



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	32
4.5.3	Tecnologías	33
5	Diseño	34
5.1	Introducción	34
5.2	Diseño de la aplicación OTT	34
5.2.1	Introducción al diseño de la aplicación <i>ott</i>	34
5.2.2	Diseño de la arquitectura del sistema	35
5.2.3	Diseño de la interfaz de usuario	38
5.2.4	Diagramas UML	43
5.3	Diseño de la aplicación de análisis de datos	43
5.3.1	Introducción al diseño de la aplicación de análisis de datos	43
5.3.2	Estudio y diseño de la aplicación de análisis de datos	43

6 Implementación	47
6.1 Introducción a la implementación	47
6.2 Implementación de la aplicación OTT	47
6.2.1 Desarrollo de las funcionalidades básicas	47
6.2.2 Adaptación a los diferentes sistemas operativos de televisión	55
6.3 Implementación de la aplicación de análisis de datos	60
6.3.1 Instalación, configuración e integración de la API de Matomo	60
6.3.2 Desarrollo de la aplicación de análisis de datos	61
6.3.3 APIs y librerías utilizadas	69
7 Pruebas	73
7.1 Introducción	73
7.1.1 Plan de pruebas unitarias, integración y sistema	73
7.1.2 Pruebas de aceptación	74
7.1.3 Pruebas realizadas por los marketplaces	75
7.2 Resultados de las pruebas	75
8 Conclusiones	76
8.1 Logros Alcanzados	76
8.2 Desafíos y Soluciones	77
8.3 Aprendizajes	77
8.4 Conclusión Final	77
9 Futuros Objetivos	78
Bibliografía	82

Índice de Figuras

5.1	Captura de la interfaz en las primeras fases del proyecto	39
5.2	Captura de la interfaz más recientes, todavía con pruebas por parte del cliente	39
5.3	Captura de la interfaz casi final	40
5.4	Ejemplos de un widget básico de tipo "banner" y "destacado"	43
5.5	Ejemplos de un widget básico de tipo "poster"	43
6.1	Menú lateral	51
6.2	Pantalla de detalle de un contenedor	53
6.3	Pantalla de detalle	53

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.

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 va a realizar un estudio de las características de estos elementos y como estos encajan entre sí para el desarrollo de la aplicación.

4.5.1 Sistemas operativos

Cuando se comenzó a hablar del proyecto, el objetivo principal era el de comenzar con el desarrollo enfocado en televisores, pero sin perder la perspectiva del resto de dispositivos. Además, también se tenía una idea más o menos clara de los sistemas operativos con los que comenzar: Android TV, Tizen, webOS y Apple TV. Como ya se ha mencionado en la sección 4.4.1, Apple TV tuvo que ser descartado, pero los otros tres sistemas operativos siguen siendo válidos. Se estudiaron las características de estos sistemas operativos a través de la documentación oficial y de la (escasa) información que se podía encontrar en la red. Las características tecnológicas básicas ya han sido comentadas en la sección de fundamentos tecnológicos 2.3.3. En este apartado vamos a extender un poco más estas características, a través del análisis realizado para el proyecto.

características de los sistemas operativos Como ya se ha comentado las tecnologías que se querían utilizar para el desarrollo de la aplicación eran JavaScript, HTML y CSS. El primer análisis que se realizó fue el de la compatibilidad de estas tecnologías con los distintos sistemas operativos. Aquí se descubrió que tanto Tizen como webOS son compatibles con estas tecnologías, por lo menos con las funcionalidades más básicas y necesarias, ya que existen funcionalidades más recientes o complejas que, aunque poco a poco se van integrando, todavía no están disponibles en estos sistemas operativos. Por otro lado, Android TV propuso un reto, ya que, por defecto, no es compatible con estas tecnologías. Sin embargo, estudiando un poco más a fondo, se descubrió que Android TV permite el desarrollo de aplicaciones utilizando tecnologías web a través de una herramienta llamada Apache Cordova ??.

Empaquetado y distribución de aplicaciones Otro punto importante que fue objeto de estudio fue como se debían configurar y empaquetar las aplicaciones para poder probar y distribuir en los distintos sistemas operativos. De nuevo, Tizen y webOS son muy similares en este aspecto, ya que aunque tienen formatos de empaquetado diferentes (wgt e ipk respectivamente), la estructura de los archivos son muy parecidos: ambas plataformas requieren un archivo HTML principal, un archivo de configuración (config.xml y package.json respectivamente) principal de la aplicación como nombre, versión, etc. , el código de la aplicación (JavaScript, HTML y CSS) y los recursos necesarios (imágenes, fuentes, etc.). Por otro lado, la estructura de Android TV es distinta ya que la estructura principal de la aplicación es la de Apache Cordova y todo el código de este necesario para convertir la aplicación a un formato

compatible con Android TV. Dentro de esta estructura encontramos:

- **www**: Directorio que contiene el código de la aplicación (JavaScript, HTML y CSS).
- **config.xml**: Archivo de configuración de la aplicación.
- **platforms**: Directorio que contiene las plataformas en las que se ha empaquetado la aplicación.
- **plugins**: Directorio que contiene los plugins de Cordova que se han utilizado en la aplicación.

Herramientas de desarrollo Por último, se estudiaron las herramientas de desarrollo que se podían utilizar para el desarrollo de la aplicación. El desarrollo llevado a cabo ha sido a través de un código común desarrollado en Visual Studio Code. Sin embargo, la ejecución de las pruebas y la generación de los archivos de empaquetado no se podía realizar de la misma manera. Para cada SO se han creado unas plantillas con la estructura necesaria en cada caso y en las que se adjuntaba el código común y se añadía el código específico para cada SO (un archivo de configuración básico necesario para unas pocas funcionalidades específicas de cada SO).

Cada SO ofrece distintas opciones para la ejecución, pruebas y empaquetado de las aplicaciones.

- **Tizen**: Tizen Studio es la herramienta oficial de Samsung para el desarrollo de aplicaciones para Tizen. Esta herramienta ofrece muchas funcionalidades para el desarrollo como la gestión de dispositivos, emuladores, etc. Además, permite la generación de los archivos de empaquetado y la instalación de la aplicación en un televisor Samsung con Tizen OS.
- **WebOS**: LG ofrece una CLI (Command Line Interface) para la ejecución de las funcionalidades necesarias para la creación, empaquetado y distribución de aplicaciones para WebOS. Además, también está disponible una extensión para Visual Studio Code que facilita aún más el desarrollo.
- **Android TV**: Las aplicaciones para Android TV se pueden empaquetar y distribuir a través de Android Studio, que ofrece todas las funcionalidades necesarias para el desarrollo de las aplicaciones, desde escribir código hasta depurar e inspeccionar la aplicación. Sin embargo, como en este proyecto no se está utilizando Android de forma nativa, si no que se está adaptando a través de Cordova, Android Studio deja de tener todas las funcionalidades disponibles, debido a que ni tiene la estructura ni el lenguaje de programación que se utiliza por defecto en Android. La funcionalidad más afectada

por esto es la de ejecución ya que no permite crear el apk correctamente porque no reconoce el código. Por este motivo, el empaquetado e instalación de la aplicación en un televisor Android se ha realizado a través de un terminal utilizando los comando ofrecidos por Cordova.

4.5.2 Dispositivos

Para el estudio de los dispositivos, los puntos en los que se centró el análisis fueron la capacidad de los dispositivos para ejecutar la aplicación, la resolución de pantalla y el tamaño de la pantalla.

El segundo punto arrojó una respuesta muy clara: la aplicación debía ser adaptativa. La aplicación debía ser capaz de adaptarse a cualquier resolución de pantalla, ya que se iba a ejecutar en televisores de distintas marcas y modelos, en móviles, en Pc, etc.

La filosofía de diseño que se tomó desde un principio fue la filosofía de diseño responsivo. Esta filosofía permite adaptarse automáticamente al tamaño de la pantalla del dispositivo en el que se visualiza. Para ello, se utilizan unidades relativas como porcentajes o vh y vw en lugar de unidades absolutas como px. De esta manera, la aplicación se adapta a cualquier resolución de pantalla, ya sea un televisor, un móvil o una tablet. Por ejemplo, para este proyecto las pruebas se han realizado en dos televisores (Samsung y Lg) que tienen una resolución de 1920x1080 ambas, y con una TCL(Android TV) que tiene una resolución de 920*540. La aplicación se ha adaptado correctamente a los tres dispositivos, mostrando la misma información y funcionalidades en todos ellos.

Por otro lado, el primer punto fue un estudio clave, ya que la capacidad actual varia enormemente entre los distintos dispositivos sobre los que queremos trabajar, incluso entre dispositivos de la misma familia como los televisores, existen diferencias de eficiencia y capacidad entre los distintos modelos y SO. Unas primeras pruebas de eficiencia demostrarón que las funcionalidades que en un portatil funcionaban sin ningún tipo de problema ni sobrecarga para el dispositivo, a la hora de probar en las televisiones los movimientos se veían pesados y lentos. Como no es posible estudiar todos los modelos de los diferentes dispositivos de las distintas marcas y la aplicación busca llegar al mayor número de dispositivos posible, el estudio se realizó sobre las herramientas y funcionalidades del desarrollo más eficientes.

Gracias a este estudio se consiguió mejorar la eficiencia y fluidez de la aplicación en los dispositivos de menor capacidad buscando cual era la manera más eficiente de realizar las distintas funcionalidades. Por ejemplo, se demostró que utilizar la propieda transform: translate() en lugar de la propiedad top o left para mover los elementos de la aplicación era mucho más eficiente y fluido.

