UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

StreamBudget - Aplicación móvil para obtener el servicio de streaming más económico en base a lo que quiere visionar el usuario

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Fernández Mondéjar, Adrián

Tutor/a: Andrés Martínez, David de

CURSO ACADÉMICO: 2021/2022

A Rosa y Noa, por su apoyo y acompañamiento

en los momentos donde no lo creía posible.

Resumen

En la actualidad cada vez más empresas crean sus propios servicios de streaming enfocados al entretenimiento. Podemos encontrar cada vez más servicios que nos ofrecen música, películas, series y hasta videojuegos, creando una fragmentación de contenido que cada vez hace más difícil elegir qué contratar. Para este proyecto se desarrollará una aplicación Android enfocada en los servicios de películas y series, que permita una vez indicadas nuestras preferencias buscar el servicio más rentable para suscribirse.

Palabras clave: streaming, Android, app, video, películas, series.

Resum

En l'actualitat cada vegada més empreses creen els seus propis serveis de streaming enfocats a l'entreteniment. Podem trobar cada vegada més serveis que ens ofereixen música, pel·lícules, sèries i fins a videojocs, creant una fragmentació de contingut que cada vegada fa més difícil triar què contractar. Per a aquest projecte es desenvoluparà una aplicació Android enfocada en els serveis de pel·lícules i sèries, que permet una vegada indicades la nostra preferències buscar el servei més rendible per a subscriure's.

Paraules clau: streaming, Android, app, vídeo, pel·lícules, sèries.

Abstract

Nowadays, more companies are creating their own streaming services focused on entertainment. We can find increased services that offer music, movies, series and even video games, creating a fragmentation of content that makes it increasingly difficult to choose a subscription plan. For this project we will develop an Android application focused on movie and series services, which will allow, once we indicate our preferences, to search for the most profitable service to subscribe to.

Keywords: streaming, Android, app, video, movies, series.

Índice de contenido

Índice de términos

T

Índice de figuras

Índice de tablas

1. Introducción

La industria del entretenimiento digital ha evolucionado hacia un consumo bajo demanda, al alcance de todos y en cualquier lugar. Centrándonos en contenido multimedia, concretamente en películas y series, hoy día es difícil no conocer a alguien que no tenga al menos una plataforma de *streaming*, ya sea porque la contrata o porque la toma prestada de amigos o familiares.

Si se compara con un tiempo relativamente cercano, hace 20 años, para poder ver una película de estreno se debía acudir físicamente al cine. Películas con más tiempo o las series podían verse en televisión, sin poder elegir cuando verlo y no siempre habiendo algo que fuera de interés. Si bien había cines, como los de verano, que también proyectaban películas, aunque no siempre tan nuevas, en general la elección para ver estos contenidos estaba limitada, especialmente en el caso de las series.

Por supuesto siempre existía la opción de comprar o alquilar estos contenidos en un medio físico, inicialmente el VHS y posteriormente en DVD y Blu-Ray. El alquiler estaba relativamente normalizado, existiendo los videoclubs que eran tienda básicamente especializadas en estos alquileres.

Volviendo al presente, hoy es raro ver un videoclub en activo y, aunque no han desaparecido, los soportes físicos han visto su uso reducido. En muchos hogares ya no se cuenta con equipos dedicados a la reproducción de estos medios, aunque cabe decir que las consolas de videojuego actuales son capaces de reproducirlos, actuando de forma secundaria como reproductores.

Desde la penetración en el mercado español por parte de Netflix en 2015 y a medida que las redes de datos móviles mejoraban su capacidad y la normalización de tener un dispositivo móvil, la gente comenzó a ver una alternativa muy rentable para consumir este contenido. Con el tiempo y el avance de los dispositivos inteligentes, no solo estaban disponibles en los móviles, sino también en tables o televisores inteligentes (SmartTV).

Teniendo las ventajas de poder elegir que ver igual que al alquiler y el hecho de que al precio de una o dos entradas de cine se dispusiera de un mes entero para poder visualizar contenido sin límites, los servicios de *streaming* comenzaron a comer terrero a los medios más tradicionales e incluso a la piratería ilegal de estos contenidos, ya que se percibía como más costoso el hecho de piratearlos que simplemente pagar la cuota.

Hoy día muchas empresas han querido “su trozo del pastel”, habiendo proliferado los servicios de *streaming* en número, resultado a veces algo tedioso elegir qué servicio contratar. Esto también hace que los servicios que entran al mercado de manera tardía suelan ofrecer ciertos atractivos, ya sea con precios más bajos, con contenido exclusivo. En el caso del servicio Disney+, se llegó al extremo de ofrecer las nuevas películas únicamente en su servicio de *streaming*, no llevándolas al cine tradicional.

En este mercado actual con tantas opciones, precios distintos y contenidos solapados, nace la idea de StreamBudget. StreamBudget es una aplicación disponible para Android cuya principal función consiste en ayudar al usuario a elegir la o las plataformas de *streaming* que más rentabilidad le van a ofrecer por su suscripción.

Para esto, en base a una selección previamente realizada por el usuario sobre los contenidos que le gustaría visualizar, los precios actualizados de cada servicio de *streaming* y una lista de los contenidos disponibles en cada plataforma de *streaming*; realizar una serie de cálculos para ofrecer al usuario la solución óptima según el modo seleccionado. Cada modo se basa en algoritmos distintos y tiene en cuenta distintos parámetros a la hora de realizar los cálculos.

Motivación

A nivel personal, tengo interés por el desarrollo en Android y una tendencia a la frugalidad, lo que me hizo buscar desarrollar una aplicación que permitiese optimizar gastos. Últimamente y con el creciente número de servicios de *streaming*, empezó a molestarme tener que pagar cada vez más servicios y pensando en otras situaciones donde no hubiera podido dedicar tanto presupuesto nació la idea de StreamBudget. Además de esto, siempre me ha resultado curiosa la relación entre la piratería y el coste apreciado de algo y aunque no sería el principal objetivo, podría ayudar a combatir la piratería al hacer más sencillo encontrar los servicios que tengan más valor para el usuario.

A nivel profesional, "*(…) en el año 2013 el mercado de los Smartphones estuvo dominado por Android, con un 75% de cuota de mercado (…)*" [1, p. 38] y de manera general en España en los últimos 8 años más del 80% de los dispositivos usaban Android [2] por lo que es sin duda el perfecto candidato para el desarrollo de una nueva aplicación orientada a dispositivos móviles. En los últimos años cada vez más empresas quieren tener su propio servicio de *streaming* y hay cierto sentimiento creciente de vuelta a la piratearía. Podría existir una relación clara entre este sentimiento y la pérdida del valor percibido de contratar un servicio de *streaming*, ya sea por fatiga decisoria o por un incremento del gasto al tener que contratar varios servicios. No es descabellado asumir que una aplicación que elimine la fatiga decisoria y ayude a optimizar el presupuesto puede tener una buena aceptación en la época actual.

Objetivos

- Desarrollar una aplicación para dispositivos móviles que permita de manera sencilla saber el mejor servicio en función del coste percibido.
- Implantar varios algoritmos de ordenación según coste que el usuario pueda elegir según su preferencia.
- Desarrollar un *backend* que mantenga la cache de los catálogos de los distintos servicios de *streaming*, el cual será consultado por la aplicación móvil.
- Implantar las siguientes funciones adicionales, en función de los tiempos disponibles y el interés obtenido de una encuesta a posibles usuarios:
    - Perfiles de usuario y sistema de actualizaciones mensuales con los resultados.
    - Características de red social donde poder seguir otros perfiles, ver el contenido que tienen marcado para ver y capacidad de mensajería entre usuarios.

