estas páginas se diseñó de manera dinámica, permitiendo la creación rápida y sencilla de nuevas secciones. Estas páginas están enfocadas en mostrar datos de los visitantes desde diferentes perspectivas. Para una OTT de estas características, pensada para su uso en diversas plataformas y dispositivos, una sección relevante es la de dispositivos, que ofrece información sobre los sistemas operativos, marcas y modelos desde los cuales se accede. Otro punto clave son las reproducciones, donde se muestra información sobre la visualización de contenidos, como el número de reproducciones, tiempo de visualización y los videos más vistos.

Con los enfoques y datos claramente definidos, se procedió al diseño de las páginas y secciones, centrándose en las funcionalidades de la API de Matomo que permiten obtener los datos necesarios. Tras analizar las funcionalidades disponibles, y considerando cuáles ya estaban implementadas y optimizadas en los sistemas de la empresa, se desarrollaron las pantallas de reproducciones, dispositivos, videos, visitas, además del dashboard de inicio y el comparador.

- **Inicio:** Esta página presenta los gráficos más generales y resumidos de los datos, como el número de visitas, reproducciones y dispositivos en tiempo real. Está diseñada para ofrecer al usuario una visión global de los datos de un solo vistazo.
- **Reproducciones:** Aquí se muestran gráficos relacionados con las reproducciones de videos, incluyendo el número de reproducciones, tiempo medio de visualización y visitantes únicos. El objetivo es que el usuario pueda analizar los patrones de visualización y obtener insights sobre el comportamiento de los usuarios.

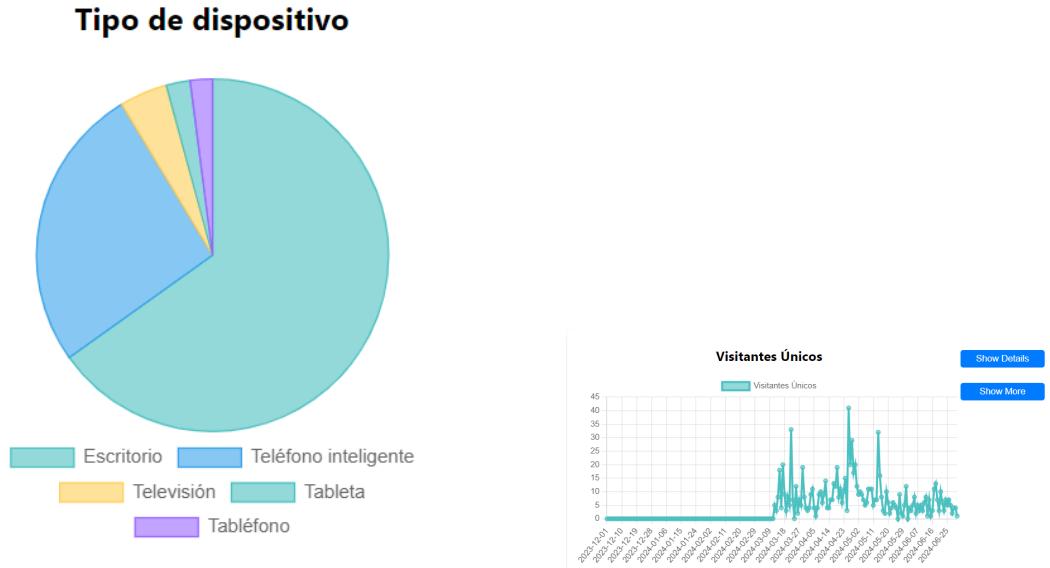


Figura 5.8: Ejemplo de grafico circular mostrando la distribución de los tipos de dispositivos en los que se accede a la web



Figura 5.9: Ejemplo de gráfico lineal mostrando la evolución del número de visitantes únicos en la web

- **Dispositivos:** En esta página se despliegan gráficos relacionados con los dispositivos desde los que se accede a la web, como sistemas operativos, marcas y modelos. Esto permite al usuario analizar las preferencias de los usuarios en cuanto a dispositivos.
- **Videos:** Esta página incluye una lista de los contenidos de la plataforma con información detallada de cada uno, como número de reproducciones, tiempo medio de visualización y tasa de finalización. Además, la tabla permite ordenar los datos por diferentes campos, facilitando el análisis.
- **Visitas:** Esta sección está compuesta por varias páginas que detallan la información sobre las visitas a la web. Actualmente incluye páginas dedicadas a resumen de visitas, tiempo de navegación, frecuencia de visitas e interés de los visitantes.
- **Comparador:** En esta página el usuario puede visualizar y comparar gráficos de diferentes módulos para analizar correlaciones, tendencias o patrones entre los datos. Esto facilita la obtención de insights sobre la relación entre diferentes variables y módulos, ayudando a tomar decisiones informadas.

Funcionalidades de la aplicación

Además de la visualización de gráficos y tablas, la aplicación incluye una funcionalidad que permite al usuario obtener explicaciones sobre los datos que se están mostrando. Esta

opción ofrece tanto una breve descripción como un análisis más detallado de los datos a través de la API de OpenAI.

Para generar este análisis, se utiliza un contexto general de la sección de la aplicación desde la cual se visualizan los datos, complementado con la información de la gráfica específica. Este contexto, que incluye metadatos relacionados con los usuarios y las interacciones dentro de la OTT, se almacena en MongoDB. Así, a medida que se va utilizando la plataforma, este contexto se enriquece y se optimiza para proporcionar análisis cada vez más precisos y detallados

Capítulo 6

Implementación

6.1 Introducción a la implementación

En este capítulo se describirá la implementación de la solución propuesta en el capítulo anterior. Se explicará cómo se ha llevado a cabo la implementación de ambas aplicaciones. Se describirá como se han implementado las funcionalidades básicas de ambos proyectos. Se detallará cómo funcionan y como se han utilizado las tecnologías y herramientas necesarias para llevar a cabo la implementación de ambos proyectos.

6.2 Implementación de la aplicación OTT

6.2.1 Desarrollo de la funcionalidades básicas

En esta sección se describe el proceso de implementación de las funcionalidades básicas de la aplicación OTT. Como se indicó en secciones anteriores, las tecnologías principales utilizadas para el desarrollo son JavaScript, HTML y CSS, que permiten crear una interfaz interactiva, dinámica y adaptable a diferentes dispositivos y plataformas.

Indicaciones previas

Esta sección describe el comportamiento básico de la aplicación cuando se utiliza en televisores, ya que este es el enfoque principal y ha supuesto los mayores desafíos durante el desarrollo del código.

El desarrollo para televisores presenta una particularidad: la aplicación debe estar diseñada para ser controlada con un mando a distancia. Aunque algunas televisiones ofrecen la opción de utilizar un "magic remote", un mando que incluye un puntero que se mueve por la pantalla, muchos modelos no cuentan con esta funcionalidad. Desarrollar la aplicación solo para este tipo de dispositivos limitaría significativamente el mercado. Por ello, la aplicación

ha sido diseñada desde un enfoque que permite su uso con un mando tradicional.

Aun así, también se ha contemplado la posibilidad de que la aplicación sea compatible con un puntero o ratón, como en el caso de los ordenadores, dado que, aunque este no fue un objetivo en las primeras versiones, uno de los principales propósitos del código es ser transversal y adaptable a cualquier dispositivo.

Creación de los componentes de la aplicación

El primer paso del desarrollo fue la creación de la estructura básica de la aplicación y de los primeros componentes, sobre los cuales se ejecutan las funcionalidades principales. Estos componentes incluyen los menús, los "widgets" y los contenidos.

La creación de estos componentes sigue un enfoque en cascada. Cuando se inicia la aplicación, se realiza una llamada a la API de la empresa para obtener un archivo JSON conocido como "interfaz". Este JSON contiene la información principal para configurar la aplicación, incluyendo los colores (fondo, botones, textos), fuentes, textos legales, imágenes (logos, splash, etc.), y otros elementos utilizados para la personalización del cliente. Además, este archivo también proporciona la información sobre los menús que estarán disponibles en la aplicación.

Para la creación de los menús, existen dos opciones: menús con una pantalla asignada o menús con un comportamiento predefinido. Los menús con comportamiento predefinido incluyen el menú de configuración, búsqueda y catálogo, que no requieren información específica sobre la pantalla que deben mostrar.