Se encontró otra manera de aumentar la eficiencia, por ejemplo, con una propiedad de CSS que permite acelerar el renderizado de la aplicación. Esta propiedad es la de will-change. Esta

propiedad permite al navegador saber que un elemento va a cambiar y que debe prepararse para ello. De esta manera, el navegador puede optimizar el renderizado de la aplicación y mejorar la experiencia del usuario.

4.5.3 Tecnologías

En cuanto al estudio de las tecnologías, este es constante a lo largo de todo el desarrollo del proyecto. Si bien también lo es el estudio de los sistemas operativos y dispositivos en casos concretos, el estudio de las tecnologías es algo que se realiza de manera constante, ya que no solo las tecnologías cambian y evolucionan, sino que es muy importante para un proyecto buscar, estudiar y analizar las distintas tecnologías para sacarles el máximo partido y mejorar la eficiencia y la calidad de la aplicación. Así, a lo largo de todo el desarrollo se han buscado las mejores funciones que nos ofrece JavaScript que nos pudieran ayudar con cada una de las funcionalidades a implementar. También se estudian las mejores librerías disponibles para utilizar en la aplicación de análisis de datos, a través de Node.js, una potente herramienta con un catálogo de librerías muy amplio, el cual si se estudia a fondo puede mejorar nuestros desarrollos de manera exponencial. Por otro lado, estudiar las mejores estructuras y elementos que nos ofrece HTML y CSS, y las mejores combinaciones entre estas tecnologías, va a ser clave tanto para la eficiencia de la aplicación como para la presentación de la interfaz.

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 un tanto peculiar, ya que en un principio se trabajó sobre diseños de interfaz ya realizados en otras plataformas para otros clientes, buscando crear una base sobre la que trabajar para desarrollar todas las funcionalidades, conociendo de antemano que no sería el diseño final, y posteriormente, una vez la aplicación comenzó a estar en una fase más avanzada con gran cantidad de las funcionalidades ya implementadas, se comenzaron las reuniones con los distintos clientes en las que además de recoger sus impresiones y sugerencias y estudiar nuevas funcionalidades a implementar, se comenzó a detallar a su gusto la interfaz y la experiencia de usuario de la aplicación.

A lo largo del desarrollo de la aplicación, se han modificado y añadido nuevas funcionalidades. Gracias a la flexibilidad de la metodología ágil, se ha podido ir adaptando la aplicación

a estas nuevas funcionalidades, a las necesidades de los clientes y a las nuevas tendencias del mercado, de tal manera que cada vez que se terminaba una funcionalidad o una iteración, se realizaban reuniones internas o con los clientes para validar y planificar las siguientes etapas. Una vez determinado el objetivo de la nueva iteración, comenzaba el ciclo de análisis, diseño, implementación y pruebas. Esto permite una adaptación continua y una 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 esta diseñada para integrarse en la arquitectura general de la empresa y se basa en una arquitectura basada en microservicios.

Arquitectura basada en microservicios

La arquitectura basada en microservicios es un enfoque para el diseño de aplicaciones que consiste en un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP como es el caso de este proyecto) [6].

Cada microservicio está especializado en una tarea concreta y trabaja de forma independiente. De esta manera tendremos las funcionalidades de la plataforma OTT distribuidas en diferentes microservicios, aislando las funcionalidades y permitiendo que cada uno de ellos pueda ser desarrollado, desplegado y escalado de forma independiente. Esto facilita la evolución de la plataforma, ya que se puede mejorar cada microservicio con la confianza de que si se hace correctamente no afectará al resto de la plataforma. Lo mismo ocurre con la tolerancia a fallos, ya que si la arquitectura está bien diseñada, un fallo en un microservicio no debería afectar al resto de la plataforma, o debería hacerlo lo menos posible. Además, cada microservicio puede ser reutilizado en diferentes proyectos, lo que facilita la creación de nuevas aplicaciones y la 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 de los mismos, así como la gestión de perfiles, intereses, preferencias y historial.
- **Directus - CMS:** encargado de la gestión y almacenamiento de los contenidos de la plataforma, de los metadatos de los contenidos y de los ficheros multimedia.

- **CAS - Servicio de acceso condicional:** encargado de la gestión de los accesos condicionales a los contenidos protegidos de la plataforma, incluyendo la gestión de licencias, DRM, cifrado y protección de contenidos.
- **Orquestador:** Su objetivo es realizar las comprobaciones y procesos que dependan de más de uno de los microservicios de la aplicación
- **Importer:** Microservicio utilizada en los casos en los que el cliente posee una Base de Datos con todos los contenidos e información de la plataforma OTT y se necesita importar a nuestra BD interna que es la que alimenta a la plataforma.
- **Player:** Utilizado en otras aplicaciones para la obtención del un HTML que contiene el reproductor de video. Esto no es posible por el momento de utilizar en la aplicación OTT multiplataforma porque para cada SO es necesario reproductores distintos y el backend todavía no tiene soporte. Sin embargo, a raíz del desarrollo de esta aplicación se ha creado un endpoint que devuelve la URL de un video concreto para poder reproducirlo. Esta URL en un principio era construida por la aplicación OTT, sin embargo, para poder aliviar a la plataforma de tareas de este estilo se ha modificado el microservicio para que sea él quien construya la URL. Se va a trabajar ahora en adaptación del microservicio para que pueda devolver un HTML con el reproductor necesario para cada SO.

Estos microservicios son utilizados a través de una API REST, que permite la comunicación entre los diferentes componentes de la plataforma. Cada microservicio se comunica con los demás a través de esta API, enviando y recibiendo datos en formato JSON. De esta manera, el frontend de la aplicación está en continua comunicación con el backend, solicitando y enviando datos a través de las distintas rutas de la API. Cada microservicio ofrece un catálogo de peticiones que se pueden realizar, y el frontend las utiliza cuando el usuario interactúa con la aplicación.

Objetivos de la arquitectura

La arquitectura utilizada en la aplicación está creada con el objetivo de crear una aplicación escalable, flexible y fácil de mantener. Uno de los pilares fundamentales son los microservicios ya comentados, que permiten aislar las funcionalidades de la aplicación y desarrollarlas de forma independiente. Sin embargo, los microservicios por sí solos no son suficientes para garantizar que la aplicación sea escalable y flexible, sino que el uso que se haga de ellos y la forma en la que se comuniquen entre ellos también es importante. Por ello, se han seguido una serie de buenas prácticas y patrones de diseño que garantizan que la aplicación sea robusta y fácil de mantener. Algunos de los objetivos de la arquitectura son:

- **Escalabilidad:** Durante la etapa de diseño, la escalabilidad fue un objetivo clave. Se diseñó la arquitectura del sistema para permitir un crecimiento tanto en la capacidad de usuarios como en la adición de nuevas funcionalidades. Esto se logró mediante el diseño modular y el desacoplamiento de componentes, lo que permite que cada parte de la aplicación funcione de manera independiente y pueda escalar horizontalmente (añadiendo más instancias) y verticalmente (optimizando los recursos) según sea necesario.

El diseño modular fue esencial para manejar la naturaleza multicliente de la plataforma, permitiendo que cada cliente personalice la aplicación sin afectar a los demás. Se implementaron patrones de diseño como microservicios y técnicas de gestión eficiente de la comunicación entre servicios, asegurando que el sistema mantenga su rendimiento y estabilidad a medida que crece.

- **Flexibilidad:** El diseño de la plataforma se centró en crear un sistema flexible que pudiera adaptarse a las necesidades específicas de cada cliente. Esto incluyó la capacidad de personalizar la interfaz de usuario y ajustar las funcionalidades sin modificar el código base. La flexibilidad se logró mediante la implementación de una arquitectura basada en componentes, lo que permite que los módulos sean activados o desactivados según los requisitos del cliente.

Además, se diseñó la plataforma para ser interoperable con otros sistemas y servicios, facilitando la integración con APIs de terceros y garantizando que la plataforma pueda operar en diversos entornos tecnológicos.

- **Interoperabilidad:** Desde la etapa de diseño, se priorizó la interoperabilidad del sistema, asegurando que la plataforma pueda integrarse y comunicarse eficazmente con otros sistemas y servicios externos. El diseño se centró en utilizar estándares de la industria y tecnologías que permitan una integración sin problemas, asegurando que la plataforma sea compatible con una amplia gama de servicios, desde APIs hasta sistemas de gestión de contenido y análisis de datos.

La capacidad de manejar diferentes protocolos de comunicación y formatos de datos fue integrada desde el inicio del diseño, permitiendo que la plataforma se adapte fácilmente a nuevas integraciones y expansiones futuras.

- **Adaptación Multiplataforma:** Durante la fase de diseño, se puso un gran énfasis en asegurar que la plataforma OTT fuera capaz de adaptarse a una amplia variedad de dispositivos y sistemas operativos, incluyendo navegadores web, dispositivos móviles y smart TVs. El diseño responsivo y el uso de frameworks multiplataforma permitieron reutilizar gran parte del código base, optimizando el desarrollo y garantizando una experiencia de usuario coherente.

El diseño también incluyó configuraciones específicas para cada plataforma, permitiendo que la aplicación detecte y se ajuste automáticamente al entorno en el que se ejecuta, lo que garantiza un rendimiento óptimo y la máxima funcionalidad en todos los dispositivos.

- **Mantenibilidad:** La mantenibilidad del sistema fue un objetivo central durante la etapa de diseño. Se diseñó la arquitectura modularmente, permitiendo que los componentes sean actualizados o reemplazados sin afectar el sistema completo. Además, se implementaron prácticas de diseño que facilitan el mantenimiento continuo, como la documentación detallada y la automatización de pruebas y despliegues.