- Buscador de ofertas para los servicios de *streaming*
- Calendario de cambios de contenido en el catálogo (adiciones y eliminaciones).
- Buscador de grupos para poder compartir la suscripción con otros usuarios.

Metodología

Como metodología de desarrollo de software se ha elegido el sistema tradicional en cascada (o waterfall en inglés). Este sistema se basa en un proceso secuencial estricto, en el que las fases se van realizando una detrás de otra, como fluyendo en una cascada, obteniendo de ahí su nombre. Las fases principales de esta metodología se describen brevemente en Fig. 1:

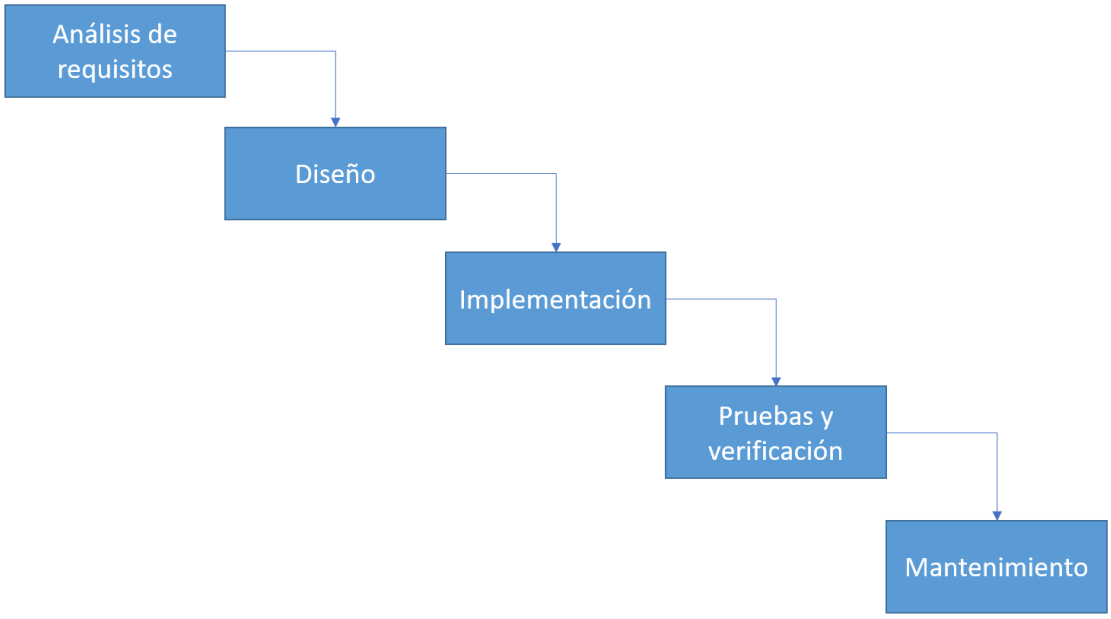

Fig. 1 – Modelo de desarrollo en cascada

Como podemos ver en Fig. 1, tras cada salto en la cascada se produce un proceso de verificación, pudiendo retornarse a la fase anterior en caso de no superarse. Del mismo modo, estando ya en la fase de mantenimiento es posible incorporar nuevas funcionalidades (evolutivos) o mejoras (perfectivos) comenzando desde el inicio de la cascada y versionando el software.

En este tipo de metodología es muy importante tener un análisis y diseño robustos, ya que, por su tipología, un error en las últimas fases es mucho más costoso de corregir que en las iniciales. Siguiendo el símil de su nombre, cuanto más abajo nos encontremos en la cascada, más dificultad tendremos para remontarla.

Aun utilizándose la metodología de cascada, se aprovecharán algunos conceptos más propios de las metodologías agiles como el método MoSCoW para priorizar requisitos en las fases de análisis.

Respecto a la arquitectura del código, se utilizará el modelo de programación por capas, representado en Fig. 2. Este modelo se basa en desacoplar en distintas capas las partes que componen el software, siendo el más habitual el sistema de tres capas, donde:

- La capa de presentación contiene las interfaces gráficas y es la capa con la que el usuario final interacciona.
- La capa de negocio, también llamada capa de lógica contiene los procesos para realizar las distintas operaciones de la aplicación. Comunica con la capa de presentación para recibir las instrucciones del usuario y enviarles los resultados y comunica con la capa de persistencia para consultar datos o guardarlos en memoria persistente.
- La capa de persistencia es la capa encargada de la interacción con los sistemas de datos para obtener o guardar información.

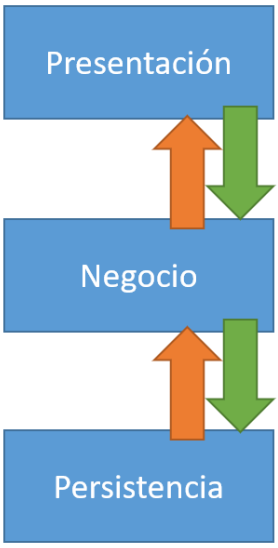

Fig. 2 – Modelo de Programación por capas

Impacto Esperado

El usuario podrá evitar la fatiga decisoria, un problema cada vez más acentuado en la actual sociedad de consumo acelerado donde a diario se deben tomar cada vez más decisiones. Esto se alcanzaría al delegarse en la aplicación las comparaciones, dando respuesta a qué servicio o servicios serían los óptimos contratar. También podría suponer una reducción de gastos económicos para aquellos usuarios que actualmente contraten todos los servicios por evitarse decidir, puesto que con la aplicación se podrían detectar los servicios redundantes o que ofrecen muy poco valor.

Las empresas que ofrecen servicios podrían observar un descenso en la piratería, ya que, al reducir la complejidad de la toma de decisiones en el proceso de contratación, sería más sencillo para el usuario contratar el servicio que quiere que tratar de piratear el contenido.

Estructura de la memoria

La memoria quedará estructura de la siguiente forma, coincidiendo los apartados principales con la metodología de desarrollo en cascada que se utilizará.

- **Introducción**: Capitulo introductorio en el que se definen la motivación, los objetivos, la metodología y el impacto esperado, así como la estructura general del documento.
- **Estado del arte**: En este capítulo se investigan aplicaciones del mercado actual que puedan tener similitud con la que se propone en este proyecto, así como una revisión a trabajos similares presentados con anterioridad en la Universitat Politècnica de València.
- **Análisis del problema**: Esta sección contiene los distintos análisis requeridos para definir la aplicación. Se incluyen el análisis de mercado,

requisitos funcionales y no funcionales, análisis legal. Se analizan diversas soluciones posibles y se ofrece una propuesta de solución y un presupuesto estimado.

- **Diseño de la solución**: En este apartado pueden encontrarse los diversos diseños de la aplicación: arquitectura, infraestructura y modelo de datos, maquetas de pantallas. Se incluye un apartado con todas las tecnologías y herramientas utilizadas.
- **Desarrollo de la solución propuesta:** En este apartado se habla del proceso de desarrollo, codificación e implementación de la aplicación.
- **Resultado final:** Este capítulo incluye los resultados obtenidos con capturas de la aplicación final donde puede verse su aspecto y funciones. Se incluyen un apartado con las pruebas realizadas.
- **Conclusiones**: Se incluyen las conclusiones obtenidas tras la realización del proyecto.
- **Trabajos futuros**: Contiene las evoluciones del proyecto que no han podido completarse en la versión actual.
- **Referencias**: Contiene las referencias bibliográficas utilizadas para la elaboración de este proyecto.
- **Glosario**: Contiene las definiciones de una serie de términos y acrónimos para facilitar la lectura a aquellas personas que no sean especialistas en la materia.
- **Anexos**: Contiene los anexos de lectura opcional.