- **Configuración:** Este menú aloja las distintas opciones de configuración de la aplicación. Según las características de la aplicación, se mostrarán diferentes opciones. Por ejemplo, si la aplicación admite múltiples idiomas, en este menú se podrá cambiar el idioma. Si hay usuarios registrados, se mostrará la información del usuario y se podrá cerrar sesión. Una funcionalidad común a todas las aplicaciones es la opción de mostrar los términos y condiciones.
- **Búsqueda:** Este menú permite filtrar todos los contenidos de la aplicación por el nombre del contenido, proporcionando una funcionalidad de búsqueda.
- **Catálogo:** También conocido como "A la carta", muestra en una sola página todos los contenidos disponibles en la aplicación.

En el caso de los menús que tienen una pantalla asignada, esta contiene la lista de "widgets" que deben mostrarse. Los "widgets" son contenedores de elementos, y según su tipo, tendrán un diseño y unas funcionalidades asignadas. Además, para cada widget existirá una lista de contenidos junto con la información de cada uno de ellos.

Por lo tanto, la creación de las pantallas principales se realiza de la siguiente manera:

- Se obtiene la información del menú a través de la API.
- Se analiza para determinar si es un menú predefinido o si tiene una pantalla asignada.
 - Si es un menú predefinido, se crea la pantalla según el comportamiento predefinido.
 - Si tiene una pantalla asignada, se obtiene la información de la pantalla.
 - * Se crea el elemento del DOM que contendrá la pantalla.
 - * A partir de la lista de widgets, se crea un elemento del DOM para cada uno de ellos.
 - * Se obtiene la información de los contenidos de cada widget.
 - * Se crea cada uno de los contenidos con las características correspondientes, en función del tipo de widget que los contiene.
 - * Se añade cada contenido al widget correspondiente en el orden adecuado.
 - * Se añade el widget al elemento del DOM que contiene la pantalla.

Durante la creación de cada elemento HTML, se le asignan una serie de clases según el tipo de elemento y widget correspondiente. Estas clases permiten aplicar el estilo a través de las hojas de estilo CSS.

El tipo de widget también determina el comportamiento de los contenidos que contiene. Por ejemplo, si el widget es de tipo *featured* con el campo *slider* activado, los contenidos destacados se mostrarán en un carrusel. Si el widget es de tipo *mosaico*, los contenidos se mostrarán en una cuadrícula, y el movimiento, en lugar de ser una lista que se desplaza horizontalmente, será una cuadrícula que se mueve en ambas direcciones.

Movimiento por la aplicación

Una vez creados los elementos de la aplicación, el siguiente paso es habilitar la navegación dentro de ella. Para ello, se ha creado una serie de funciones que permiten moverse por los distintos elementos de la aplicación. Se ha tenido en cuenta que la aplicación será utilizada con un mando a distancia, por lo que el movimiento debe ser gestionado y controlado por el código.

Al utilizar la aplicación en televisores, es fundamental que exista un foco en un elemento en todo momento. Este foco indica en qué parte de la aplicación se encuentra el usuario y permite la navegación. En las pantallas principales de esta aplicación, se ha utilizado un foco fijo, es decir, el foco permanece en la misma posición casi siempre y los elementos se mueven en función de la dirección en la que el usuario navegue.

El movimiento a través de los widgets se realiza de la siguiente manera:

- **Movimiento horizontal:** Utilizando las teclas de dirección izquierda y derecha. Si el usuario se encuentra en un widget, el movimiento se realizará sobre los contenidos de dicho widget. Al pulsar una tecla de dirección, el widget se desplaza horizontalmente en la dirección indicada, colocando el elemento seleccionado en la posición del foco.
- **Movimiento vertical:** Se realiza con las teclas de dirección arriba y abajo. Si existen widgets adicionales en la dirección pulsada, la pantalla se desplazará hacia arriba o hacia abajo hasta colocar el elemento seleccionado en la posición del foco.
- **Movimiento en carrusel:** En caso de que el widget sea de tipo *featured* con el campo *slider* activado, el movimiento se realiza dentro del carrusel de contenidos destacados. El desplazamiento horizontal cambia la imagen y la información mostrada en el carrusel.
- **Movimiento en mosaico:** Para los widgets de tipo *mosaico*, el movimiento se realiza dentro de una cuadrícula de contenidos. Los elementos se ordenan en filas de izquierda a derecha y de arriba hacia abajo. El movimiento vertical desplaza la pantalla hacia arriba o hacia abajo, mientras que el horizontal mueve el foco hacia el contenido seleccionado.

El movimiento por el menú lateral es más sencillo. Se puede acceder al menú pulsando el botón *return* desde cualquier pantalla principal, o bien pulsando la tecla de dirección izquierda si no quedan más elementos a la izquierda. Una vez seleccionado el menú, este se despliega, mostrando tanto los iconos como los nombres de los menús. El usuario puede moverse por el menú con las teclas de dirección arriba y abajo, destacando el elemento seleccionado con un aumento de tamaño.

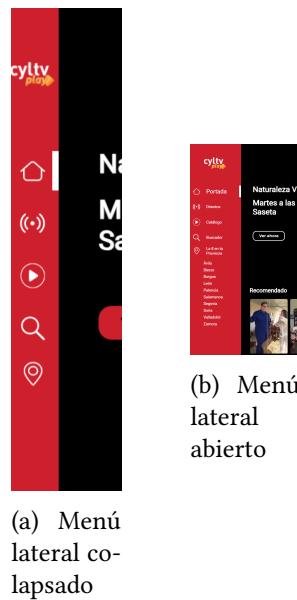


Figura 6.1: Menú lateral

Durante el desarrollo del proyecto se probaron diferentes maneras de implementar el movimiento en la interfaz. Inicialmente, se comenzó moviendo los elementos hacia el foco y, más tarde, se decidió utilizar las funcionalidades de *scroll* de HTML, que permite desplazar elementos sin realizar cálculos de posición. Sin embargo, ambas opciones fueron finalmente descartadas en favor de la utilización de las funcionalidades de *transform* de CSS. Tras analizar las tres alternativas, se concluyó que el uso de *transform* era prácticamente obligatorio. Aunque la opción de *scroll* es más sencilla y presenta múltiples ventajas, su uso resulta menos eficiente, exigiendo un mayor esfuerzo al navegador. En el caso de ordenadores y dispositivos móviles, este esfuerzo no es tan significativo, pero en el caso de las televisiones, que cuentan con menos recursos, la aplicación se percibía lenta y pesada. La primera opción fue descartada porque es más efectivo y eficiente mover la pantalla completa en lugar de reordenar cada elemento por separado.

En futuras implementaciones, y con una optimización más avanzada de la aplicación, no se descarta estudiar nuevamente la funcionalidad de *scroll* debido a sus ventajas. Sin embargo, por el momento, se seguirá utilizando la funcionalidad de *transform* de CSS. Aunque el cambio no es urgente ni necesario, ya que para el usuario no supone una diferencia significativa en cuanto a la experiencia de uso, sí mejora el rendimiento general de la aplicación.

Selección de un componente

Una vez que se ha movido el foco al elemento deseado, el siguiente paso es seleccionar dicho elemento. Para ello, cada componente tiene asociado un evento de selección que se activa al pulsar el botón de selección del mando a distancia. Este evento permite ejecutar la acción correspondiente al elemento seleccionado.

Las acciones desencadenadas por estos eventos pueden ser la creación de una nueva pantalla (cuando se selecciona una opción del menú lateral) o la creación de una pantalla de detalle (cuando se selecciona un contenido). En el caso de crear una nueva pantalla, se sigue el mismo proceso utilizado para la creación de la pantalla principal, mostrándose la nueva pantalla en la aplicación y almacenando la anterior en una pila de pantallas. Cuando se selecciona un contenido, se genera una pantalla de detalle con la información correspondiente al contenido seleccionado.

Creación de la pantalla de detalle

La pantalla de detalle muestra la información detallada de un contenido. Esta pantalla se genera en función de la información recibida tras realizar una llamada a la API con el identificador del contenido seleccionado.