La elección de herramientas y tecnologías que soporten la integración continua (CI/CD) fue parte integral del diseño, asegurando que las actualizaciones y mejoras puedan ser implementadas rápidamente, con un mínimo de interrupciones.

- **Rendimiento y Optimización:** El rendimiento óptimo fue un objetivo fundamental desde la etapa de diseño. Se diseñaron estrategias para manejar grandes volúmenes de tráfico y datos sin degradar la experiencia del usuario. Esto incluyó la optimización de la comunicación entre componentes, el uso de caché, y la distribución de la carga de trabajo a través de múltiples instancias y CDNs.

La capacidad de monitorear y ajustar el rendimiento en tiempo real también se integró en el diseño, asegurando que la plataforma pueda mantener un alto rendimiento bajo condiciones de alta demanda.

- **Experiencia de Usuario (UX):** El diseño de la plataforma se centró en ofrecer una experiencia de usuario excepcional. Esto implicó la creación de una interfaz intuitiva, accesible y coherente en todos los dispositivos soportados. El diseño se basó en principios centrados en el usuario, con un enfoque en la personalización y la facilidad de uso.

Durante el diseño, se realizaron múltiples iteraciones de pruebas de usabilidad para garantizar que la interfaz no solo fuera funcional, sino también agradable y eficiente para los usuarios finales. El diseño también consideró la capacidad de personalizar la experiencia del usuario, ofreciendo recomendaciones y configuraciones ajustadas a las preferencias individuales.

5.2.3 Diseño de la interfaz de usuario

La interfaz de usuario es la parte de un programa que interactúa con el usuario. En el caso de *Ott* es la parte que se encarga de mostrar la información al usuario y de permitirle interactuar con el sistema. La interfaz de usuario es una parte fundamental de cualquier sistema,

ya que es la parte que el usuario ve y con la que interactúa. Por tanto, es importante que la interfaz de usuario sea clara, sencilla e intuitiva, para que el usuario pueda utilizar el sistema de forma eficiente y sin problemas.

Cuando se comenzó con el diseño de la aplicación, no se trabajó sobre un diseño final exacto ya que este dependía de los clientes que fueran uniéndose al proyecto y decidiendo su propia interfaz. Lo que se fue diseñando a medida que avanzaba el proyecto fue una interfaz base que permitiera ir añadiendo nuevas funcionalidades y los distintos componentes personalizables que se irían añadiendo a la interfaz.

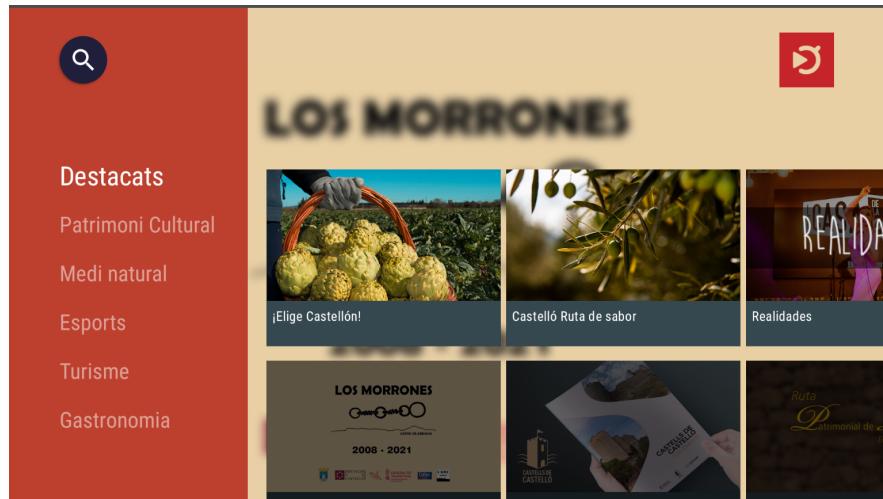


Figura 5.1: Captura de la interfaz en las primeras fases del proyecto

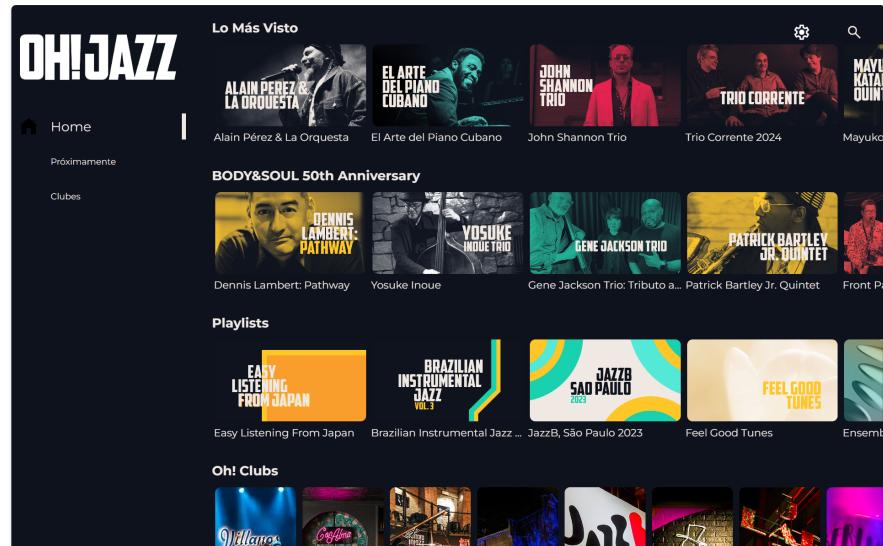


Figura 5.2: Captura de la interfaz más recientes, todavía con pruebas por parte del cliente

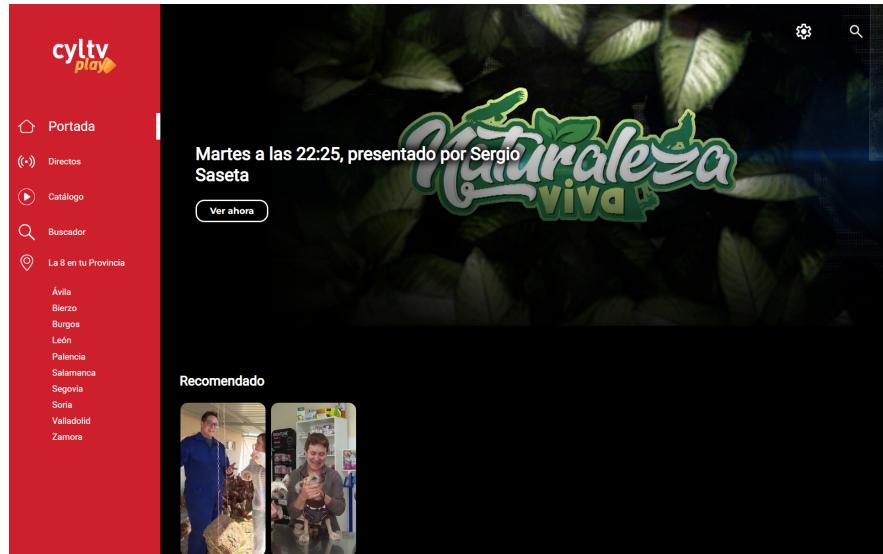


Figura 5.3: Captura de la interfaz casi final

En estas imágenes se puede apreciar la evolución de la interfaz de usuario de *Ott* desde sus inicios hasta la versión casi final. En la primera imagen se puede ver la interfaz en sus primeras fases, con un diseño muy básico y sin apenas funcionalidades. En la segunda imagen se puede ver la interfaz con un diseño más avanzado, pero donde el cliente todavía estaba ajustando diferentes parámetros (vemos que los iconos de los menús no se ven, el menu no contrasta con el fondo, etc). En la tercera imagen se puede ver la interfaz casi final, con un diseño más pulido y con todas las funcionalidades implementadas.

Más que un diseño como tal de la interfaz, lo que se ha hecho es centrarse en los diseños de los diferentes componentes y sus variantes. La estructura general, los tamaños y posiciones de los componentes principales es la misma (de momento) para todos los clientes, pudiendo modificar los colores y los textos. Estos componentes son los siguientes:

- Menú lateral: es el menú que se encuentra a la izquierda de la pantalla y que permite al usuario navegar por las diferentes secciones de la aplicación.
- Menú superior: es el menú que se encuentra en la parte superior de la pantalla y que permite al usuario acceder a las opciones de configuración de la aplicación. Para este componente se pueden llegar a modificar los iconos y los colores de fondo por parte del cliente, pero la posición (depende del número de opciones de este menú) y la funcionalidad es la misma para todos los clientes.
- Barra de búsqueda: es la barra que se encuentra en la pantalla de búsqueda y que permite al usuario buscar elementos en la aplicación. Esta barra utiliza los colores base del cliente para el difuminado y foco, pero la posición y tamaños es fija.

- Lista de elementos: es la lista de elementos que se encuentra en la parte central de la pantalla y que muestra al usuario los elementos disponibles en la aplicación. Estas listas tienen siempre la misma estructura y tamaños. El contenido (títulos y contenidos) es personalizable por parte del cliente, así como el número de listas.
- Detalle de elemento: es el detalle de un elemento que se encuentra en la parte central de la pantalla y que muestra al usuario la información detallada de un elemento. La información es personalizable y es en función de la metadata del contenido, pero la estructura, tamaños y posición de los elementos es fija. En estos momentos se está trabajando para que a través del gestor de contenidos el cliente tenga la capacidad de ordenar y personalizar más esta sección.