2. Estado del arte

Actualmente no existen en el mercado aplicaciones que realicen la función principal del sistema que se propone en este TFG. Sin embargo, existen algunas aplicaciones que realizan algunas funcionalidades secundarias de esta propuesta y además pueden relacionarse de manera clara con la temática.

En primer lugar, se analizará JustWatch una aplicación disponible tanto para dispositivos móviles como en una versión web. A nivel usuario, esta aplicación podría considerarse como un agregador de recomendaciones, donde los usuarios pueden registrarse y comenzar a recibir recomendaciones sobre series o películas que podrían interesarles en base a contenido que les gusta y ya han consumido. El beneficio de JustWatch es ofrecer recomendaciones de todos los servicios a la vez, en lugar de obtenerlas separadas servicio a servicio.

A nivel comercial, JustWatch (*Véase* Fig. 3) contiene el catálogo actualizado de los principales servicios de *streaming* de más de 40 países. Ofrece un servicio de *API* a empresas con las que puedan colaborar, no siendo esta publica ni pudiendo adquirirse mediante pago o suscripción. Dicha *API* ofrece acceso a los catálogos de los servicios de *streaming* que indexa JustWatch. En el caso de España y en fecha de redacción de este documento, JustWatch dispone del catálogo de 29 servicios de *streaming* que usan el modelo de pago de suscripción, además de los catálogos de 9 servicios que ofrecen contenido gratuito y 7 servicios que ofrecen compra o alquiler individual de contenido. Si un servicio ofrece varias de las modalidades, JustWatch lo cuantifica tantas veces como modalidades ofrezca. Dado la riqueza del catálogo y siendo la indexación de este uno de los puntos clave para el desarrollo del sistema propuesto en este TFG, se contactó con el equipo de *API* de JustWatch para obtener acceso y poder utilizarlo, pero no fue una relación en la que estuvieran interesados. Aun con todo esto, JustWatch no cuenta con la característica principal de lo que se propone en este TFG, que sería el obtener el servicio más barato en función de nuestra preferencia.

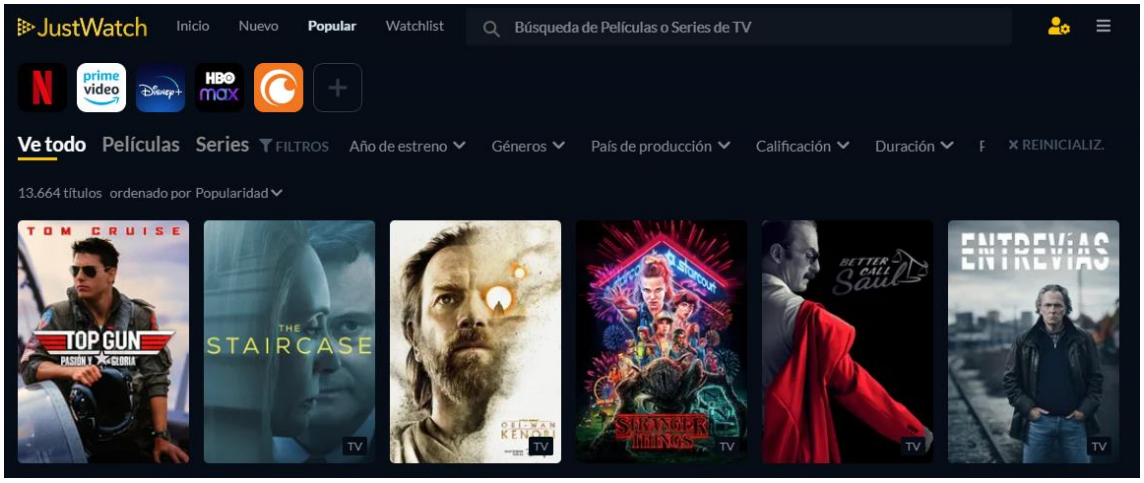

Fig. 3 – Imagen de portada de JustWatch

A continuación, se procederá a analizar dos aplicaciones cuya función es la de facilitar crear grupos en los servicios de *streaming* con el fin de aprovechar los planes familiares, que habitualmente dividiendo el precio salen más rentables que la suscripción individual, ya que esta es una de las funciones secundarias que se plantean para StreamBudget.

Las aplicaciones en cuestión son Spliiit y Together Price, teniendo ambas una versión web y en el caso de Together Price también está disponible en dispositivos móviles. El funcionamiento de ambas es prácticamente idéntico por lo que se explicarán en conjunto. En estas aplicaciones, una vez registrado, el usuario puede elegir ser el administrador de una suscripción y buscar miembros con los que dividirla (*Véase* Fig. 4) o puede optar por buscar grupos existentes y unirse. Se utiliza una cartera digital para hacer llegar los pagos al administrador y tienen sistemas antifraude que "congelan" el dinero durante un tiempo para evitar estafas, así como puntos de reputación. Agregar esta función a StreamBudget como parte de sus funciones secundarias tendría sentido, ya que lo que se busca es optimizar el presupuesto para los servicios de *streaming* y las cuentas compartidas son una forma de hacerlo que actualmente es legítima.

Fig. 4 – Imagen del rol "Admin" en Together Price

Crítica al estado del arte

Se ha realizado un trabajo de investigación sobre los trabajos existentes y no se ha encontrado ninguna aplicación o trabajo similar al sistema que se propone este TFG.

En primer lugar, existen algunos trabajos que tratan sobre Netflix y HBO, que son servicios de *streaming*, pero estos se enfocan en aspectos empresariales, siendo el caso de ***Análisis comparativo entre Netflix y HBO: cómo usan sus distintas estrategias comerciales*** [3] que está orientado sobre todo a la parte comercial y ***Análisis sobre el uso de la nostalgia en plataformas streaming (Netflix)*** [4], estando este último orientado al marketing.

Por otro lado, si bien existen trabajos donde se han desarrollado aplicaciones móviles, ya para Android u otras plataformas, ninguna está centrada en la comparación de precios entre las plataformas de *streaming*. Buscando la mayor similitud, se han revisado los siguientes trabajos:

Gestor de películas

"*Una aplicación sencilla, pero a la vez útil basada en la capacidad de consultar en todo momento las películas mejor valoradas, recientemente estrenadas, las más populares (…)*" [5].

Este trabajo ofrece una aplicación Android con la cual se pueden ver listados de películas obtenidas mediante la *API* de The Movie Database (en adelante TMDB). Los filtros de ordenación son *Popular*, *En cartelera hoy*, *Próximamente* y *Mejor valoradas*. Además, permite añadir películas a listados propios, siendo estos *Películas favoritas* y *Películas para ver.* Todas estas funciones podrían ser consideras para StreamBudget como funciones secundarias. Lo más interesante es el uso de la *API* TMDB, ya que uno de los principales obstáculos para StreamBudget es el poblado del catálogo de contenido, siendo esta *API* además alimentada por la *API* de JustWatch mencionada anteriormente en este trabajo. Si bien cuando se creó Gestor de películas la *API* de TMDB no lo tenía disponible, desde enero de 2021 es posible filtrar la respuesta de la *API* por proveedor [6], lo cual hace de esta *API* una candidata para StreamBudget.

Base de datos cinematográfica con sistema de recomendaciones por perfiles de usuarios

"*Proyecto basado en la idea de convertir los datos y opiniones sobre toda la colección de obras hechas tanto para cine como la televisión en una interacción entre diversas personas*" [7]

El objetivo de este trabajo es ofrecer una alternativa a webs como IMDb o Rotten Tomatoes, que funcionan como webs de noticias y agregadores de opiniones, pero centrándose en un aspecto de red social y en los servicios de *streaming*, llegando a mencionarse en su propuesta el filtrar las recomendaciones por servicio de *streaming* esto no llegó a la versión final. El poder factorizar las opiniones sobre contenidos multimedia de otros usuarios a modo de red social en un servicio de *streaming* concreto es una idea interesante, que podría llevarse como una opción más del algoritmo de StreamBudget, ya que es una forma de darle valor al contenido.