Existen dos tipos de pantallas de detalle en función del tipo de contenido: contenedores y contenidos. Los **contenedores** son aquellos que agrupan otros contenidos, como una se-

rie que contiene capítulos. Los **contenidos**, por su parte, son elementos finales, como una película, un capítulo o un partido, y no tienen elementos hijos asociados.

La pantalla de detalle de un contenedor incluye una ficha con la información del contenedor, una lista de los contenidos que agrupa y una lista de contenedores relacionados. El movimiento a través de los contenidos hijos se realiza de manera vertical, similar al funcionamiento en la pantalla principal, mientras que para los contenidos relacionados, se utiliza un movimiento horizontal, como en los widgets de la pantalla principal.



(a) Pantalla de detalle de un contenedor



(b) Pantalla de detalle de un contenedor

Figura 6.2: Pantalla de detalle de un contenedor

Por otro lado, la pantalla de detalle de un contenido está compuesta por una ficha con la información del contenido. En esta ficha se incluye el título, el contenedor padre (si lo tiene, como en el caso de un capítulo de una serie, donde se muestra la serie a la que pertenece), una descripción corta, un botón que permite desplegar un popup con toda la información completa, los iconos de rating y edad, un botón de reproducción y, en caso de que la funcionalidad lo permita, un botón para añadir a favoritos. También se incluye una lista de contenidos relacionados.



Figura 6.3: Pantalla de detalle

Reproducción de un contenido

La reproducción de un contenido es una de las funcionalidades más importantes de la aplicación. Para acceder a ella hay dos opciones: a través de la pantalla de detalle del contenido explicada en el punto anterior, o si un contenido tiene el *trigger* de reproducción activado, se podrá acceder a la reproducción directamente desde la pantalla principal. Al seleccionar el botón de reproducción, la aplicación obtiene la información del contenido. En esta información se comprueba si el contenido necesita autenticación para ser reproducido y, en caso afirmativo (el usuario ya debería estar logueado), se comprueba si el usuario tiene permisos para ver el contenido. Esta es una comprobación de seguridad, ya que en caso de no tener acceso al contenido, el botón de reproducción aparecerá deshabilitado. Si el usuario tiene acceso, se realizan varias comprobaciones:

Origen del contenido: Lo primero es comprobar de dónde proviene el contenido. Existen dos opciones: que esté alojado en la base de datos de la empresa o en la del cliente. Si está en la del cliente, la URL del video ya estará en la información del contenido. En caso de estar en la base de datos de la empresa, se deben realizar una serie de comprobaciones para obtener la URL del video. Estas comprobaciones varían según si el contenido es gratuito o de pago; si es gratuito, la URL se puede construir directamente con la información disponible, pero si es de pago, se requieren verificaciones adicionales.

Reproductor: No todos los dispositivos pueden utilizar el mismo reproductor. Los ordenadores son más permisivos en este aspecto, pero las televisiones no. En el caso de los ordenadores, incluso existe la opción de que la API devuelva un *player* ya construido para ser usado directamente en la aplicación, aunque esta opción no se utiliza por el momento. El código detecta en qué sistema operativo (SO) y con qué tipo de contenido se está trabajando. Por el momento, hay dos reproductores disponibles (*VideoJs* y *ShakaPlayer*) y dos tipos de video (*VoD* y *Lives* o *YouTube*). Para cada caso, se monta el reproductor de manera específica.

Una vez que el video se está reproduciendo, el usuario dispone de las funcionalidades básicas para controlar la reproducción: pausa, play, adelantar, retroceder y salir. Si la funcionalidad está permitida y el usuario está registrado, la aplicación guarda el instante en el que se detuvo la reproducción, permitiendo que se pueda continuar viendo el contenido desde el mismo punto en el que lo dejó.

Añadir a favoritos

En caso de permitir usuarios registrados, la aplicación ofrece la funcionalidad de añadir contenidos a favoritos. Para ello, en la pantalla de detalle de contenido se muestra un botón en forma de corazón, que permite agregar el contenido a la lista de favoritos. Si el contenido

ya está añadido a favoritos, el botón aparece en color rojo, y pulsando sobre él se elimina de la lista de favoritos.

Búsqueda de contenidos

La búsqueda de contenidos es una funcionalidad esencial de la aplicación. Se puede acceder a ella de dos formas: a través del menú lateral o mediante una opción situada en la parte superior derecha de todas las pantallas. Al abrir el menú de búsqueda, se muestra un campo de texto donde el usuario puede introducir el nombre completo o parcial del contenido que desea buscar. Al hacer clic en el botón de búsqueda, la aplicación realiza una llamada a la API con el nombre del contenido y recibe una lista de los contenidos que coinciden con el criterio de búsqueda. Esta lista se presenta en una pantalla de resultados en formato de mosaico.

Obtención de la información de un directo

La obtención de la información de un directo permite a la aplicación recibir datos en tiempo real de los contenidos en emisión. Para ello, se realiza una llamada a la API con el identificador del directo, obteniendo la información a través de un EPG (Electronic Program Guide). Este EPG es un archivo JSON que contiene la información de los contenidos que se emitirán en un periodo determinado. La aplicación analiza este JSON y extrae la información en tiempo real del directo, incluyendo el título del contenido y su progreso, el cual se calcula a partir de los tiempos de inicio y fin del contenido marcados en el JSON, mostrando dicha información en la barra de progreso.

6.2.2 Adaptación a los diferentes sistemas operativos de televisión

El mercado de las televisiones inteligentes ha experimentado un gran crecimiento en los últimos años, lo que ha ampliado las opciones de aplicaciones y servicios disponibles para los usuarios. Sin embargo, este mercado es relativamente nuevo y se le exigen prestaciones similares a las de los dispositivos móviles u ordenadores personales. Cumplir con esas exigencias no es sencillo debido a varias razones: la capacidad de procesamiento de las televisiones, el soporte de los sistemas operativos a las tecnologías web y la diversidad de sistemas operativos disponibles.

Capacidad de procesamiento de las televisiones En los últimos años, la capacidad de procesamiento de las televisiones ha mejorado significativamente. Sin embargo, aún no es suficiente para ofrecer una experiencia de usuario comparable a la de un ordenador personal o un dispositivo móvil. Además, las tecnologías web son cada vez más complejas y exigentes, lo que hace que las televisiones no siempre puedan soportarlas. Al comparar la capacidad de

procesamiento de una televisión de alta gama con la de un ordenador o un móvil, aunque las televisiones han mejorado, siguen estando por detrás de los dispositivos de consumo masivo en términos de rendimiento.

Las especificaciones de una televisión de alta gama actual (3-4 GB de RAM, procesador de 4 núcleos) son comparables a las de dispositivos móviles de gama media-alta de hace algunos años, que hoy en día ya cuentan con 6-8 GB de RAM y procesadores de 8 núcleos. De manera similar, los ordenadores más básicos ahora ofrecen entre 8 y 16 GB de RAM.

Sopor te de los sistemas operativos a las tecnologías web Este problema está parcialmente relacionado con el anterior. Los sistemas operativos de las televisiones inteligentes no están tan adaptados a las tecnologías web como en otros dispositivos. Esto puede deberse tanto a la capacidad de procesamiento limitada como al hecho de que las televisiones entraron relativamente tarde en el mercado de los dispositivos inteligentes, lo que ha retrasado el desarrollo de aplicaciones y servicios optimizados para estos sistemas.

Diversidad de sistemas operativos Un problema adicional en el mercado de las televisiones inteligentes es la diversidad de sistemas operativos disponibles. Aunque en el mercado de los dispositivos móviles existen varios sistemas operativos, la mayoría de los móviles utiliza Android o iOS, lo que facilita el trabajo de los desarrolladores al crear aplicaciones. En cambio, la diversidad de sistemas operativos en las televisiones es mucho mayor, lo que obliga a los desarrolladores a adaptar sus aplicaciones a cada uno de ellos.