El cliente tiene la capacidad de personalizar la interfaz de usuario de *Ott* a su gusto, pudiendo cambiar el color de fondo, el color de los textos, el color de los botones, el color del foco etc. Además, el cliente puede añadir sus propios logos y textos a la interfaz, para que esta se adapte a su imagen corporativa.

Personalización de los componentes

Como ya se ha comentado, el diseño se ha centrado en el diseño de los componentes y sus variantes. Estos componentes podemos dividirlo en dos tipos siendo el enfoque de diseño diferente para cada uno de ellos: los contenidos y los contenedores. Ambos tipos son personalizables en función del "widget" que se esté tratando. Un widget es un componente de la interfaz de usuario que almacena y muestra uno o varios contenidos.

En función del tipo de widget que se esté tratando van a cambiar varios aspectos que afectan tanto a los contenidos como al contenedor en sí. Para cada tipo de widget se debe crear el diseño y desarrollar el código necesario para que este se muestre correctamente en la interfaz de usuario. Los aspectos que afectan a los contenidos y a los contenedores son los siguientes:

- Disposición de los contenidos: en función del tipo de widget que se esté tratando, los contenidos se van a mostrar de una forma u otra. Si el tipo es uno perteneciente a los destacados, los contenidos se van a mostrar de una forma más grande, en la parte alta de la pantalla y con un diseño más llamativo. En función del tipo concreto de destacados este puede tener: botón, un único contenido, varios contenidos (slider), distinta distribución de textos, iconos, etc. Podemos diferenciar 3 grandes cambios en la disposición de los contenidos: destacados, estos se muestran en la parte superior de la pantalla, con un diseño más llamativo y con un tamaño mayor; mosaico, los contenidos se muestran en forma de mosaico, con un tamaño más pequeño y en la parte central de la pantalla; resto, para el resto de widgets los contenidos se mostrarán como una lista horizontal.

- Disposición de la información: en función del tipo de widget que se este tratando, la información se va a mostrar de una forma u otra y se van a mostrar unos campos u otros. Por ejemplo, si el widget es de tipo "destacados" se va a mostrar el título, la descripción y el botón de acceso al contenido. Si el widget es de tipo "banner" únicamente se mostrará el título.
- Forma de las imágenes: las imágenes de los contenidos se van a mostrar de formas diferentes en función del tipo. Podemos destacar 3 casos: destacados, en este caso las imágenes tienen un tamaño muy grande y están pensadas para solo mostrar una en su espacio y la información puede tener distinta distribución en función del tipo concreto; banner, las imágenes son más pequeñas y tienen una orientación horizontal utilizando un ratio de 16:9, la información se muestra justo debajo de la imagen y esta varía en función del tipo concreto; poster, las imágenes son más pequeñas y tienen una orientación vertical utilizando un ratio de 9:16, la imagen se muestra en el lateral derecho en un panel que se despliega cuando el foco está más de un segundo en el contenido.
- Otros: existen más personalizaciones posibles, por ejemplo: si es un directo el widget podrá tener definido que se muestre la barra de progreso y el nombre del programa actual, si es un tipo "banner-click" este widget está enfocado a mostrar un banner publicitario con orientación horizontal ocupando la mayoría del ancho de la pantalla y haciendo click sobre el elemento (en el caso de televisiones esta funcionalidad está desactivada) se redirige a una url externa.

Todas estas personalizaciones y otras muchas más existentes menos utilizadas o en las que se está trabajando, hay que tenerlas soportadas en la aplicación. Como en el caso de las funcionalidades, no podemos saber qué widgets va a utilizar el cliente de antemano, por lo que hay que asegurarse que todos los tipos de widget ofrecidos al cliente están soportados y se muestran correctamente y que no interfieren con el resto de widgets.

Existen algunas consideraciones como que solo puede existir un tipo de widget destacado en la pantalla, que los si un widget de tipo "mosaico" tiene muchos elementos este puede ocupar mucho espacio y dejar de ser cómodo para el usuario, que los widgets de tipo "poster", por lo general, se suelen utilizar para destacar contenidos y para mostrar más información de estos, etc.

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"

5.2.4 Diagramas UML

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 presentará el diseño de la aplicación de análisis de datos. Se describirán los diferentes componentes que forman la aplicación y cómo se relacionan entre sí. Además, se explicará cómo se ha estructurado el código de la aplicación para facilitar su mantenimiento y extensión.

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 de diseño centrada en el usuario. Para ello, se ha realizado un análisis de las necesidades de los usuarios, se han definido los requisitos de la aplicación y se ha diseñado la interfaz de usuario.

Dentro de las necesidades de los usuarios y de los requisitos de la aplicación se destaca la necesidad de que la visualización de los datos sea clara y sencilla y que para utilizar la aplicación no sea necesario tener conocimientos avanzados de análisis de datos.

En cuanto al diseño de la interfaz de usuario, se ha optado por un enfoque sencillo y minimalista, utilizando colores suaves y una tipografía clara y legible. La interfaz ha sido diseñada para ser intuitiva y fácil de usar, permitiendo al usuario acceder rápidamente a todas las funcionalidades de la aplicación. El diseño se centra en la visualización de gráficos y datos, presentando la información de manera ordenada y consistente, para que el usuario pueda comprenderla rápidamente. Todas las gráficas y tablas comparten un estilo uniforme y limpio, evitando elementos recargados que puedan distraer la atención.

De forma similar a la OTT, el diseño de la aplicación de análisis de datos se ha realizado siguiendo un enfoque modular, dividiendo la aplicación en diferentes componentes independientes que se comunican entre sí. Esto permite que la aplicación sea más fácil de mantener y extender, ya que cada componente puede ser modificado o reemplazado sin afectar al resto de los componentes ni a la interfaz. Además, el diseño modular facilita 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 detectar qué tipos de gráficos eran necesarios soportar y diseñar. Tras un análisis de la entrega de los datos que hace Matomo se decidió comenzar con el soporte para gráficos lineales, de barras, circulares y tablas. Así, 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 Lo siguiente era estudiar qué páginas o apartados de la aplicación de análisis de datos se podían implementar. La primera página en la que se pensó fue la página de inicio o dashboard, en la que se mostrarán los gráficos que presenten una información más general y resumida de los datos.

Otra página en la que se pensó durante el análisis fue una página que permitiera al usuario comparar gráficas provenientes de diferentes módulos. La idea era que el cliente tuviera la oportunidad de comparar varias gráficas de diferentes módulos en una misma página para poder analizar la información en busca de patrones, tendencias o correlaciones entre los datos.

El resto de páginas son dedicadas a agrupaciones de datos que tienen relación entre sí. La creación de estas páginas se enfocó en que fueran más dinámica y que permitiera la crea-

ción fácil y rápida de nuevas secciones. Estas páginas están enfocadas a mostrar datos de los visitantes de la web desde diferentes puntos de vista. Para una OTT de estas características, pensada para ser utilizada en varias plataformas y dispositivos, una de las secciones interesantes es la de dispositivos, que muestra información sobre los dispositivos desde los que se accede a la web, mostrando sistemas operativos, marcas, modelos, etc. Otro punto interesante obviamente son las reproducciones, aquí se buscará mostrar información sobre la visualización de los contenidos, como el número de reproducciones, el tiempo de visualización, videos más vistos, etc.

Teniendo claro que enfoques y datos queríamos mostrar, se comenzó con el diseño de las páginas y secciones y la búsqueda de las funcionalidades de la API de Matomo que nos permitieran obtener los datos necesarios para mostrar la información deseada. Tras estudiar estas funcionalidades, y teniendo en cuenta cuales estaban implementadas y optimizadas en todos los códigos de las disitintas aplicaciones de la empresa, se crearon las pantallas de reproducciones, dispositivos y videos y visitas, además de la de inicio y comparador.

- **Inicio:** En esta página se mostrarán los gráficos más generales y resumidos de los datos, como el número de visitas, reproducciones, dispositivos, etc en los últimos instantes. La idea es que el usuario pueda obtener una visión general de los datos en un solo vistazo.
- **Reproducciones:** En esta página se mostrarán los gráficos relacionados con las reproducciones de los videos, como el número de reproducciones, el tiempo medio de visualización, visitantes únicos, etc. La idea es que el usuario pueda analizar la información relacionada con las reproducciones de los videos y obtener insights sobre el comportamiento de los usuarios.
- **Dispositivos:** En esta página se mostrarán los gráficos relacionados con los dispositivos desde los que se accede a la web, como los sistemas operativos, las marcas, los modelos y tipos. La idea es que el usuario pueda analizar la información relacionada con los dispositivos y obtener insights sobre las preferencias de los usuarios.
- **Videos:** Esta página será una lista de todos los contenidos de la plataforma con información para cada uno de ellos. Entre esta información el usuario puede ver el número de reproducciones, el tiempo medio de visualización, tasa de finalización, etc. Además, la tabla permite ordenar los datos en función de los diferentes campos.
- **Visitas:** Esta sección estará compuesta de varias páginas que muestran información sobre las visitas a la web. Las páginas existentes por el momento son: resumen, tiempo, frecuencia de visitas e interés de los visitantes.
- **Comparador:** En esta página se mostrarán los gráficos de diferentes módulos para que el usuario pueda compararlos y analizar la información en busca de patrones, tendencias

o correlaciones entre los datos. La idea es que el usuario pueda obtener insights sobre la relación entre los diferentes módulos y tomar decisiones informadas.