CineMapp: Una red social de cine para dispositivos móviles

"(…) red social de cine donde los usuarios pueden comentar y valorar las películas" [8]

Esta aplicación en su versión presentada ofrece un buscador de cines cercanos, así como un catálogo de películas que permite valorar cada una por los usuarios al estilo red social. Al igual que *Gestor de películas* utiliza la *API* de TMBD para poblar su catálogo de películas y similar a *Base de datos cinematográfica con sistema de recomendaciones por perfiles de usuarios* tiene como funcionalidad el poder valorar las películas. Esta aplicación está enfocado al cine de cartelera, quedando el mercado del *streaming* sin cubrir.

Propuesta

Tras el trabajo de investigación sobre el estado del arte, se llega a una conclusión similar a la que se llegó en trabajos anteriores: en el mercado actual existen muchas aplicaciones que ofrecen catálogos de películas o series y que dan la posibilidad de entrar en una comunidad donde compartir valoraciones. Sin embargo, y este es el

matiz para destacar en este trabajo, la gran mayoría de estos servicios basa su funcionalidad en que ya se ha visualizado el contenido. Es cierto que algunos servicios ofrecen la posibilidad de marcar una serie o película para indicar que se tienen intención de verla, pero no es una función principal, en el mejor caso se usa para poder medir la anticipación sobre ese contenido. Reduciéndolo a una sola frase:

La gran mayoría de servicios enfoca su funcionalidad de manera reactiva, buscando las valoraciones una vez ya se ha consumido el contenido.

La propuesta de StreamBudget usa el enfoque contrario, buscando ser proactiva. La funcionalidad principal requiere que el usuario indique el contenido que quiere ver para después poder ofrecerle el servicio o servicios más rentables en base a unos parámetros de valor percibido. Esta función no se ha encontrado actualmente en ninguna aplicación del mercado.

Así pues, lo que se busca es la creación de una aplicación cuya función principal sea la anteriormente mencionada y, además, agregar todas las funcionalidades más frecuentes en aplicaciones similares, como poder valorar películas, calendarios de cartelera e incluso la capacidad de poder compartir la suscripción buscando más usuarios para compartir los costes. También debería contar con un sistema de cuentas de usuario para persistir la información que puede evolucionar a un sistema de red social, en el que sea posible agregar amigos, saber qué contenido les interesa e incluso recomendarles contenidos.

Mediante el uso de una tabla MoSCoW (*Véase* Tabla 1), es posible ordenar y dar prioridad a las funciones que se requieren.

Tabla 1 – Tabla MoSCoW de requisitos

Must have (Debe tener)	Catálogos de contenido (series y películas) de los servicios de *streaming* Netflix, Prime Video y HBO Max.
	Lista de contenido que se quiere visionar.
	Buscador de contenido.
	Modificar la lista de contenido a visionar.
	Algoritmo de cálculo que dé un peso igual a una serie y a una película.
	Algoritmo de cálculo que dé peso en función del número de episodios.
	Calcular y ofrecer el servicio con mayor valor para el usuario en función de su lista de contenido a visionar y su algoritmo seleccionado.
Should have (Debería tener)	Algoritmo de cálculo que dé peso en función de los minutos.
	Catálogo de contenido de Disney+.
	Filtros de contenido (género, año, …).
	Ordenación de contenido (titulo, año, …).
	Calendario de contenidos, indicando contenidos que se van a eliminar o agregar a cada servicio.
	Interfaz intuitiva y con buena usabilidad.
	Cuentas de usuario.
Could have (Podría tener)	Listas de amigos.
	Recomendar contenido a un amigo.

	Acceso al perfil de un amigo, ver su lista de contenido que quiere visionar.
	Algoritmo complementario que agregue peso a un servicio en función de las listas de mis amigos.
	Notificaciones de nuevas recomendaciones de servicios al cambiar el contenido de los catálogos.
	Valoraciones de contenido.
	Sistema de comentarios.
	Sistema para poder compartir el coste de la suscripción a un servicio.
Will not have (No tendrá)	Tráileres de las películas o series.
	Información sobre directores, actores, etc.…
	Buscador de descuentos para los servicios.

3. Análisis del problema

Análisis de mercado

Se realizó una pequeña encuesta utilizando la herramienta Google Forms con el fin de poder obtener datos de mercado sobre el interés en la propia aplicación y algunas de sus funciones (*Véase* Anexo II para más información). Con la muestra obtenida de 52 resultados, y aplicando un índice de confianza del 95% se conoce que el margen de error es del 13,59%. Respecto a la demografía de la muestra, se considera que están ajustados a la realidad y por tanto no hay un sesgo claro en los resultados (*Véase* Anexo II para más información).

El primer punto importante para revisar es el posible interés en la aplicación. En una escala del 1 al 5, el 43,31% eligió un interés de 4 y más del 80% de los encuestados marcó un interés de 3 o superior, como se puede ver en Fig. 5:

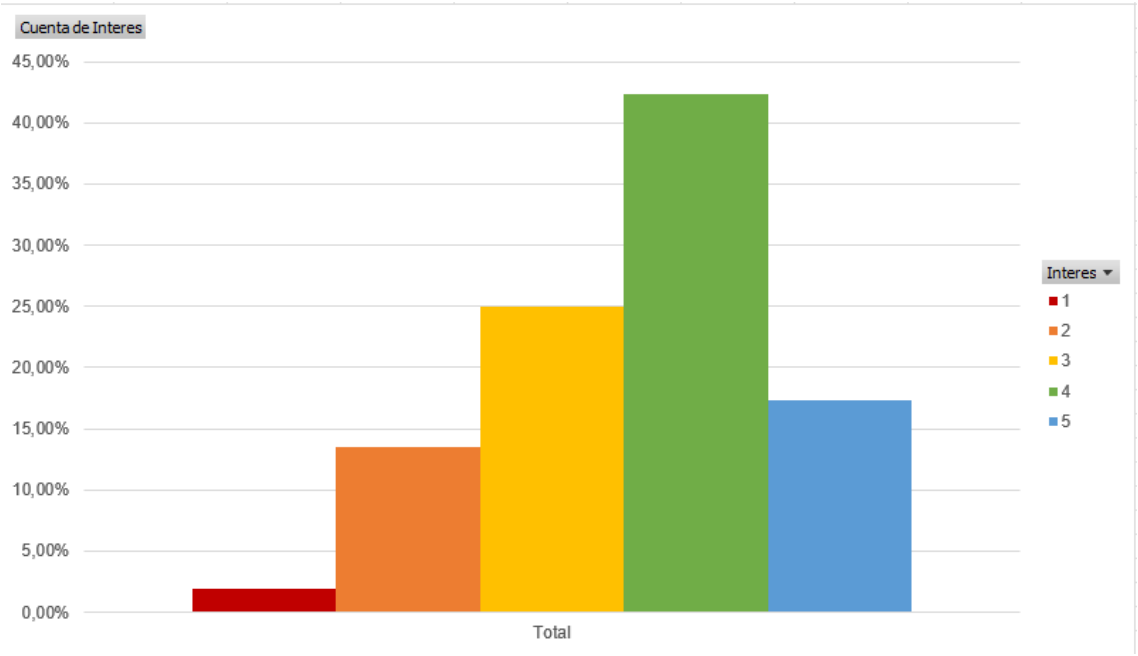

Fig. 5 – Gráfico de porcentaje del interés en la aplicación

Se puede pensar que el interés viene sobre todo de personas con ingresos más bajos, pero si agregamos el nivel de renta es posible observar que el interés se mantiene en todos los niveles. En el siguiente gráfico representado en Fig. 6, el porcentaje de cada barra representa el número de personas dentro de cada grupo de renta que eligió una opción, es decir, cada grupo de renta suma su propio 100%. Curiosamente, se puede observar que, en cada grupo de renta, el mayor porcentaje de gente tiene un interés de 4, salvo en el caso de la renta más baja, donde el interés mayoritario cae a 3 aunque también es el grupo que más porcentaje de interés máximo muestra. Definitivamente, se puede concluir que el interés se mantiene alto, independientemente del nivel de renta del encuestado.