Adaptación a los diferentes sistemas operativos de televisión

El mayor desafío de esta aplicación es su adaptación a los diferentes sistemas operativos de televisión. Desde las primeras fases, el objetivo principal ha sido asegurar su accesibilidad en televisores, dado que este era el objetivo a corto plazo. Sin embargo, siempre se tuvo en cuenta que debía ser una aplicación multiplataforma. Aunque la implementación y las pruebas han estado enfocadas en optimizar el rendimiento en televisores, la aplicación está diseñada para funcionar en ordenadores y dispositivos móviles. Reactivando ciertas funcionalidades, como el click o hover del ratón, la aplicación podría funcionar en web sin mayores problemas. No obstante, para dispositivos móviles, las pruebas han sido limitadas y aún se requieren nuevas adaptaciones, ya que su optimización es un objetivo a más largo plazo. Mientras tanto, la prioridad sigue siendo asegurar la consistencia en televisores, a medida que se prepara su lanzamiento en las tiendas de aplicaciones de los distintos sistemas operativos de televisión.

Adaptar una misma aplicación a partir del mismo código supone ajustarse a las capacidades y limitaciones de cada SO. Si un SO no soporta ciertas funcionalidades, lo ideal es buscar una alternativa que funcione en todos los dispositivos por igual, y en caso de no ser posible,

se deberá buscar una solución específica para ese SO.

Adaptaciones realizadas

A continuación se detallan las adaptaciones generales realizadas para mantener la consistencia y el correcto funcionamiento de la aplicación en los diferentes sistemas operativos de televisión.

Interfaz La interfaz debe ser consistente en cualquier dispositivo en el que se utilice la aplicación. Para ello, se ha utilizado un diseño "responsive" que se adapta a cualquier resolución de pantalla. Esto es necesario si queremos usarla en dispositivos de distintas familias, pero también en televisores de distintas marcas y modelos. Un ejemplo de ello son las televisiones utilizadas en este proyecto para pruebas: LG y Samsung, con la misma resolución de 1920x1080, y TCL (Android TV), con una resolución de 960x540. Aunque todas las televisiones son de gamas similares, las diferencias en resolución son evidentes, lo que hace esencial asegurarse de que la aplicación sea consistente en cualquier televisión disponible en el mercado.

Para lograr esto, todo el diseño y las hojas de estilo CSS utilizan medidas relativas: porcentajes, "vh" y "vw". Se evita completamente el uso de medidas absolutas como "px", y también se evita el uso de "rem" y "em" para asegurar que el diseño sea coherente en todos los dispositivos.

Soporte de tecnologías web El soporte de tecnologías web es un aspecto crucial. Como se explicó en otras secciones, las tecnologías web utilizadas en este proyecto son JavaScript, HTML y CSS. Aunque todos los SO utilizados soportan estas tecnologías, no lo hacen completamente. Durante el desarrollo, se intentó minimizar el uso de librerías externas para garantizar la mantenibilidad del código y evitar problemas de compatibilidad en actualizaciones futuras. Un ejemplo de este desafío fue la eliminación de la funcionalidad de DOMParser, que no es compatible con Tizen, lo que obligó a implementar una solución manual para procesar archivos XML.

Navegación Un ejemplo de adaptación específica para cada SO es la navegación. Los comandos de los mandos a distancia varían ligeramente entre dispositivos, por lo que se creó una adaptación específica para cada uno. Cada SO cuenta con un archivo de configuración que traduce los códigos de los botones del mando a un formato común que la aplicación puede interpretar correctamente.

Reproducción de video La reproducción de video es una de las funcionalidades más importantes de la aplicación. En el caso de las televisiones inteligentes, las opciones de reproducidores de video son más limitadas que en las páginas web. Por ejemplo, mientras que WebOS y AndroidTV soportan VideoJs, Tizen no, y en su lugar utiliza Shaka Player, que no es compatible con los otros SO. Para solucionar esto, la aplicación detecta el SO y ajusta el reproductor de video en consecuencia.

Detección del estado de la red La detección del estado de la red también varía entre sistemas operativos. Aunque se intentó unificar esta funcionalidad, no fue posible obtener resultados consistentes en todos los casos. En Tizen se utiliza la librería webApis, mientras que en AndroidTV y WebOS se emplea la funcionalidad "navigator.connection".

Cerrar la aplicación La forma de cerrar la aplicación varía entre sistemas. Se investigó la documentación de cada uno y se implementó la lógica necesaria para cerrar correctamente la aplicación según el SO detectado.

Desafíos encontrados

Durante el desarrollo de la aplicación, surgieron varios desafíos que complicaron su adaptación. A continuación, se detallan algunos de los desafíos más importantes y las soluciones aplicadas.

El primer desafío fue la falta de información y soluciones para este tipo de aplicaciones. La mayoría de la información disponible proviene de la documentación oficial, que a menudo es limitada y no cubre todos los casos posibles. Los foros y comunidades de desarrolladores también son útiles, pero en muchos casos no ofrecen respuestas a problemas específicos en el desarrollo para televisores.

Foco automático en AndroidTV En las primeras pruebas en AndroidTV, se encontró un problema con el enfoque automático de los elementos en la pantalla. Esto provocaba movimientos inesperados en la interfaz al mover el mando. La solución fue desactivar este enfoque automático mediante un "listener" que interceptaba el evento de enfoque y lo desactivaba inmediatamente.

Botón de retorno con funcionamiento predefinido en AndroidTV Otro problema en AndroidTV fue el funcionamiento del botón de retorno. En la última actualización del SO, el botón cerraba automáticamente la aplicación si no había una página anterior en el historial. Como esta aplicación usa un diseño de página única, se interceptó el evento de retorno y se anuló su comportamiento predeterminado.

Splash screen en AndroidTV Finalmente, se encontró un problema con el "splash screen" predeterminado de Cordova en AndroidTV, que no se podía desactivar. Aunque se logró cambiar la imagen mostrada, el formato aún presenta problemas. Se sigue buscando una solución definitiva para este problema.

6.3 Implementación de la aplicación de análisis de datos

6.3.1 Instalación, configuración e integración de la API de Matomo

Para el desarrollo de la aplicación de análisis de datos, se ha utilizado la herramienta de análisis web Matomo. Aunque existen otras opciones como Google Analytics, se ha optado por Matomo en este primer desarrollo, ya que es la herramienta más utilizada en las distintas aplicaciones de la empresa.

Matomo [13] es una herramienta de análisis web de código abierto que permite a los administradores de sitios web obtener información detallada sobre los visitantes. Ofrece funcionalidades como la recopilación de datos, generación de informes, y la personalización de los mismos. Una de las principales características es su API REST, que permite a los desarrolladores acceder a los datos recopilados y realizar diversas operaciones sobre ellos.

A través de la API, se puede obtener la información necesaria para construir la aplicación de análisis de datos, utilizando estos datos como entrada para generar gráficas y tablas.

Matomo ofrece una integración sencilla con las aplicaciones en las que se requiera la recopilación de datos. Los pasos a seguir son:

1. Registrarse en la plataforma de Matomo.
2. Dar de alta un nuevo sitio web en la plataforma.
3. Descargar el código de seguimiento y añadirlo a la aplicación.

Tras estos pasos, Matomo comenzará a recopilar los datos de los visitantes de la aplicación. Además, es posible configurar opciones más avanzadas, como eventos personalizados, objetivos o la creación de segmentos. Para obtener datos sobre las reproducciones de vídeo, el reproductor debe configurarse para enviar eventos a Matomo cada vez que se reproduzca un vídeo.

Ejemplo de código de seguimiento de Matomo

```
1 <!-- Matomo -->
2 <script>
3 var _paq = window._paq = window._paq || [];
4 /* tracker methods like "setCustomDimension" should be called
before "trackPageView" */
```

```

5   _paq.push(['trackPageView']);
6   _paq.push(['enableLinkTracking']);
7   (function() {
8     var u="https://tiivii-ott.matomo.cloud/";
9     _paq.push(['setTrackerUrl', u+'matomo.php']);
10    _paq.push(['setSiteId', 'IdSite']);
11    var d=document, g=d.createElement('script'),
12    s=d.getElementsByTagName('script')[0];
13    g.async=true;
14    g.src='https://cdn.matomo.cloud/tiivii-ott.matomo.cloud/matomo.js';
15    s.parentNode.insertBefore(g,s);
})();
</script>
<!-- End Matomo Code -->
```

Una vez que Matomo está recopilando los datos, se pueden realizar consultas a la API para obtener la información necesaria. Para ello, es necesario obtener un *token* de acceso a la API, disponible en la configuración de la cuenta de Matomo, y registrar la URL de la página web desde la que se realizará la consulta. Con esta configuración y sabiendo qué datos se quieren obtener [5.3.2](#) y qué métodos utilizar, se puede empezar a realizar las consultas a la API.