Funcionalidades de la aplicación

El enfoque de la aplicación de análisis de datos es proporcionar al usuario una herramienta sencilla y eficaz para analizar los datos de la OTT. Para ello, además de la visualización de gráficos y tablas, otro de los requisitos de la aplicación es que los clientes tengan la posibilidad de obtener una explicación de los datos y de las gráficas que se muestran. Para esto los usuarios dispondrán de una opción que les permita obtener una breve explicación de lo que se está mostrando en la gráfica y otra opción gracias a la cual al ser seleccionada el usuario obtendrá un análisis más detallado de los datos a través de la API de OpenAi. Esta funcionalidad utilizará un contexto general de la aplicación de la cual se están visualizando los datos y la información de la gráfica para obtener un análisis más detallado y preciso. Este contexto se irá mejorando y detallando con el tiempo y la utilización de la plataforma y será almacenado a través de la API de MongoDB para poder ser suministrado a la API de OpenAi cuando sea necesario.

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 va a desarrollar el proceso de implementación de las funcionalidades básicas de la aplicación OTT. Como ya se indicó en anteriores puntos, las tecnologías utilizadas para el desarrollo de la aplicación son JavaScript, HTML y CSS.

Indicaciones previas

Esta sección va a describir el comportamiento básico de la aplicación cuando está siendo utilizada en televisiones ya que es el enfoque principal de la aplicación en estos momentos y es el enfoque que más desafíos supuso durante el desarrollo del código.

Trabajar en aplicaciones para televisiones tiene una particularidad y es que hay diseñar la aplicación para ser utilizada con un mando a distancia. Aunque en algunas televisiones existe la posibilidad de utilizar el "magic remote" que es un mando a distancia que tiene un puntero que se mueve por la pantalla, la mayoría de las televisiones no tienen esta funcionalidad y desarrollar la aplicación únicamente para televisores con esta opción sería un error ya que limita mucho el mercado al que se puede llegar. Es por eso que la aplicación esta desarrollada

desde un punto de vista que permita ser utilizada con un mando a distancia. Aun así, se ha tenido en cuenta la posibilidad de que la aplicación pueda ser utilizada con un puntero o un ratón en el caso de los ordenadores ya que, aunque no fue un objetivo para las primeras versiones de la aplicación, uno de los objetivos del código es que sea transversal y pueda ser utilizado en cualquier dispositivo.

Creación de los componentes de la aplicación

El primer paso del desarrollo fue la creación de la estructura básica de la aplicación de los primeros componentes que se iban a utilizar y sobre los que se ejecutan las funcionalidades básicas de la aplicación. Estos componentes son los menús, los "widgets" y los contenidos.

La creación de estos componentes es una creación en cascada podríamos decir. Cuando se inicia la aplicación, se llama a través de la API de la empresa a lo que internamente conocemos como "interfaz". Esta interfaz es un JSON que contiene toda la información principal de la aplicación y con la que se dan los primeros pasos de creación de la misma. En este JSON tendremos los colores utilizados a lo largo de toda la aplicación (colores de fondo, de los botones, de los textos, etc), las fuentes de texto, textos legales, imágenes (logos, splash, etc) y demás información que se utilizará para la personalización para cada cliente de la aplicación. Dentro de esta información también se encuentra la información de los distintos menús que va a tener la aplicación.

Para la creación de cada uno de estos menús existen dos opciones: que tengan una pantalla asignada o que sean menús con un comportamiento predefinido. Dentro de los menús con comportamientos predefinidos encontramos los menús de configuración o perfil, de búsqueda y de catálogo. Estos casos tienen un comportamiento específico y no necesitan la información sobre la pantalla que deben mostrar.

- **Configuración:** Este menú alojará las distintas opciones de configuración que tenga cada aplicación. No siempre son las mismas, pero en función de las características de la aplicación se mostrarán unas u otras. Si la aplicación para determinado cliente debe soportar multilenguaje, en este menú se podrá cambiar el idioma de la aplicación. Si la aplicación tiene usuarios registrados, se mostrará la información del usuario y se podrá cerrar sesión. Una funcionalidad que aparece en todas las aplicaciones es la de mostrar los términos y condiciones de la aplicación. Para otras funcionalidades hay que acordar con el cliente pertinente los requisitos y se añadiría para su aplicación la funcionalidad necesaria.
- **Búsqueda:** Este menú alojará la funcionalidad de búsqueda de la aplicación. En este menú se podrá filtrar todos los contenidos de la aplicación por el nombre del contenido.

- **Catálogo:** También conocido como "A la carta" permite mostrar en la misma página todos los contenidos disponibles para ver en la aplicación.

En el caso de los menús que tienen una pantalla asignada, esta pantalla contiene la lista de los "widgets" que se deben mostrar en ella. Estos widgets son contenedores de elementos y cada uno en función de su tipo tendrá un diseño y unas funcionalidades asignadas. A su vez, para cada uno de estos widgets existirá una lista de los 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 consigue la información del menú a través de la API.
- Se analiza para saber 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 consigue 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 consigue la información de los contenidos de cada widget.
 - * Se crea cada uno de los contenidos con las características correspondientes en función del tipo del widget que los contiene.
 - * Se añade cada contenido al widget correspondiente según el orden correspondiente.
 - * Se añade el widget al elemento del DOM que contiene la pantalla.

A lo largo de la creación de cada elemento html se le irá asignando según el tipo de elemento y widget correspondiente una serie de clases que permitirán darle el estilo gracias a 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 mueve 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 poder movernos por ella. Para ello, se ha creado una serie de funciones que permiten moverse por los distintos

elementos de la aplicación. Para este punto se ha tenido en cuenta que la aplicación pueda ser utilizada con un mando a distancia y por lo tanto, ese movimiento debe de estar desarrollado y controlado por el código.

Cuando la aplicación se utiliza en televisión hay que tener en cuenta que se debe tener un foco en un elemento en todo momento. Este foco es el que indica en qué elemento de la aplicación se encuentra el usuario y es el que permite moverse por la aplicación. Para esta aplicación, en las pantallas principales se ha utilizado un foco fijo, es decir, por norma general el foco se encuentra siempre en la misma posición y son los elementos los que se mueven en función de la dirección en la que se mueva el usuario hacia ese foco.

El movimiento por los widgets se realiza de la siguiente manera:

- **Movimiento horizontal:** El movimiento horizontal se realiza con las teclas de dirección izquierda y derecha. Si estamos en un widget, el movimiento se realizará en los contenidos del widget. Cada vez que se pulsa una tecla de dirección, el widget se mueve horizontalmente en función de la dirección en la que se haya pulsado colocando el elemento seleccionado en la posición del foco.
- **Movimiento vertical:** El movimiento vertical se realiza con las teclas de dirección arriba y abajo. Si estamos en una pantalla y todavía quedan widgets por mostrar hacia la dirección en la que se ha pulsado, se desplazará la pantalla hacia arriba o hacia abajo en función de la dirección en la que se haya pulsado hasta colocar el elemento seleccionado en la posición del foco.
- **Movimiento en carrusel:** En el caso de que el widget sea de tipo "featured" y tenga el campo "slider" activado, el movimiento se realizará en el carrusel de los contenidos destacados. En este caso, el movimiento horizontal cambiará la imagen y la información mostrada en el carrusel.
- **Movimiento en mosaico:** En el caso de que el widget sea de tipo "mosaico", el movimiento se realizará en la cuadrícula de los contenidos. En este caso los contenidos aparecen ordenados por filas de izquierda a derecha y de arriba a abajo. El movimiento vertical desplazará la pantalla hacia arriba o hacia abajo en función de la dirección en la que se haya pulsado y el movimiento horizontal en este caso en concreto sí que moverá el foco hacia el elemento seleccionado.

Por otro lado el movimiento por el menú lateral es más simple. Para acceder al menú lateral hay varias opciones: se puede acceder pulsando el botón "return" siempre y cuando estemos en una pantalla principal o se puede acceder pulsando la tecla "izquierda" siempre y cuando no queden más elementos a la izquierda. Una vez seleccionado el menú este se despliega ocupando más sitio y muestra no solo los iconos si no también los nombres de los

menús. Para moverse por el menú se utilizarán las teclas de arriba y abajo y el foco hará que el elemento seleccionado destaque en tamaño.

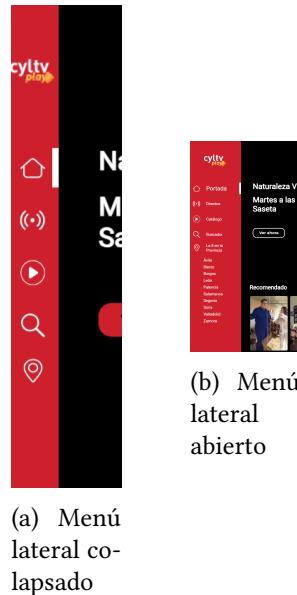


Figura 6.1: Menú lateral

Durante el desarrollo del proyecto se han ido probando diferentes maneras de efectuar el movimiento. Se comenzó moviendo los elementos hacia el foco y más tarde se decidió utilizar las funcionalidades de Html de "scroll", que permite mover elementos sin realizar calculos de posición. Sin embargo, estas dos opciones fueron descartadas en favor de la utilización de las funcionalidades de "transform" de CSS. Tras analizar las tres opciones, la utilización de "transform" casi obligada. Si bien es cierto que la utilización de "scroll" es más sencillo y presenta multiples ventajas, su utilización es menos eficiente y optima y por ello exige un mayor esfuerzo al navegador. En el caso de los ordenadores y dispositivo móviles este esfuerzo no es tan significativo, pero en el caso de las televisiones, que tienen menos recursos, el esfuerzo es mucho mayor y la aplicación se notaba lenta y pesada. La primera opción se descarto debido a que es mas efectivo y eficiente mover la pantalla con todos los elementos que mover cada elemento por separado para reordenarlos.