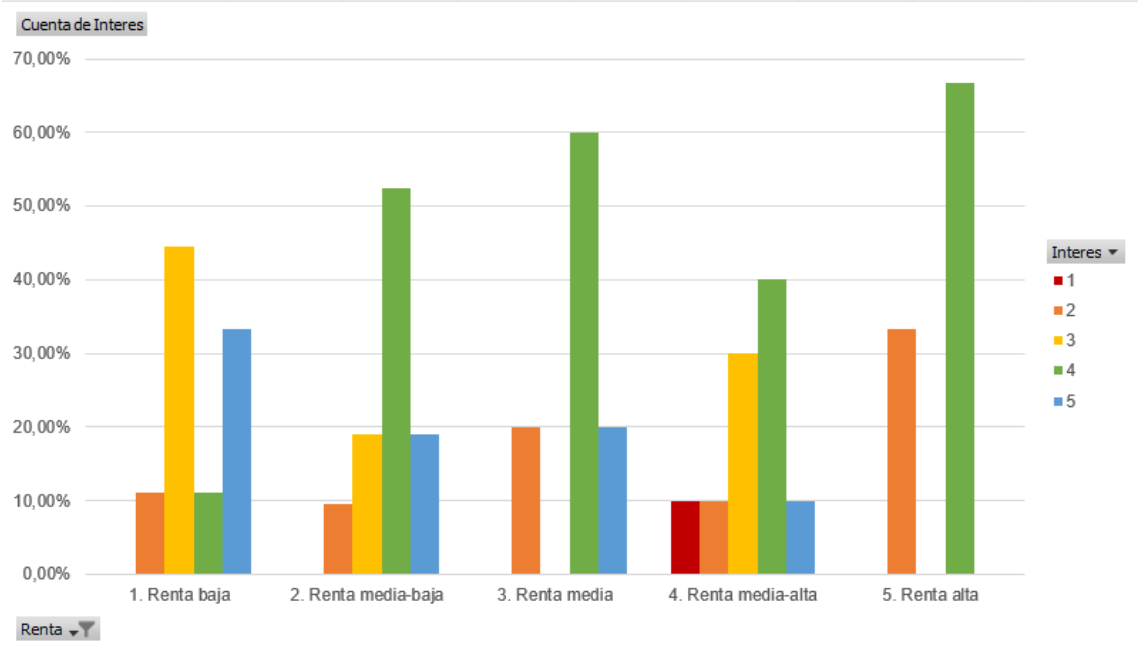

Fig. 6 – Gráfico de porcentaje por nivel de renta del interés en la aplicación

También ha sido posible conocer el grado de suscripción a diversos servicios de *streaming* (*Véase* Fig. 7), con el fin de priorizar o descartar los menos populares. En la siguiente gráfica de segmentos podemos observar que los servicios *Apple TV*, *Filmin* y *Rakuten TV* apenas tienen suscriptores según la muestra obtenida, por lo que no son prioridad a la hora de incorporar sus catálogos en la aplicación. Por el contrario, Netflix y Prime Video deben ser incorporados a la versión inicial.

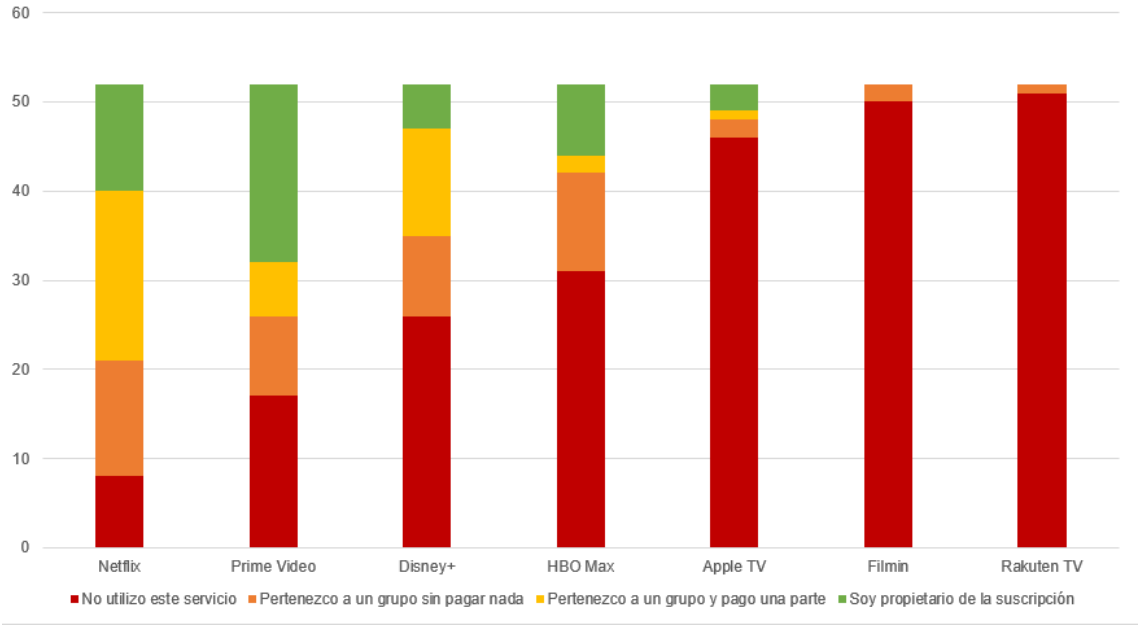

Fig. 7 – Gráfico de segmentos de suscripciones por servicio

Mediante la siguiente gráfica mostrada en Fig. 8, ha sido posible determinar cuáles son los algoritmos que más se valorarían a la hora de utilizar la aplicación. Estos datos se aprovecharon para priorizar estas funciones en Tabla 1.

Fig. 8 – Grafico de valor otorgado al servicio

Finalmente, se proponían a los encuestados cinco funcionalidades secundarias y se les pedía que les asignaran un valor del 1 al 5 en función del interés que tuvieran en ellas, pudiendo verse en Fig. 9. Similar al caso anterior, los datos obtenidos sirvieron para ayudar en la priorización de Fig. 3. No es una sorpresa ver que las funciones de red social despiertan el menor interés, ya que como se vio en el capítulo *2* esta función ya existe en muchas otras aplicaciones. La función “Ofertas especiales para los servicios de *streaming*” aunque queda en tercer lugar debe descartarse debido a su inviabilidad, ya que no es posible conseguir ofertas especiales en este punto del ciclo de vida de StreamBudget. Se tratará de priorizar como parte de las funciones secundarias el calendario de cambios de contenido y la recepción de actualizaciones sobre el mejor servicio, esto sería en el caso de que cambiasen los catálogos y otro servicio pasara a tener mayor peso que el actual.