6.3.2 Desarrollo de la aplicación de análisis de datos

Una vez configurada la API de Matomo, se pueden comenzar a realizar consultas a la misma para obtener la información necesaria. Estos datos serán utilizados para construir las gráficas y tablas que se mostrarán en la aplicación de análisis de datos.

Para la obtención, procesamiento y gestión de los datos, se ha desarrollado la aplicación con un enfoque escalable y mantenable. Durante el desarrollo, se ha empleado la tecnología de React, una biblioteca de JavaScript para la creación de interfaces de usuario, complementada con el uso de HTML, CSS, así como diversas librerías y APIs.

Estructura de la aplicación

La aplicación está compuesta por varios módulos, cada uno encargado de una tarea específica. La estructura de la aplicación es la siguiente:

- **Configuración:** Módulo responsable de configurar la aplicación. Este módulo genera las URLs necesarias y realiza las consultas a la API de Matomo. Cada URL se construye según el módulo correspondiente de Matomo, la acción deseada y las variables necesarias para la consulta. También permite configurar qué gráficos y datos se mostrarán en cada página.

- **Consultas:** Módulo encargado de realizar las consultas a la API de Matomo. Contiene las funciones que ejecutan las consultas y devuelven los datos obtenidos. Para cada acción, se utiliza el módulo API y la función `getProcessedReport` para obtener la información detallada de Matomo sobre cada acción.
- **Procesamiento:** Módulo responsable de procesar los datos obtenidos y crear las distintas páginas de la aplicación. Se solicitan los datos necesarios y se envían a los componentes de gráficas y tablas para su creación, ordenando los resultados para presentarlos en la página correspondiente.
- **Componentes:** Módulo que contiene los componentes de gráficas y tablas utilizados en la aplicación. Cada componente es un archivo independiente encargado de generar una gráfica o tabla específica.
- **Diseño:** Módulo que contiene las hojas de estilo CSS utilizadas en la aplicación para proporcionar un diseño atractivo y usable.

Desarrollo de la aplicación

Para el desarrollo de la aplicación utilizando la API de Matomo, el primer paso tras configurar y preparar Matomo fue obtener un servidor donde alojar la aplicación, ya que por razones de seguridad, la API bloquea las consultas realizadas desde un servidor local. Para ello, se utilizó el servicio de Netlify [14], una plataforma de alojamiento de aplicaciones web que permite desplegar aplicaciones de manera sencilla y rápida.

Una vez alojada la aplicación, se procedió con su desarrollo. El primer paso fue trabajar en los archivos de configuración, los cuales se encargan de generar las URLs necesarias para realizar las consultas a la API.

Ejemplo de archivo de configuración de la API de Matomo

```

1  const methodBase = 'MediaAnalytics';
2  const module = 'API';
3
4  export const MediaAnalytics_get = (idSite, period = 'day', date
5    = '2023-12-01,2024-07-01') => {
6    const method = `${methodBase}.get`;
7    return { url: getBaseUrl(module, method, { idSite, period,
8      date }), title: 'Overall Metrics' };
9  };
10
11  export const MediaAnalytics_getCurrentNumPlays = async (idSite,
12    lastMinutes = 180) => {
13    const action = 'getCurrentNumPlays';
14    const method = `${methodBase}.getCurrentNumPlays`;
```

```

12     return await fetchData(idSite, { module: methodBase,
13         action,url: getBaseUrl(module ,method, { idSite, lastMinutes
14     })));
15 };

```

Estos archivos reciben los parámetros necesarios para realizar las consultas a la API, crean la URL (getBaseUrl) y consiguen los datos (fetchData) necesarios para la creación de las gráficas y tablas.

Función para la creación de la URL de la API de Matomo

```

1 export const getBaseUrl = (module, method, params = {}) => {
2     const baseUrl =
3         `${baseURL}index.php?module=${module}&format=${format}&method=${method}&token_auth`;
4     const queryParams = new URLSearchParams(params).toString();
5     return `${baseUrl}&${queryParams}`;

```

Función para la obtención de los datos de la API de Matomo

```

1
2 export const fetchData = async (idSite, requestData) => {
3     var newChartData = null;
4     try {
5         try {
6             let dataUrl = API_getProcessedReport(idSite, 'year',
7                 'yesterday', requestData.module, requestData.action, language);
8             let response = await axios.get(dataUrl.url);
9             var processedData = response.data;
10        } catch (error) {
11            console.error('Error fetching data:', error);
12        }
13
14        // Usar la función de API general
15        const response1 = await axios.get(requestData.url);
16        const responseData = response1.data;
17
18        const data = {
19            value: responseData,
20            info: processedData ? processedData : {}
21        };
22
23        newChartData = data;
24        console.log('Data fetched:', newChartData);
25    } catch (error) {
26        console.error('Error fetching data:', error);
27    }

```

```

27
28     return newChartData;
29 }

```

Una vez obtenidos los datos y procesados según la configuración de la página correspondiente, se formatean y unifican antes de ser enviados a los componentes de gráficas y tablas para su creación. Tras generar los componentes, estos se añaden a la página, mostrando así los resultados.

Ejemplo de creación de una gráfica

```

1   return (
2       <div key={index} className="data-table-section">
3           <h2>{chartConfig.title}</h2>
4           <div className="chart-group">
5               {metrics.map((metric, metricIndex) => (
6                   <GraphRenderer
7                       key={metricIndex}
8                       chart={{
9                           type: chartConfig.type,
10                          labels: labels,
11                          data: dataPoints[metric],
12                          title: chartConfig.metrics[metric],
13                          metricType:
14                              chartConfig.data.info?.metadata.metricTypes[metric] || 'number',
15                          }
16                      />
17                  )))
18             </div>
19         );

```

Para detectar los formatos y los componentes que deben crearse, se desarrolló el componente *GraphRenderer*, que recibe los datos y el tipo de gráfica a generar, y se encarga de invocar al componente correspondiente para su creación. Este componente fue diseñado como un puente entre los datos y los componentes de gráficas y tablas, favoreciendo así la escalabilidad y mantenibilidad de la aplicación.

Código del componente GraphRenderer

```

1
2  const GraphRenderer = ({ chart, chartIndex }) => {
3      const { type, labels, data, title, metricType } = chart;
4
5
6      switch (type) {
7          case 'lineal':

```

```

8         return (
9             <div className="graph_component" key={chartIndex}>
10                <ChartComponent
11                  labels={labels}
12                  data={data}
13                  label={title}
14                  title={title}
15                  metricType={metricType}
16                />
17            </div>
18        );
19
20        case 'pie':
21            return (
22                <div className="graph_component" key={chartIndex}>
23                  <PieChartComponent
24                    labels={labels}
25                    data={data}
26                    title={title}
27                  />
28            </div>
29        );
30        ...
31    }
32 };

```

Para la creación de los distintos gráficos y tablas se ha utilizado la librería *Chart.js* [15]. Esta librería facilita la creación de gráficos y tablas de forma sencilla y rápida, ofreciendo una amplia gama de opciones que pueden personalizarse y adaptarse a las necesidades de la aplicación. Gracias a su flexibilidad, es posible generar gráficos y tablas ajustados a los requisitos específicos del proyecto. Hasta el momento, se han empleado los componentes *Line*, *Bar*, *Pie* y *Bubble* para la creación de las gráficas.