En futuras implementaciones y con la aplicación mucho más optimizada no se descarta, de hecho se tiene en mente, la posibilidad de estudiar y volver a probar la funcionalidad de "scroll" debido a sus ventajas. Pero, por el momento, se seguirá utilizando la funcionalidad de "transform" de CSS y no es un cambio urgente ni necesario ya que para el usuario de la aplicación no supone una diferencia significativa en cuanto a funcionamiento, pero si en cuanto a rendimiento.

Selección de un componente

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

Estos eventos pueden ser creación de una nueva pantalla (en el caso de seleccionar una opción del menú lateral) o creación de una pantalla de detalle (en el caso de seleccionar un contenido). En el caso de crear una nueva pantalla, se seguirá el mismo proceso que se ha seguido para la creación de la pantalla principal y se mostrará la nueva pantalla en la aplicación, almacenando la pantalla anterior en una pila de pantallas. En el caso de seleccionar un contenido, se creará una pantalla de detalle con la información del contenido seleccionado.

Creación de la pantalla de detalle

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

Existen dos tipos de pantallas de detalle en función del tipo del contenido: los contenedores y los contenidos. Los contenedores son aquellos contenidos que contienen otros contenidos. Por ejemplo, una serie es un contenedor que contiene los capítulos de la serie. Los contenidos son elementos finales, como una película, un capítulo, una partida, etc. Estos no tienen hijos asociados.

La pantalla de detalle de un contenedor está compuesta por una ficha con la información del elemento, una lista de los contenidos que contiene y una lista de los contenedores relacionados. Para moverse por los contenidos hijos haremos uso del movimiento vertical de igual forma que en la pantalla principal y una vez estemos en los contenidos relacionados el funcionamiento es el mismo que el de los widgets de la pantalla principal con el movimiento horizontal.

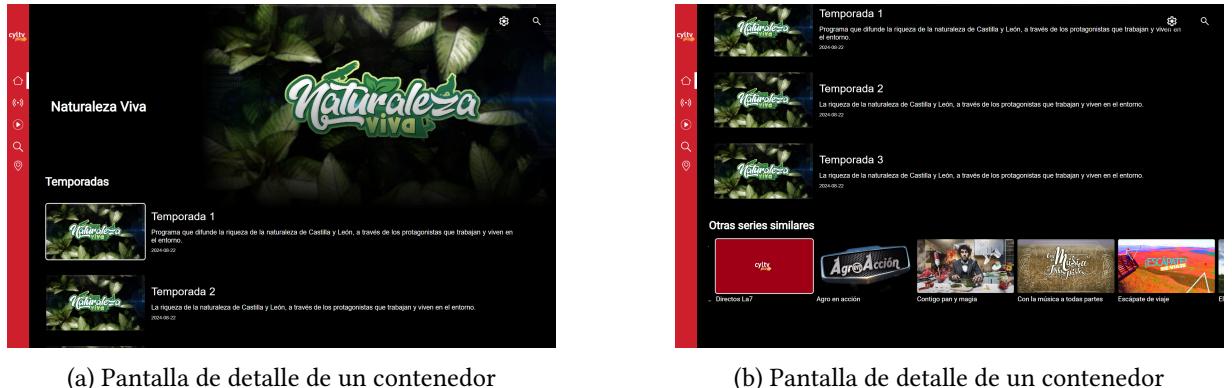


Figura 6.2: Pantalla de detalle de un contenedor

Por otro lado, la pantalla de detalle de contenido está compuesta por una ficha con la información del contenido, donde se incluye el título, contenedor padre (si lo tiene , en el caso de un capítulo de una serie, la serie a la que pertenece), una descripción corta, un botón que permite mostrar un popup con todo la información al completo, los iconos de rating y edad, un botón de reproducción y en caso de permitir la funcionalidad, un botón de añadir a favoritos. También tiene 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 de 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 autentificación para ser reproducido y en caso afirmativo (el usuario ya debería estar logueado) se comprueba si el usuario tiene permisos para ver el contenido. Es una comprobación de seguridad ya que en caso de no tener acceso al contenido el botón de reproducción aparece como deshabilitado. En caso de tener acceso,

el se realizan varias comprobaciones:

Origen del contenido: Lo primero es comprobar de donde proviene el contenido. Aquí hay dos opciones: o esta alojado en la base de datos de la empresa o en la del cliente. Si esta en la del cliente la url del video estará ya en la información del contenido. En caso de estar en la base de datos de la empresa, hay que realizar una serie de comprobaciones para obtener la url del video. Estas comprobaciones son en base a si el contenido es gratuito o de pago ya que si es gratuito la url se podrá montar directamente con la información que se tiene, pero si es de pago hay que realizar una serie de comprobaciones para obtener la información.

Reproductor: No todos los dispositivos pueden usar el mismo reproductor. Los ordenadores son más permisivos con este detalle, pero las televisiones no. En el caso de los ordenadores existe la opción incluso de que la API nos devuelva un player ya construido para utilizar directamente en la aplicación. Este caso no se utiliza por el momento. Lo hace el código es detectar en qué SO y con qué contenido se está trabajando. Aquí tenemos por el momento dos reproductores (VideoJs y ShakaPlayer) y dos tipos de video (VoD y Lives o Youtube). Para cada caso se monta el reproductor de una manera específica.

Una vez el video se está reproduciendo, el usuario dispone de las funcionalidades básicas para controlar la reproducción: pause, play, adelantar, retroceder y salir. En caso de permitir la funcionalidad y tener usuarios registrados la aplicación guarda el instante en el que se detuvo la reproducción para que el usuario pueda continuar viendo el contenido en el mismo punto en el que lo dejó.

Añadir a favoritos

En caso de permitir usuarios registrados, la aplicación permite añadir a favoritos los contenidos. Para ello, en la pantalla de detalle de contenido se muestra un botón en forma de corazón que permite añadir el contenido a favoritos. En caso de que el contenido ya esté añadido a favoritos, el botón aparecerá en color rojo y pulsando sobre él se eliminará de la lista de favoritos.

Búsqueda de contenidos

La búsqueda de contenidos es una funcionalidad básica de la aplicación. Para acceder a ella se puede hacer de dos formas: a través del menú lateral o una opción situada en la parte superior derecha de todas las pantallas. Al abrir el menú de búsqueda, se muestra un campo de texto en el que se puede introducir el nombre del contenido que se desea buscar. Una vez escrito el nombre completo o parcial del contenido y hacer click en el botón de búsqueda, la aplicación realiza una llamada a la API con el nombre del contenido y obtiene una lista de los

contenidos que coinciden con el nombre introducido. Esta lista se muestra en una pantalla de resultados de búsqueda en forma de mosaico.

Obtención de la información de un directo

La obtención de la información de un directo es una funcionalidad que permite obtener la información de un directo en tiempo real. Para ello, la aplicación realiza una llamada a la API con el identificador del directo y obtiene la información del directo en tiempo real a través de un epg. Este epg es un JSON que contiene la información de los contenidos que se van a emitir en un periodo de tiempo determinado. La aplicación analiza este Json y obtiene la información del directo en tiempo real. De esta información se obtiene el título que se mostrará en el contenido correspondiente, y el progreso, que se calcula a partir de los tiempos de inicio y fin del contenido marcados en el Json, y se muestra en la barra de progreso del contenido.

6.2.2 Adaptación a los diferentes sistemas operativos de televisión

El mercado de las televisiones inteligentes está sufriendo en los últimos años un gran crecimiento, lo que provoca que las opciones de aplicaciones y servicios que se pueden ofrecer a los usuarios sean cada vez mayores. Sin embargo, es un mercado relativamente nuevo y al que se le está exigiendo prestaciones similares a las que se pueden encontrar en los dispositivos móviles o ordenadores personales. Sin embargo, cumplir 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 que se pueden encontrar en el mercado.

Capacidad de procesamiento de las televisiones Aunque la mejoría y el aumento de la capacidad de procesamiento en los últimos años ha sido más que notable, no es suficiente para poder ofrecer una experiencia de usuario similar a la que se puede encontrar en un ordenador personal o un dispositivo móvil. Tampoco ayuda que las tecnologías web sean cada vez más pesadas y complejas y que el resto de dispositivos sí que puedan soportarlas. Si comparamos la capacidad de procesamiento de una televisión con la de un ordenador personal o de un dispositivo móvil, podemos ver que una televisión de alta gama nueva tiene una capacidad de procesamiento que, aunque impresionante para su categoría, aún se queda corta en comparación con otros dispositivos de consumo masivo.

Las características que ofrece una televisión de alta gama nueva (3/4 GB de RAM, procesador de 4 núcleos...) son las mismas características que ofrecían hace ya unos años los dispositivos móviles de gama media-alta (hoy cuentan con 6/8 GB de RAM y procesadores de 8 núcleos) y los ordenadores más básicos (hoy no bajan de 8/16 GB de RAM).

Soporte de los sistemas operativos a las tecnologías web Este problema en parte está relacionado con el anterior. Los sistemas operativos de las televisiones inteligentes no están tan adaptados a las tecnologías web como otros casos. Esto puede ser debido a varios motivos, como la capacidad de procesamiento que se comentaba o la entrada tardía de las televisiones en el mercado de los dispositivos inteligentes y por lo tanto a la oferta de aplicaciones y servicios.