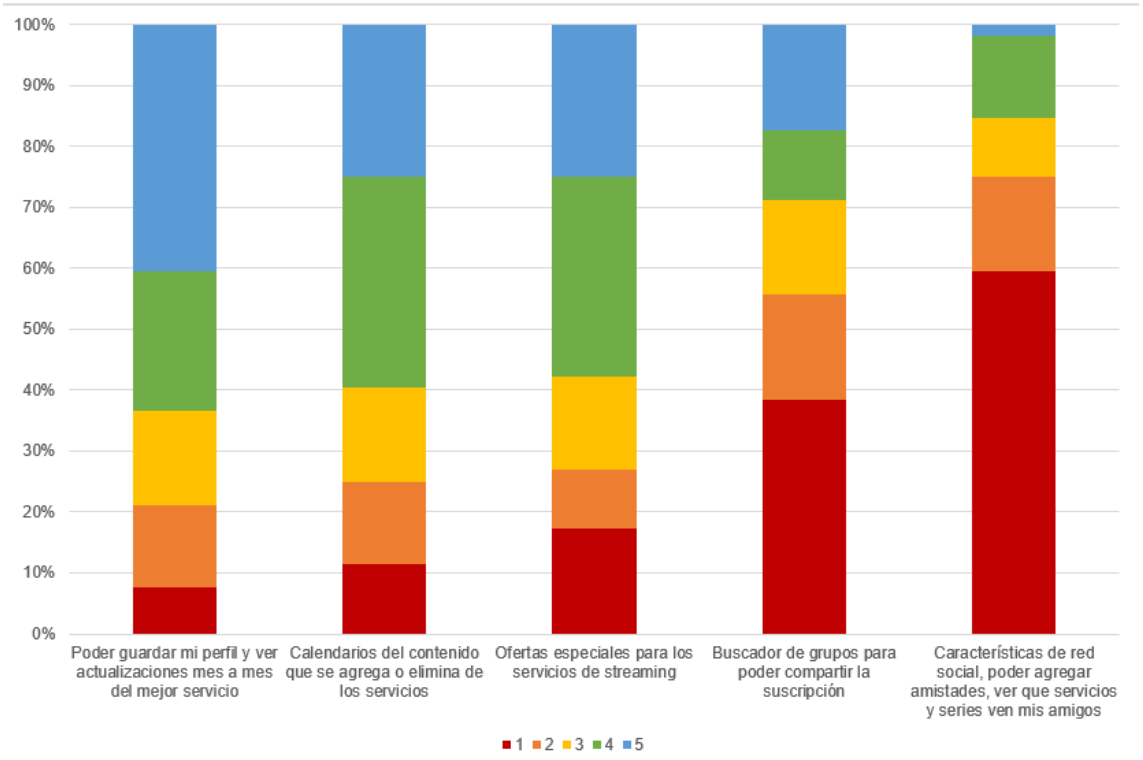

Fig. 9 – Grafico de interés en las funciones secundarias

Análisis de requisitos no funcionales

Los siguientes requisitos no están relacionados directamente con una funcionalidad concreta, sino que aplican a la aplicación al completo y son complementarios a los requisitos funcionales. Cada requisito tendrá asignado un identificador para poder ser referenciado con facilidad posteriormente. Se han identificado los siguientes requisitos, ordenándose por categorías.

Experiencia de usuario y usabilidad

- **UX-01**: La aplicación debe ser intuitiva y sencilla de utilizar. Debería poder navegarse desde-hasta cualquier menú en 4 acciones o menos.

- **UX-02**: La aplicación debe tener una curva rápida de aprendizaje del uso de sus funciones.

- **UX-03:** La aplicación debe impedir interacciones que lleven a errores del usuario. Deben deshabilitarse los campos cuando no deban ser rellenados tras seleccionar opciones concretas o cuando sean incompatibles con otros datos introducidos.

Rendimiento

- **RND-01:** Tiempo máximo de respuesta a las peticiones 2 segundos. Debe trabajarse para obtener el menor tiempo posible.

- **RND-02:** La aplicación debe ocupar el menor espacio posible en el dispositivo y nunca debe superar 1GB. Debe evitarse guardar grandes cantidades de datos en cache.

Seguridad

- **SEC-01:** Las conexiones entre la aplicación y el *backend* deben ser seguras, realizadas usando el protocolo HTTPS.

- **SEC-02:** Un usuario puede acceder únicamente a sus propios datos privados, nunca debe ser posible acceder a datos privados de otros usuarios. Los datos compartidos deliberadamente mediante funciones de red social no se consideran datos privados.

- **SEC-03:** Las contraseñas deben ser almacenadas usando el algoritmo criptográfico MD5. Nunca deben guardarse como texto plano.

- **SEC-04:** Los datos personales de los usuarios deben salvaguardarse siguiendo las normativas legales vigentes y no deben ser filtrados o cedidos, salvo que se soliciten de manera legal por parte de las fuerzas del orden.

Confiabilidad

- **RLB-01**: La aplicación ha de estar disponible de manera continua, deben evitarse caídas en el servicio.

- **RLB-02**: La aplicación debe tener la capacidad de recuperarse ante un fallo o un *timeout* al contactar con el *backend*.

- **RLB-03**: Debe asegurarse que los datos mostrados son consistentes. No deben perderse datos de los usuarios.

Compatibilidad y portabilidad

- **PTB-01**: La aplicación ha de ser compatible de Android 8.0 en adelante (nivel de *API* 26).

Soporte

- **SUP-01**: La aplicación debe mantenerse y corregir errores mediante correctivos.

- **SUP-02**: La aplicación debe evolucionarse, agregando nuevas funciones o mejorando las actuales.

- **SUP-03:** Se realizará un seguimiento de los fallos de la aplicación mediante el uso de la herramienta Firebase Crashlytics, que permite recibir errores producidos en cualquier dispositivo, incluyendo la información necesaria para su depuración. Además de esto, se revisará de manera periódica los comentarios negativos que se realicen en la Google Play Store.

Análisis de requisitos funcionales

En este punto se detallarán los requisitos funcionales, mediante el uso de tablas (*Véanse* Tabla 2 a Tabla 13) y diagramas de casos de uso (*Véanse* Fig. 10 a Fig. 11). Estos requisitos están relacionados directamente con las funcionalidades de la aplicación y permiten describir los comportamientos, las condiciones previas o posteriores al comportamiento y las entradas y salidas esperadas de cada caso. Para la versión inicial de la aplicación se consideran dentro del alcance solamente los requisitos de las funcionalidades Debe tener y Debería tener de la tabla de MoSCoW (*Véase* Tabla 1*)*. Se diferencian tres actores principales, el usuario no registrado, el usuario invitado y el usuario registrado. El usuario invitado puede utilizar un subgrupo de las funciones del usuario registrado, considerándose en el diagrama que el usuario registrado hereda todos los casos de uso del usuario invitado.

Usuario no registrado

Tabla 2 – CU-01, registrar usuario

Identificador y nombre	**CU-01** – Registrar usuario
Objetivo	Crea una nueva cuenta de usuario usando su correo electrónico.
Descripción	El usuario podrá registrar una cuenta en la aplicación mediante un correo electrónico y una contraseña.
Precondición	El usuario no debe estar registrado previamente. El usuario debe estar conectado a internet.
Secuencia	1. Iniciar la aplicación. 2. Presionar en el botón registrar cuenta. 3. La aplicación muestra el formulario de registro. 4. El usuario completa los campos y pulsa el botón registrarse. 5. La aplicación valida los datos y da de alta al usuario si son correctos. 6. La sesión del usuario queda iniciada y se le redirige al menú principal.
Postcondición	El usuario queda registrado en la aplicación.
Excepciones	Los datos no son correctos, la aplicación avisa para su revisión.
Notas adicionales	

Tabla 3 – CU-02, acceso como invitado

Identificador y nombre	**CU-02** – Acceso como invitado
Objetivo	Continuar a la aplicación sin necesidad de crear una cuenta en la aplicación.
Descripción	El usuario podrá acceder a la aplicación sin necesidad de crear una cuenta, pudiendo acceder a un conjunto limitado de funcionalidades.
Precondición	El usuario debe estar conectado a internet.
Secuencia	1. Iniciar la aplicación. 2. Presionar en el botón acceder como invitado. 3. La sesión como usuario invitado queda iniciada y se le redirige al menú principal.
Postcondición	Ninguna.
Excepciones	Ninguna.