Ejemplo de creación de un grafico lineal

```

1
2 const ChartComponent = ({ data, labels, title, metricType }) => {
3     const chartData = {
4         labels,
5         datasets: [
6             {
7                 label: title,
8                 data,
9                 fill: false,
10                backgroundColor: 'rgba(75, 192, 192, 0.6)',
11                borderColor: 'rgba(75, 192, 192, 1)'
12            }
13        }
14    };
15
16    return (
17        <div className="graph_component" key={chartIndex}>
18            <LineChartComponent
19              chartData={chartData}
20              labels={labels}
21              title={title}
22            />
23        </div>
24    );
25};

```

```

12     },
13   ],
14 };
15
16 if(metricType === 'percentage') {
17   chartData.datasets[0].yAxisID = 'percentage';
18 }
19 const options = {
20   scales: {
21     x: {
22       beginAtZero: true,
23     },
24     y: {
25       beginAtZero: true,
26     },
27   },
28 };
29
30 return (
31   <div className="graph">
32     <h2>{title}</h2>
33     <Line data={chartData} options={options} />
34   </div>
35 );
36

```

Como se puede ver en el ejemplo de código, ChartJs permite la personalización de los distintos componentes modificando los colores, títulos, ejes, etc.

Automatización y configuración de las páginas

Siguiendo con el enfoque de escalabilidad y mantenibilidad de la aplicación, las páginas se han creado de forma dinámica. Para cada una de ellas, existe un archivo de configuración que especifica qué métodos y acciones deben ejecutarse para su creación. Además, se dispone de otro archivo que contiene la información básica de cada gráfica, en caso de que la llamada a *API.getProcessedReport* no devuelva los datos necesarios. Este archivo también incluye la función que permite obtener la información requerida para generar dicha gráfica.

Ejemplo de archivo de configuración de una página

```

1 export const pagesConfig = [
2 {
3   path: '/visitorSummary',
4   title: 'Resumen del visitante',
5   chartsConfig:visitCharts_summary,

```

```

6   components: [ "chartOptions", "DataOverviewTable",
7     "GraphRenderer" ],
8   },
9   {
10   path: '/visitTime',
11   title: 'Tiempo',
12   chartsConfig: visitCharts_time,
13   components: [ "GraphRenderer", "periodSelecter" ]
14   ...
15 ];

```

Ejemplo de archivo de configuración de una gráfica

```

1 export const visitsCharts_frequency = [
2   {
3     title: 'Visits - Frequency',
4     description: 'Get the frequency of visits.',
5     action: "get",
6     module: 'Visits',
7     period: 'month',
8     date: '2024-03-01,yesterday',
9     type: 'lineal',
10    metrics: {
11      "nb_visits_new" : "Nuevas visitas",
12      "nb_visits_returning": "Visitas que regresan"
13    },
14    data : [],
15    params: [ "period" ],
16    fetchDataFunction: visitFrequency_get,
17    async getData(idSite, period = this.period, date = this.date){
18      this.data = await visitFrequency_get(idSite, period, date)
19      if(this.data.info.metadata){
20        this.description = this.data.info.metadata.documentation;
21        this.title = this.data.info.metadata.name;
22        this.metrics = this.data.info.columns ||
23        this.data.info.metadata.metrics || this.metrics;
24      }
25      return this;
26    }
27  },
28];

```

Esta función se llamará desde la página correspondiente y se comenzará el proceso de creación de las gráficas.

Context y Hooks

Para la gestión de los datos y la comunicación entre los distintos componentes de la aplicación, se ha utilizado la API de Context y Hooks de React. Context es una API que permite compartir datos entre componentes sin la necesidad de pasarlos a través de *props*. Se crea un contexto y se envuelve la aplicación en un componente que provee dicho contexto. Los componentes que lo requieran podrán acceder a los datos desde este contexto.

En esta aplicación, se ha creado un contexto para la detección del *id* del sitio web que se está analizando. Este *id* es necesario para realizar consultas a la API de Matomo y obtener los datos correspondientes al sitio web.

Ejemplo de uso de Context:

```
const { idSite } = useContext(IdSiteContext);
```

Hooks es una API que permite a los componentes utilizar el estado y otras características de React sin la necesidad de escribir una clase. Son empleados para la gestión del estado de los componentes y la comunicación entre los mismos.

Los hooks más utilizados en esta aplicación son `useState` y `useEffect` (y `useContext` para el uso de Context del *id*).

useState: Hook que permite añadir estado a los componentes funcionales y actualizarlo cuando sea necesario.

```
const [isLoading, setIsLoading] = useState(true);
```

useEffect: Hook que permite realizar efectos secundarios en los componentes funcionales. Se ejecuta después de cada renderizado del componente.

Ejemplo de uso de `useEffect` para la actualización de los datos cuando cambia el *id* del sitio web:

```
useEffect(() => {
  const fetchData = async () => {
    setLoading(true);
    try {
      const evolutionData = await homeCharts_VisitsSection_Evolution;
      setVisitsEvolution(evolutionData);
    } catch (error) {
      console.error('Error fetching visits data:', error);
    }
  };
  fetchData();
});
```

```

    } finally {
      setLoading(false);
    }
  };
  fetchData();
}, [idSite]);

```

Despliegue de la aplicación

A medida que se iba desarrollando la aplicación, se realizaron pruebas para asegurar su correcto funcionamiento. Para ello, se utilizó Postman, una herramienta que permite realizar peticiones a una API y verificar que los datos devueltos sean correctos. El despliegue se llevó a cabo en el servicio de Netlify [14], que facilita la publicación de aplicaciones web de manera rápida y sencilla, permitiendo comprobar que la aplicación se desplegaba correctamente. Actualmente, la aplicación sigue alojada en Netlify y es accesible a través de la URL <https://kanaloa-dev.netlify.app>.

6.3.3 APIs y librerías utilizadas

Librerías

Para la implementación de la aplicación de análisis de datos se han utilizado diversas librerías que han facilitado el desarrollo. Estas librerías proporcionan funciones y métodos que permiten realizar operaciones de forma más rápida y eficiente que si se tuvieran que implementar desde cero.

Una de estas librerías es *Axios*, la cual facilita la realización de peticiones HTTP desde el cliente a un servidor. Esta herramienta, basada en promesas, simplifica la comunicación con APIs, permitiendo realizar solicitudes *GET*, *POST*, *PUT* y *DELETE*, entre otras, de manera eficiente y con un manejo sencillo de respuestas y errores. Su facilidad de uso y capacidad para manejar solicitudes asíncronas y configuraciones avanzadas, como cabeceras personalizadas y gestión de tiempos de espera, han sido clave para la integración con las APIs REST de la plataforma. Además, *Axios* soporta la interceptación de solicitudes y respuestas, lo que facilita la implementación de mecanismos de autenticación y gestión centralizada de errores, mejorando así la robustez y seguridad de la aplicación.

```

const config = {
  method: 'post',
  url: `${BASE_URL}/insertOne`,
  headers: {
    'Content-Type': 'application/json',
  }
}

```

```
'api-key': API_KEY,  
},  
data: data  
};  
  
try {  
  const response = await axios(config);  
  ...  
}
```

Otra librería utilizada fue *ChartJs*, explicada en la sección 6.3.2. Esta librería permite la creación de gráficos de forma sencilla y rápida. Ofrece una amplia variedad de gráficos, como barras, líneas y gráficos circulares, que se pueden personalizar con colores, títulos y leyendas.

APIs: OpenAi y MongoDB

Además de la API de Matomo, se han integrado otras APIs para dotar de funcionalidades adicionales a la aplicación de análisis de datos. Una de estas funcionalidades es el análisis y explicación de los datos recopilados mediante la API de *OpenAi*. Esta funcionalidad permite a los usuarios obtener una interpretación de los datos, lo que facilita la toma de decisiones.

La API de *OpenAi* utiliza tanto el contexto de la página como los datos de las gráficas para generar una explicación en lenguaje natural, comprensible para cualquier usuario sin conocimientos técnicos. Los elementos necesarios para este proceso son: contexto, datos y petición.

Contexto: Información general del sitio web. Este contexto se va creando y enriqueciendo con cada petición a la API, permitiendo ofrecer explicaciones más detalladas y precisas en futuras consultas.

Datos: Datos específicos de la gráfica a analizar, recopilados a través de la API de Matomo y enviados a la API de *OpenAi* para su interpretación.

Petición: *Prompt* generado a partir de la información del contexto y los datos de la gráfica, que se envía a la API de *OpenAi* para generar una explicación en lenguaje natural.

Tanto el contexto como los datos se almacenan para alimentar futuras llamadas y crear *prompts* más precisos. Para el almacenamiento de estos datos se utiliza *MongoDB*, una base de datos no relacional.