Diversidad de sistemas operativos Otro problema que se encuentra en el mercado de las televisiones inteligentes es la diversidad de sistemas operativos que se pueden encontrar. Aunque en el mercado de los dispositivos móviles también se pueden encontrar varios sistemas operativos, la mayoría de los dispositivos móviles cuentan con Android o iOS, lo que facilita la tarea de los desarrolladores a la hora de crear aplicaciones y servicios. En el caso de las televisiones, la diversidad de sistemas operativos es mucho mayor, lo que provoca que los desarrolladores tengan que adaptar sus aplicaciones y servicios a cada uno de los sistemas operativos que se encuentran en el mercado.

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 adaptarse a las capacidades y limitaciones de cada SO. Si un SO no soporta ciertas funcionalidades, lo ideal es buscar una alternativa conjunta 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 adapte a cualquier resolución de pantalla. Esto es necesario si queremos utilizarlo en dispositivos de distintas familias, pero también en televisores de distintas marcas y modelos. Un ejemplo de ello son las televisiones que se están utilizando en este proyecto como televisores de pruebas. La televisión LG y la televisión Samsung utilizadas tienen la misma resolución, 1920x1080, mientras que la TCL (Android TV) tiene una resolución de 960x540. Las tres televisiones son nuevas y de gamas similares, pero la resolución de la TCL, que es la que mejor resolución de reproducción de video tiene, es la mitad que las otras dos. Y esto son 3 televisiones que se toman de ejemplo, pero si se quiere llegar al mayor número de usuarios posible, hay que asegurarse que la aplicación es consistente en cualquier televisión que se pueda encontrar en el mercado.

Para conseguir esto, todo el diseño y todas las hojas de estilo de CSS utilizan medidas relativas: porcentajes, "vh" y "vw". Se evita a toda costa el uso de medidas absolutas, como "px" (no se utiliza nunca en este código), y también se ha evitado utilizar "rem" y "em" ya que con diferencias tan grandes no se puede asegurar que el diseño sea consistente en todos los dispositivos.

Soporte de tecnologías web El soporte de tecnologías web es un punto importante a tener en cuenta. Como ya se explicó en otras secciones, las tecnologías web utilizadas para este proyecto son JavaScript, HTML y CSS. Estas tecnologías son soportadas por todos los SO utilizados, pero no al completo. A lo largo del desarrollo se ha intentado utilizar las mínimas librerías posibles para asegurar la mantenibilidad del código y problemas futuros con las actualizaciones de las librerías. Pero estos no son los motivos por los que no se han utilizado librerías, existen soluciones a esos problemas en caso de necesitar las librerías. El caso es que estas librerías pueden no ser compatibles con los SO utilizados y, ante el desconocimiento en algunos casos y la comprobación de que no funcionan en otros, se ha optado por no utilizarlas. El día que se está escribiendo este documento sin ir más lejos, se ha tenido que eliminar el uso de la funcionalidad de DOMParser, ya que no es soportada por Tizen, y se tuvo que implementar a mano una solución para poder leer los archivos XML que se reciben con la información de los directos.

Navegación Este es un ejemplo de adaptación específica para cada SO. Los comandos que recibe la aplicación cuando se pulsa un botón del mando no son iguales en todos los dispositivos, no varían mucho, pero hay diferencias y se ha optado por hacer una adaptación específica para cada SO. Para cada SO existe un archivo de configuración con los códigos de los botones del mando y estos se traducen a un formato común que entiende la aplicación para realizar las acciones correspondientes.

Reproducción de video La reproducción de video es una de las partes más importantes de la aplicación y por lo tanto hay que asegurar que funciona correctamente, sino la aplicación no tendría sentido. A diferencia de las páginas web, en las que hay mayor libertad de elección del reproductor de video, en el caso de las televisiones inteligentes estas opciones son más limitadas. En nuestra aplicación en concreto, los SO utilizados hasta el momento no utilizan el mismo reproductor, y es que a diferencia de WebOs y AndroidTv, Tizen no soporta el reproductor VideoJs y por el contrario utiliza Shaka Player, el cual no es soportado por los otros dos SO. Este es otro ejemplo de adaptación específica ya que para solucionar este problema la solución pasa por detectar el SO y crear un reproductor de video u otro.

Detección del estado de la red La detección del estado de la red es una parte importante de la aplicación, ya que si no hay conexión a internet no se puede acceder a los contenidos. En este caso, la detección del estado de la red es una adaptación específica para cada SO. Tras intentar utilizar las mismas funcionalidades para todos los SO, la detección del estado de red no era consistente en todos los casos, por lo que hubo que buscar una solución específica para cada SO. En el caso de Tizen se utiliza la librería webApis, en el caso de AndroidTv y webOs se permite utilizar las funcionalidades de navigator y más concretamente navigator.connection.

Cerrar la aplicación Como era de esperar, la forma de cerrar la aplicación también es diferente en cada caso. Para ello, se buscó información en la documentación correspondiente y se desarrolló la aplicación para detectar el SO y cerrar la aplicación de la forma correcta en cada caso.

Desafíos encontrados

A lo largo del desarrollo de la aplicación se han encontrado varios desafíos que han complicado la adaptación de la aplicación. A continuación se detallan algunos de los desafíos más importantes/raros encontrados y cómo se han solucionado.

El primero de los desafíos encontrados fue la falta de información y de soluciones sobre problemas en este tipo de aplicaciones. La mayoría de la información que se encuentra en internet sobre aplicaciones para televisores inteligentes es proviene de las documentación oficial que en muchos casos es escasa y no cubre todos los casos posibles. Los foros y comunidades de desarrolladores son una buena fuente de información, pero en muchos casos no se encuentran soluciones a los problemas que se pueden encontrar en el desarrollo en estos dispositivos.

Foco automático en AndroidTv Cuando se comenzó a probar la aplicación en AndroidTv, se encontró un problema un tanto peculiar. La aplicación estaba detectando elementos auto-

máticamente y con cada movimiento del mando no solo realizaba el movimiento esperado, si no que buscaba otro elemento en el que enfocarse y podía provocar movimiento indeseados al desplazar la pantalla o mover otros elementos. Para solucionar este problema había que desactivar ese foco. Se buscaron varias soluciones: utilizar la propiedad "tabindex", probar distintas combinaciones z-index de los elementos, estudiar las configuraciones internas y las APIs que utiliza Cordova para AndroidTV, pero ninguna de ellas funcionó. La solución fue utilizar un "listener" que detectara cuando un elemento obtenía el foco y lo desactivara inmediatamente. En las televisiones el foco que se utiliza es un foco "artificial" ya que al elemento enfocado se le añade la clase "focused" y no se utiliza la propiedad "focus" de HTML. Esta función que desactivaba el foco se ejecutaba en el evento "focusin" y tuvo que ser ajustada más tarde para permitir el foco en los elementos que lo necesitaban como la caja del buscador.

Botón de retorno con funcionamiento predefinido en AndroidTv Otro de los problemas encontrados en AndroidTv fue el funcionamiento del botón de retorno. En la última actualización de AndroidTv, el botón de retorno sufrió un cambio en su funcionamiento. Al ser pulsado, si no encontraba ninguna página anterior en el historial, cerraba la aplicación. Esto no era el comportamiento esperado, ya que esta aplicación usa un diseño de página única, solapando las páginas en lugar de cargar una nueva. Para solucionar este problema, se tuvo que añadir un "listener" al botón de retorno que detectara cuando se pulsaba y que cancelara el evento por defecto, evitando que se cerrara la aplicación. El problema fue que conseguir interceptar el evento antes de que se cerrara la aplicación no fue sencillo, y hubo que probar varias soluciones hasta encontrar la correcta.

Splash screen en AndroidTv Otros de los problemas encontrados con Cordova en AndroidTv fue que desde las últimas actualizaciones implementa un "splash screen" por defecto que no se puede desactivar. Este "splash screen" se muestra al inicio de la aplicación y justo a continuación el splash propio de la aplicación. Esto no es un problema en sí, pero el "splash screen" de Cordova tiene una personalización muy limitada y compleja y por el momento solo se ha conseguido cambiar la imagen de que se muestra, pasando del logo de Cordova al logo de la aplicación. Pero aún así, no es una solución definitiva ya la imagen se ve con un formato extraño. Se ha intentado desactivar el "splash screen" y modificarlo de todas las formas posibles que indican tanto en la documentación como en los foros, pero conseguir el resultado al 100%

6.3 Implementación de la aplicación de análisis de datos

6.3.1 Instalación, configuración e integración de la API de Matomo

Para la construcción y desarrollo de la aplicación de análisis de datos, se ha utilizado la herramienta de análisis web Matomo. Aunque existen otras herramientas de análisis web, como Google Analytics, para un primer desarrollo se ha optado únicamente por esta herramienta, ya que es la 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 sobre los visitantes de sus aplicaciones y páginas web. Ofrece una amplia gama de funcionalidades, como la recopilación de datos, la generación de informes y la personalización de los mismos. Una de las funcionalidades que ofrece es la utilización de una API REST, que permite a los desarrolladores acceder a los datos recopilados por Matomo y realizar operaciones sobre ellos.

Con estas consultas, se puede obtener la información necesaria para la construcción de la aplicación de análisis de datos. A través de la API se obtienen los datos que servirán como entrada para la construcción de las gráficas y tablas.