Notas adicionales	El acceso como invitado permitirá al usuario seleccionar su lista de contenido y poder obtener el servicio recomendado, pero no tendrá persistencia de estos datos en diferentes dispositivos ni podrá usar las funciones sociales.

Usuario invitado

Tabla 4 – CU-03, agregar contenido a la lista "quiero ver"

Identificador y nombre	**CU-03** – Agregar contenido a la lista “quiero ver”
Objetivo	Agregar películas o series a la lista de contenido a visualizar.
Descripción	El usuario podrá ir agregando los distintos contenidos (películas o series) a su lista “quiero ver”.
Precondición	El usuario debe tener una sesión iniciada como usuario registrado o como invitado. El usuario debe estar conectado a internet. El contenido no debe estar ya en la lista “quiero ver”.
Secuencia	1. Iniciar la aplicación 2. Acceder como invitado o con cuenta de usuario. 3. Acceder al menú del catálogo de contenido. 4. La aplicación cargará los catálogos de películas y series de los servicios de *streaming*. 5. Presionar el botón con icono de ojo gris ().
Postcondición	La aplicación cambia el icono al color negro para indicar que se ha añadido el contenido a la lista. El contenido queda agregado a la lista “quiero ver”.
Excepciones	En caso de fallo al recuperar el contenido en el paso 4, la aplicación mostrará un aviso.
Notas adicionales	

Tabla 5 – CU-04, eliminar contenido de la lista "quiero ver"

Identificador y nombre	**CU-04** – Eliminar contenido de la lista “quiero ver”
Objetivo	Eliminar películas o series a la lista de contenido a visualizar.
Descripción	El usuario podrá eliminar los distintos contenidos (películas o series) de su lista “quiero ver”.
Precondición	El usuario debe tener una sesión iniciada como usuario registrado o como invitado. El usuario debe estar conectado a internet. El contenido debe estar agregado a la lista “quiero ver”.
Secuencia	1. Iniciar la aplicación 2. Acceder como invitado o con cuenta de usuario. 3. Acceder al menú del catálogo de contenido. 4. La aplicación cargará los catálogos de películas y series de los servicios de *streaming*. 5. Presionar el botón con icono de ojo negro ().
Postcondición	La aplicación cambia el icono al color gris para indicar que se ha eliminado el contenido de la lista. El contenido queda eliminado de la lista “quiero ver”.
Excepciones	En caso de fallo al recuperar el contenido en el paso 4, la aplicación mostrará un aviso.
Notas adicionales	

Tabla 6 – CU-05, buscar contenido

Identificador y nombre	**CU-05** – Buscar contenido

Objetivo	Buscar películas o series por su título.
Descripción	El usuario podrá buscar películas o series por su título para facilitar agregarlas a su lista.
Precondición	El usuario debe tener una sesión iniciada como usuario registrado o como invitado. El usuario debe estar conectado a internet. El usuario debe estar en el menú catálogo de contenido.
Secuencia	1. Iniciar la aplicación 2. Acceder como invitado o con cuenta de usuario. 3. Acceder al menú del catálogo de contenido. 4. La aplicación cargará los catálogos de películas y series de los servicios de *streaming*. 5. Presionar sobre la barra de búsqueda. 6. Introducir el título de la película o serie.
Postcondición	Se muestran los contenidos encontrados que coincidan con el texto introducido. Si no hay coincidencias la pantalla de resultados aparecerá vacía con el texto “No se encontraron resultados”.
Excepciones	En caso de fallo al recuperar el contenido en el paso 4, la aplicación mostrará un aviso.
Notas adicionales	

Tabla 7 – CU-06, filtrar contenido

Identificador y nombre	**CU-06** – Filtrar contenido
Objetivo	Filtrar el listado películas o series según los parámetros indicados.
Descripción	El usuario podrá filtrar películas o series por su año de estreno y/o género cinematográfico. También podrá filtrar el contenido “marcado para ver”
Precondición	El usuario debe tener una sesión iniciada como usuario registrado o como invitado. El usuario debe estar conectado a internet. El usuario debe estar en el menú catálogo de contenido.
Secuencia	1. Iniciar la aplicación 2. Acceder como invitado o con cuenta de usuario. 3. Acceder al menú del catálogo de contenido. 4. La aplicación cargará los catálogos de películas y series de los servicios de *streaming*. 5. Presionar sobre el icono de “ordenación y filtros” (⏷). 6. El usuario seleccionará una o más opciones de la sección “filtros”. 6.1. Marcadas para ver. 6.2. Año de estreno. 6.3. Género. 7. Presionar el botón filtrar.
Postcondición	Se muestran los contenidos filtrados en base a las opciones seccionadas. Si la combinación de opciones no produce resultados, la pantalla aparecerá vacía con el texto “Los filtros seleccionados no producen ningún resultado”.
Excepciones	En caso de fallo al recuperar el contenido en el paso 4, la aplicación mostrará un aviso.
Notas adicionales	Las opciones podrán usarse en conjunto con CU-07

Tabla 8 – CU-07, ordenar contenido

Identificador y nombre	**CU-07** – Ordenar contenido
Objetivo	Ordenar el listado películas o series según los parámetros indicados.
Descripción	El usuario podrá ordenar películas o series por su año de estreno o su título.
Precondición	El usuario debe tener una sesión iniciada como usuario registrado o como invitado. El usuario debe estar conectado a internet. El usuario debe estar en el menú catálogo de contenido.
Secuencia	1. Iniciar la aplicación 2. Acceder como invitado o con cuenta de usuario. 3. Acceder al menú del catálogo de contenido. 4. La aplicación cargará los catálogos de películas y series de los servicios de *streaming*. 5. Presionar sobre el icono “ordenación y filtros” ( ). 6. El usuario seleccionará una única opción del apartado “ordenar por”. 6.1. Título. 6.2. Año de estreno. 7. El usuario elegirá si ordenar de manera ascendente o descendente. 8. Presionar el botón filtrar.
Postcondición	Se muestran los contenidos ordenados en base a las opciones seccionadas.
Excepciones	En caso de fallo al recuperar el contenido en el paso 4, la aplicación mostrará un aviso.
Notas adicionales	Las opciones podrán usarse en conjunto con CU-06

Tabla 9 – CU-08, acceder a calendario de contenidos

Identificador y nombre	**CU-08** – Acceder a calendario de contenidos
Objetivo	Ver los contenidos que se agreguen o eliminen de los servicios en el mes actual.
Descripción	El usuario podrá acceder al listado de películas y series que se han agregado o eliminado de cada servicio de *streaming* desde el día 1 del mes actual.
Precondición	El usuario debe tener una sesión iniciada como usuario registrado o como invitado. El usuario debe estar conectado a internet.
Secuencia	1. Iniciar la aplicación 2. Acceder como invitado o con cuenta de usuario. 3. Acceder al menú del calendario de contenidos.
Postcondición	La aplicación mostrará el diferencial de contenidos desde el día 1 del mismo mes.
Excepciones	En caso de fallo al recuperar el listado en el paso 3, la aplicación mostrará un aviso.
Notas adicionales	

Tabla 10 – CU-09, calcular recomendaciones de servicios a contratar

Identificador y nombre	**CU-09** – Calcular recomendaciones de servicios a contratar
Objetivo	Calcular qué servicios serían recomendables para el usuario en función de su lista “quiero ver”.