MongoDB: MongoDB es una base de datos no relacional que permite almacenar datos en formato JSON. Su escalabilidad y flexibilidad permiten almacenar grandes cantidades de datos y realizar consultas de manera eficiente. Para este proyecto, se ha utilizado el servicio en la nube *MongoDB Atlas*, que ofrece funcionalidades avanzadas como replicación de datos, recuperación ante desastres y escalabilidad automática, garantizando disponibilidad y rendimiento.

MongoDB Atlas se utiliza para almacenar los datos de contexto y las gráficas, permitiendo su análisis y generación de explicaciones.

API de MongoDB

```
1 const BASE_URL = 'https://eu-west-2.amazonaws.com
2 /app/data-hrcfvpe/endpoint/data/v1/action';
3
4 exports.handler = async (event, context) => {
5   const { collection, query } = JSON.parse(event.body);
6
7   const data = JSON.stringify({
8     collection: collection,
9     database: 'kanaloa',
10    dataSource: 'kanaloa',
11    filter: query,
12  });
13
14
15  const config = {
16    method: 'post',
17    url: `${BASE_URL}/findOne`,
18    headers: {
19      'Content-Type': 'application/json',
20      'api-key': API_KEY,
21    },
22    data: data
23  };
24
25  try {
26    const response = await axios(config);
27    return {
28      statusCode: 200,
29      body: JSON.stringify(response.data.document),
30    };
31  } catch (error) {
32    console.error(error);
33    return {
34      statusCode: 500,
35      body: 'Error fetching data',
36    };
37  }
38}
```

```
36 } ;  
37 }  
38 }
```

Capítulo 7

Pruebas

7.1 Introducción

En este capítulo se describen las pruebas realizadas durante el desarrollo del proyecto. Estas pruebas tienen como objetivo verificar el correcto funcionamiento de la aplicación y comprobar que cumple con los requisitos establecidos en la fase de análisis.

7.1.1 Plan de pruebas unitarias, integración y sistema

Desde el inicio del proyecto, se planificó un enfoque de pruebas exhaustivo que abarcara pruebas unitarias, de integración y de sistema, adaptado a la naturaleza de la aplicación y sus requisitos específicos. Dada la naturaleza de la aplicación, las pruebas se realizaron principalmente de manera manual, llevadas a cabo por diferentes miembros del equipo de desarrollo y calidad.

Pruebas Unitarias

Las pruebas unitarias se enfocaron en validar componentes individuales de la aplicación en sus fases iniciales de desarrollo. Estas pruebas se realizaron en paralelo al desarrollo de cada funcionalidad, comprobando que cada módulo y microservicio funcionara correctamente de forma aislada. El objetivo era identificar y corregir errores en las unidades más pequeñas de código antes de integrarlas en el sistema global.

Pruebas de Integración

Una vez validadas las unidades individuales, se realizaron pruebas de integración para verificar que los diferentes componentes del sistema, incluidos microservicios y módulos de la aplicación, interactuaran correctamente entre sí. Estas pruebas fueron esenciales para garantizar que la comunicación entre los distintos componentes y partes de la aplicación funcionara

sin problemas y que no se produjeran errores de integración.

Pruebas de Sistema

Las pruebas de sistema se llevaron a cabo utilizando la aplicación completa en dispositivos reales. Aunque inicialmente se consideró el uso de emuladores, se constató que estos no ofrecían una representación precisa de las capacidades de procesamiento y el comportamiento de los dispositivos reales. Por ello, se optó por realizar pruebas directamente en los dispositivos finales, asegurando que la aplicación funcionara de manera fluida y sin errores en condiciones reales de uso.

El proceso de pruebas de sistema incluyó tanto pruebas funcionales, que verificaron que todas las funcionalidades de la aplicación operaban como se esperaba, como pruebas de rendimiento, que aseguraron que la aplicación mantuviera un comportamiento óptimo bajo diferentes cargas y en distintos dispositivos.

Enfoque de Pruebas Continuas

Durante el desarrollo de cada funcionalidad, las pruebas fueron iterativas y continuas. Se realizaban pruebas parciales conforme se avanzaba en la implementación, asegurando que cada nueva parte integrada no introducía errores ni afectaba negativamente al rendimiento. Al finalizar cada funcionalidad, se llevaban a cabo pruebas más exhaustivas, tanto por el desarrollador como por otros miembros del equipo, para garantizar que la aplicación en su conjunto funcionaba correctamente antes de proceder con nuevas adiciones.

Este enfoque de pruebas continuas permitió identificar y resolver problemas de manera temprana, reduciendo el riesgo de errores acumulados en las etapas finales del desarrollo. Además, este método aseguró una alta calidad del producto final al abordar las posibles fallas en cada fase del desarrollo.

7.1.2 Pruebas de aceptación

Al estar desarrollando aplicaciones para terceros, las pruebas de la aplicación OTT realizadas por el equipo de desarrollo no son suficientes para garantizar la calidad del producto final. Por ello, se planificaron pruebas de aceptación que involucraran a los clientes, con el objetivo de validar que la aplicación cumplía con sus expectativas y requerimientos.

Las pruebas de aceptación se llevaron a cabo en diferentes etapas del desarrollo, permitiendo a los clientes evaluar y proporcionar retroalimentación sobre la aplicación en su estado actual. Esta retroalimentación fue fundamental para ajustar y mejorar la aplicación en función de las necesidades y preferencias de los clientes, asegurando que el producto final cumpliera con sus expectativas.

Las pruebas de aceptación se realizaron a través de un servidor de pruebas donde estaba alojada la aplicación, permitiendo a los clientes acceder a la aplicación y probarla en un entorno controlado. Una vez que el cliente completaba las pruebas, se realizaba una reunión telemática para obtener los resultados de las pruebas y el feedback del cliente. Durante estas reuniones se discutían los resultados y se acordaban los cambios y mejoras necesarios para la aplicación.

7.1.3 Pruebas realizadas por los marketplaces

Además de las pruebas de aceptación realizadas por los clientes, la aplicación OTT también es sometida a pruebas por parte de los marketplaces en los que se quiere publicar. Cada marketplace tiene sus propios requisitos y estándares de calidad que la aplicación debe cumplir para ser aprobada y publicada en su plataforma. Por ello, se realizan pruebas específicas para cada marketplace, asegurando que la aplicación cumple con los requisitos técnicos y de calidad establecidos por cada uno.

Una vez completadas las pruebas de cada marketplace y corregidas las deficiencias detectadas, se envía la aplicación en el formato adecuado para cada plataforma, junto con la documentación y la información requerida. A continuación, se espera la aprobación o los resultados de las distintas pruebas que realizan los marketplaces, que pueden incluir pruebas de rendimiento, seguridad, usabilidad, entre otras.

7.2 Resultados de las pruebas

Para este proyecto ya se han realizado pruebas con versiones del código en varios marketplaces. Uno de ellos fue el de LG. Los resultados de su prueba arrojaron un problema con la reproducción de los vídeos en la aplicación en un primer envío (fue corregido de inmediato en la siguiente versión ya que no era un problema grave). Al reenviar la aplicación, las funcionalidades pasaron satisfactoriamente las pruebas de funcionalidad y de rendimiento. Sin embargo, se encontró otro problema importante: la aplicación no se inicia en versiones más antiguas de webOS 5.0, esta incluida. Este problema se está intentando solucionar, pero no es un problema grave ya que a los clientes se ofreció una aplicación compatible y funcional con televisores que usen alguna de las últimas 4 versiones de webOS (webOS 6.0, 22, 23, 24), por lo que será un error que se intentará solucionar, pero que no afecta a la publicación de la aplicación.

Capítulo 8

Conclusiones

El desarrollo de la plataforma OTT ha supuesto un desafío técnico y organizativo significativo, dada la complejidad del proyecto y la necesidad de adaptarse a múltiples plataformas y clientes. A lo largo del proyecto, se han logrado importantes avances en términos de escalabilidad, flexibilidad y rendimiento, lo que ha permitido crear una aplicación robusta y adaptable, capaz de satisfacer las necesidades de diferentes usuarios y dispositivos.