Matomo ofrece una rápida y sencilla instalación e integración con las distintas aplicaciones en las que se requiera la recogida de datos. Los pasos a seguir son:

1. Darse de alta 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.

Con estos pasos, Matomo comenzará a recopilar los datos de los visitantes de la aplicación. Si se desea obtener información más detallada y concreta existen más opciones de configuración, como la creación de eventos personalizados, la configuración de objetivos o la creación de segmentos. Para conocer datos sobre las reproducciones de los vídeos, habrá que configurar el reproductor de vídeo para que envíe 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 Matomo ya está recopilando los datos, se puede comenzar a realizar consultas a la API para obtener la información necesaria. Para ello, se debe 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 en la que se está realizando la consulta. Con esta configuración lista, y conociendo los datos que se quieren obtener 5.3.2 y a través de qué métodos, se puede comenzar 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 puede comenzar a realizar consultas a la misma para obtener la información. Esta información será la que se utilizará para la construcción de 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 utilizado la tecnología de React, una biblioteca de JavaScript para la creación de interfaces de usuario. Complementado con el uso de Html y Css y 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 encargado de la configuración de la aplicación. Gracias a este módulo la aplicación crea las URLs necesarias y realiza las consultas a la API de Matomo. Cada URL está formada en función del módulo correspondiente de Matomo, la acción pertinente y las variables necesarias para la consulta. También existen configuraciones para indicar que gráficos y datos se muestran en cada página.
- **Consultas:** Módulo encargado de realizar las consultas a la API de Matomo. En este módulo se encuentran las funciones que realizan las consultas a la API y devuelven

los datos obtenidos. Para cada acción se intenta llamar al Módulo API y a la acción getProcessedReport para obtener toda la información que nos ofrece Matomo acerca de cada acción.

- **Procesamiento:** Módulo encargado de pedir y procesar los datos obtenidos para crear las distintas páginas de la aplicación. Para cada página se buscarán los datos y llamadas asignadas para la creación de la página, se pedirán los datos y se enviarán a los componentes de gráficas y tablas para su creación y ordenará los resultados de estos componentes para mostrarlos en la página.
- **Componentes:** Dentro de este módulo se encuentran los componentes de gráficas y tablas que se utilizan en la aplicación. Cada componente es un archivo independiente que se encarga de la creación de una gráfica o tabla en concreto.
- **Diseño:** En este módulo se encuentran las hojas de estilo de Css que se utilizan en la aplicación para darle un diseño más atractivo y usable.

Desarrollo de la aplicación

Para el desarrollo de la aplicación haciendo uso de la API el primer paso tras tener configurado y listo Matomo fue obtener un servidor donde alojar la aplicación ya que por seguridad el API bloquea las consultas que se realizan desde un servidor local. Para ello se utilizó el servicio de Netlify [14], un servicio de alojamiento de aplicaciones web que permite desplegar aplicaciones de forma sencilla y rápida.

Una vez alojada la aplicación, se comenzó a desarrollar la aplicación. Para ello se comenzó con los archivos de configuración los cuales se encargan de crear 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, a través de la configuración de la página pertinente y con el formato adecuado y unificado se envían a los componentes de gráficas y tablas para su creación. Una vez creados los componentes, estos son añadidos a la página y se muestran 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 hay que crear, se creo el componente GraphRenderer que recibe los datos y el tipo de gráfica que se quiere crear y se encarga de llamar al componente correspondiente para la creación de la gráfica. Este componente se creo como puente entre los datos y los componentes de gráficas y tablas para favorecer 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 han utilizado la librería de Chart.js [15]. Esta librería permite la creación de gráficos y tablas de forma sencilla y rápida. Ofrece una amplia gama de gráficos y tablas que se pueden personalizar y adaptar a las necesidades de la aplicación. Es muy flexible lo que permite la creación de gráficos y tablas adaptado a las necesidades de la aplicación. Se han utilizado los componentes Line, Bar, Pie y Bubble por el momento para la creación de las gráficas.

Ejemplo de creación de un grafico lineal

```

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

```

```

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 aplicaciones se han creado de forma dinámica. Para cada página existe un archivo de configuración que indica qué métodos y acciones se deben realizar para la creación de la página. Existe otro archivo con la información básica de cada gráfica por si la llamada a API.getProcessedReport no devuelve la información necesaria y con la función para obtener los datos correspondientes para crear 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 de la aplicación sin tener que pasar los datos a través de props. Se crea un contexto y se envuelve la aplicación en un componente que provee el contexto. Los componentes utilizarán el contexto para acceder a los datos.

En el caso de esta aplicación, se ha creado un contexto para la detección del id del sitio web que se está analizando. Este id se utiliza para realizar las consultas a la API de Matomo y obtener los datos correspondientes a ese sitio web.

Ejemplo de uso de Context:

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

Hooks es una API que permite a los componentes de la aplicación utilizar el estado y otras características de React sin tener que escribir una clase. Se utilizan para la gestión del estado de los componentes y para la comunicación entre los distintos componentes de la aplicación.

Los hooks más utilizados en la 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

Conforme se iba desarrollando la aplicación, se iban realizando pruebas para comprobar que todo funcionaba correctamente. Para realizar estas pruebas se utilizó postman, una herramienta que permite realizar peticiones a una API y comprobar que los datos devueltos son los correctos y el servicio de Netlify [14] que permite desplegar aplicaciones de forma sencilla y rápida para comprobar que la aplicación se desplegaba correctamente. Por el momento la aplicación sigue alojada en Netlify y se puede acceder a ella 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 facilitaron el desarrollo de la misma. Estas librerías establecen una serie de funciones y métodos que permiten realizar operaciones de forma más sencilla y rápida que si se tuviesen que implementar desde cero.

Una de estas librerías es *Axios*. Esta librería facilita la realización de peticiones HTTP desde el cliente a un servidor. Esta popular herramienta está basada en promesas que simplifica la comunicación con APIs. Permite realizar solicitudes *GET*, *POST*, *PUT*, *DELETE*, entre otras, de manera eficiente y con un manejo simplificado de las respuestas y errores. Su facilidad de uso y su capacidad para manejar tanto solicitudes asíncronas como configuraciones avanzadas, como el establecimiento de cabeceras personalizadas y la gestión de tiempos de espera, han sido fundamentales 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 la 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 cuyo uso se ha explicado en la sección 6.3.2. Esta librería permite la creación de gráficas de forma sencilla y rápida. Ofrece una amplia variedad de gráficos, como barras, líneas, radar, polar, entre otros, y permite personalizar los con colores, títulos, leyendas, entre otros.

APIs : OpenAi y MongoDB

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

Esta API utiliza tanto el contexto de la página como los datos de la gráfica a analizar para generar una explicación en lenguaje natural y comprensible para cualquier usuario sin conocimientos técnicos. Para ello, hay varios elementos necesarios: contexto, datos y petición.

Contexto: Información general de la página web. Este contexto se va creando y detallando con el tiempo ya que cada vez que se realiza una petición a la API, además del análisis, se pide que se actualice ese contexto para siguientes peticiones para poder dotar a la explicación de un mayor detalle y precisión.

Datos: Datos de la gráfica a analizar. Estos datos son los que se recopilan a través de la API de Matomo y se envían a la API de *OpenAi* para su análisis.

Petición: Promt generado a partir de información de la página (Contexto) y los datos de la gráfica (descripción, variables, valores, etc). Este promt se envía a la API de *OpenAi* para que genere una explicación en lenguaje natural.

Tanto el contexto como los datos son almacenados para alimentar a futuras llamadas y crear prompts más precisos y detallados. Para almacenar estos datos se ha utilizado una base de datos no relacional, en concreto *MongoDB*.

MongoDB: MongoDB es una base de datos no relacional que permite almacenar datos en formato JSON. Es una base de datos escalable y flexible que permite almacenar grandes cantidades de datos y realizar consultas de forma rápida y eficiente. Para la integración de MongoDB con la aplicación de análisis de datos se ha utilizado el servicio en la nube de MongoDB Atlas. Este servicio ofrece una base de datos en la nube que permite almacenar y consultar datos de forma segura y eficiente. Además, MongoDB Atlas ofrece una serie de funcionalidades avanzadas, como la replicación de datos, la recuperación ante desastres y la escalabilidad automática, que garantizan la disponibilidad y el rendimiento de la base de datos.

MongoDB Atlas se ha utilizado a través de su API para almacenar los datos de contexto y datos de las gráficas para su posterior análisis y generación de explicaciones.

API de MongoDB

```

1  const BASE_URL = 'https://eu-west-2.aws.data.mongodb-api.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  const config = {
15    method: 'post',
16    url: `${BASE_URL}/findOne`,
17    headers: {
18      'Content-Type': 'application/json',
19      'api-key': API_KEY,
20    },
21    data: data
22  };
23
24  try {
25    const response = await axios(config);
26    return {
27

```

```
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 } ;
```

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. Dado 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, incluyendo 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 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 mantenía un comportamiento óptimo bajo diferentes cargas y en distintos dispositivos.

Enfoque de Pruebas Continua

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 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 funcionalidades 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 televisiones con alguno 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 integración y adaptación a 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 de móvil y tableta 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 de 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 el cual quiero llevar a cabo a muy corto plazo es la implementación en la personalización que podemos ofrecer a 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 unos de otros, siempre procurando reducir el tiempo de desarrollo y mantenimiento de la aplicación.

Se continuarán implementando funcionalidades de productos de la empresa como puede ser la previsualización de los contenidos, el cual ofrece a los usuarios un video corto con un breve resumen de un contenido, generado este a través de inteligencia artificial. Este producto está siendo utilizado en estos momentos 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 aplicación. Por el momento está habilitada para el 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/>