8.1 Logros Alcanzados

Uno de los principales logros del proyecto ha sido la adopción de una arquitectura basada en microservicios, la cual ha demostrado ser altamente escalable y flexible. Aunque no implementé directamente los microservicios, el desafío consistió en integrar de manera eficiente estos componentes existentes en la aplicación y asegurar su correcta interacción. Esto incluyó la adaptación de la aplicación al uso de estos microservicios, permitiendo que cada uno de ellos funcione de manera independiente, lo que facilita el mantenimiento del sistema y la adición de nuevas funcionalidades. Además, se contribuyó a la mejora de algunos microservicios específicos, optimizando su rendimiento e integración. A través de este proceso, se logró una integración efectiva con APIs y servicios externos, garantizando la interoperabilidad del sistema en diferentes entornos tecnológicos.

El enfoque en la experiencia de usuario (UX) ha dado lugar a una interfaz intuitiva y coherente en todas las plataformas soportadas. Se ha logrado un diseño minimalista que facilita la navegación y comprensión de los datos por parte del usuario, lo que se ha reflejado en la fluidez de la interacción y la satisfacción de los usuarios finales.

Otro logro clave ha sido el desarrollo de un plan de pruebas integral que incluyó pruebas unitarias, de integración y de sistema. Este enfoque permitió identificar y corregir errores de manera temprana, asegurando un alto nivel de calidad en el producto final. Las pruebas realizadas en dispositivos reales, en lugar de emuladores, han sido fundamentales para garantizar

que la aplicación funcione correctamente en las condiciones de uso previstas.

8.2 Desafíos y Soluciones

El proyecto no estuvo exento de desafíos. Uno de los mayores retos fue garantizar que la aplicación funcionara de manera óptima en una amplia gama de dispositivos y sistemas operativos. La diferencia en las capacidades de procesamiento y las peculiaridades de cada plataforma obligaron a realizar ajustes específicos en el código y a adoptar enfoques de diseño multiplataforma más sofisticados.

8.3 Aprendizajes

A lo largo del desarrollo de este proyecto, se han obtenido valiosas lecciones que pueden aplicarse en futuros desarrollos. Algunas de las lecciones más importantes tienen que ver con la recogida de información sobre problemas, tecnologías y soluciones, la importancia de la colaboración y la comunicación en equipos multidisciplinarios, y la necesidad de adaptarse a los cambios y desafíos que surgen durante el desarrollo de un proyecto. Otro aprendizaje ha sido la importancia de la comunicación con los clientes y la retroalimentación continua, lo que ha permitido ajustar y mejorar la aplicación en función de las necesidades y preferencias de los usuarios.

8.4 Conclusión Final

En resumen, el desarrollo de la plataforma OTT ha sido un proyecto exitoso que ha cumplido con los objetivos propuestos, proporcionando una aplicación escalable, flexible y de alto rendimiento. Aunque se enfrentaron desafíos significativos, las soluciones implementadas han permitido superar estos obstáculos y entregar un producto que no solo satisface las necesidades actuales de los clientes, sino que también está preparado para evolucionar y adaptarse a futuros requerimientos. Este proyecto no solo representa un avance técnico importante, sino que también establece una base sólida para desarrollos futuros en el ámbito de las plataformas de distribución de contenido.

Capítulo 9

Futuros Objetivos

Los objetivos futuros, tanto a corto como a largo plazo, ya han sido explicados en mayor o menor medida a lo largo de la memoria. Estos objetivos abordan futuras mejoras y ampliaciones de la plataforma OTT, así como la exploración de nuevas tecnologías y funcionalidades que puedan aportar valor añadido a la aplicación.

Uno de los objetivos principales es la expansión de la aplicación a la mayor cantidad de dispositivos y plataformas posibles. Para esto, se planea seguir trabajando en la adaptación de la aplicación a nuevas plataformas y sistemas operativos. Los objetivos a corto plazo son conseguir una aplicación funcional en los *marketplaces* tanto de Vidaa OS (Hisense) como de FireTV, y a largo plazo, tener las aplicaciones móviles y de tabletas en los *marketplaces* de Google Play y App Store.

Otro objetivo es seguir optimizando el rendimiento de la aplicación, tanto en términos de velocidad y fluidez como en el consumo de recursos, para dotar a la aplicación de una experiencia de usuario a la altura de las mejores aplicaciones OTT del mercado. Para ello, hay que continuar con el análisis de nuevas tecnologías y técnicas de optimización, así como con la mejora de los microservicios y la infraestructura de la aplicación para garantizar un rendimiento óptimo en todo momento.

Un objetivo muy importante, que se quiere llevar a cabo a corto plazo, es la implementación de una mayor personalización para los clientes. Esto incluye la posibilidad de personalizar la interfaz de usuario, los contenidos y las funcionalidades de la aplicación, manteniendo un solo código y permitiendo a los clientes diferenciarse cada vez más entre ellos, siempre procurando reducir el tiempo de desarrollo y mantenimiento de la aplicación.

También se continuará implementando funcionalidades de productos de la empresa, como la previsualización de los contenidos, que ofrece a los usuarios un video corto con un breve resumen de un contenido, generado mediante inteligencia artificial. Este producto está siendo utilizado actualmente por clientes como RTVE, y se planea implementarlo en las próximas semanas en este proyecto.

CAPÍTULO 9. FUTUROS OBJETIVOS

En cuanto a la aplicación de análisis de datos, el plan es optimizar y maximizar la información recogida en las distintas aplicaciones, y continuar con la expansión de la misma. Por el momento, está habilitada para uso interno, pero se planea, en un futuro cercano, ofrecerla a los clientes para que puedan analizar los datos de sus aplicaciones y tomar decisiones basadas en ellos.

Apéndices

Bibliografía

- [1] C. N. de los Mercados y la Competencia, “Dos de cada tres hogares que consumen contenidos audiovisuales online de pago usan más de una plataforma,” 2023. [En línea]. Disponible en: <https://www.cnmc.es/prensa/panel-hogares-audiovisual-20231215>
- [2] B. Comunicación, “Análisis sobre la evolución de otts en acceso población,” 2023. [En línea]. Disponible en: <https://barloventocomunicacion.es/barometrotv-ott/analisis-sobre-la-evolucion-de-ottts-en-acceso-poblacion/>
- [3] ——, “6ª ola junio 2024 barómetro ott - multidispositivo,” 2024. [En línea]. Disponible en: <https://barloventocomunicacion.es/barometrotv-ott/6a-ola-junio-2024-barometro-ott-multidispositivo/>
- [4] Wikipedia, “Servicio ott.” [En línea]. Disponible en: <https://es.wikipedia.org/wiki/ServicioOTT>
- [5] ——, “Arquitectura de software.” [En línea]. Disponible en: https://es.wikipedia.org/wiki/Arquitectura_de_software
- [6] ——, “Microservicio.” [En línea]. Disponible en: https://es.wikipedia.org/wiki/Arquitectura_de_microservicios
- [7] ——, “Experiencia de usuario.” [En línea]. Disponible en: https://es.wikipedia.org/wiki/Experiencia_de_usuario
- [8] ——, “Tizen.” [En línea]. Disponible en: <https://es.wikipedia.org/wiki/Tizen>
- [9] ——, “Webos.” [En línea]. Disponible en: <https://es.wikipedia.org/wiki/WebOS>
- [10] ——, “Android tv.” [En línea]. Disponible en: https://es.wikipedia.org/wiki/Android_TV
- [11] ——, “Apache cordova.” [En línea]. Disponible en: https://es.wikipedia.org/wiki/Apache_Cordova

- [12] T. C. T. M. Association, “Hisense’s viddaa is now the no. 2 smart tv os globally behind samsung tizen, trade group says (chart of the day).” [En liña]. Dispoñible en: <https://www.nexttv.com/news/hisenses-vidaa-is-now-the-no-2-smart-tv-os-globally-behind-samsung-tizen-trade-group-says-chart-of-the-day>
- [13] Wikipedia, “Matomo.” [En liña]. Dispoñible en: <https://es.wikipedia.org/wiki/Matomo>
- [14] ——, “Netlify.” [En liña]. Dispoñible en: <https://en.wikipedia.org/wiki/Netlify>
- [15] ChartJs.org, “Chartjs.” [En liña]. Dispoñible en: <https://www.chartjs.org/>