Descripción	El usuario podrá ajustar mediante opciones cómo dar valor a los servicios (lo cual utilizará un algoritmo u otro) y poder recibir las recomendaciones en función a esos parámetros y su lista “quiero ver”.
Precondición	El usuario debe tener una sesión iniciada como usuario registrado o como invitado. El usuario debe estar conectado a internet. El usuario debe haber añadido al menos un elemento a su lista “quiero ver”.
Secuencia	1. Iniciar la aplicación 2. Acceder al menú calculadora de recomendaciones. 3. La aplicación muestra una pantalla con un desplegable que representa los distintos algoritmos. 4. El usuario selecciona el tipo de algoritmo a utilizar. 4.1. Número de elementos. 4.2. Número de episodios. 4.3. Número de minutos disponibles. 5. El usuario presiona el botón calcular.
Postcondición	La aplicación muestra el listado de servicios recomendados, ordenados de mayor a menor puntuación obtenida en el algoritmo.
Excepciones	
Notas adicionales	El algoritmo número de elementos considera el mismo peso para una película que una serie. El algoritmo número de episodios considera una película como peso 1 y mientras que el peso de una serie sería igual a su número de episodios.

Usuario registrado

Tabla 11 – CU-10, iniciar sesión

Identificador y nombre	**CU-10** – Iniciar Sesión
Objetivo	Iniciar sesión en la cuenta de usuario.
Descripción	El usuario podrá iniciar sesión en su cuenta, para poder ver sus datos en diferentes dispositivos.
Precondición	El usuario debe estar registrado. El usuario no debe tener sesión iniciada. El usuario debe estar conectado a internet.
Secuencia	1. Iniciar la aplicación 2. Introducir el correo y contraseña en la pantalla de inicio de sesión. 3. Pulsar el botón acceder.
Postcondición	La sesión del usuario queda iniciada.
Excepciones	Si los datos introducidos no coinciden con usuario registrado, la aplicación mostrará un aviso.
Notas adicionales	

Tabla 12 – CU-11, modificar datos de la cuenta

Identificador y nombre	**CU-11** – Modificar datos de la cuenta
Objetivo	Modificar los datos de la cuenta del usuario.
Descripción	El usuario podrá modificar los datos de su cuenta, como su nombre o su contraseña.
Precondición	El usuario debe tener una sesión iniciada como usuario registrado. El usuario debe estar conectado a internet.

Secuencia	1. Iniciar la aplicación 2. Acceder al menú cuenta de usuario. 3. La aplicación carga un formulario con los datos actuales del usuario. 4. Pulsar el icono editar (✎) para poder modificar un campo. El usuario puede cambiar los datos que desee. 5. Introducir la contraseña en el campo contraseña actual. 6. Pulsar el botón actualizar.
Postcondición	Los datos de la cuenta quedan actualizados
Excepciones	En caso de fallo al recuperar los datos en el paso 3, la aplicación mostrará un aviso. En caso de fallo al enviar los datos en el paso 5, la aplicación mostrará un aviso y los datos no serán actualizados.
Notas adicionales	

Tabla 13 – CU-12, eliminar la cuenta

Identificador y nombre	**CU-12** – Eliminar la cuenta
Objetivo	Eliminar la cuenta de usuario y sus datos.
Descripción	El usuario puede eliminar su cuenta y todos los datos asociados a ella de la aplicación.
Precondición	El usuario debe tener una sesión iniciada como usuario registrado. El usuario debe estar conectado a internet.
Secuencia	1. Iniciar la aplicación 2. Acceder al menú cuenta de usuario. 3. La aplicación carga un formulario con los datos actuales del usuario. 4. Introducir la contraseña en el campo contraseña actual. 5. Pulsar el botón Eliminar cuenta de usuario. 6. La aplicación muestra un mensaje solicitando confirmación. 7. Pulsar el botón aceptar.
Postcondición	La cuenta de usuario queda eliminada y sus datos son purgados del servidor.
Excepciones	En caso de fallo al recuperar los datos en el paso 3, la aplicación mostrará un aviso. En caso de que el usuario no confirme en el paso 5, los datos no serán eliminados. En caso de fallo al eliminar los datos en el paso 6, la aplicación mostrará un aviso y los datos no serán eliminados.
Notas adicionales	

Diagramas de casos de uso

Fig. 10 – Diagrama de casos de uso del usuario no registrado

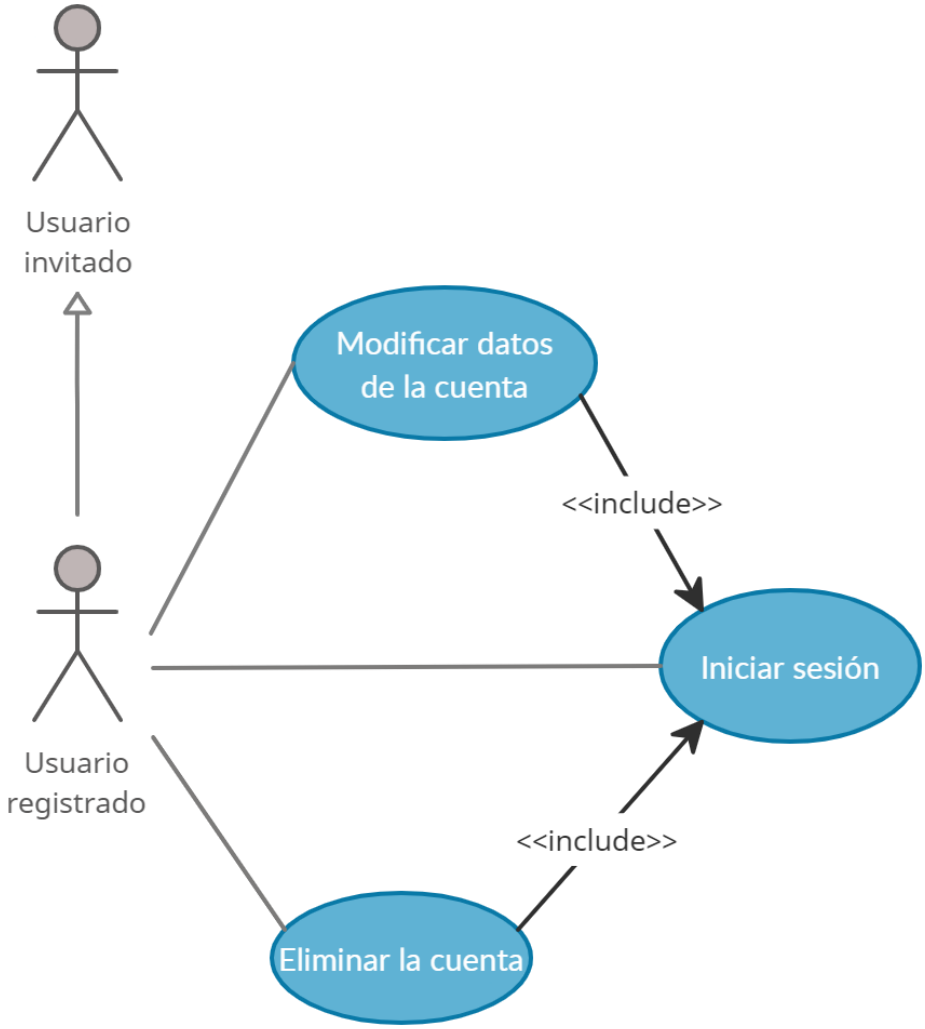

Fig. 11 – Diagrama de casos de uso del usuario registrado

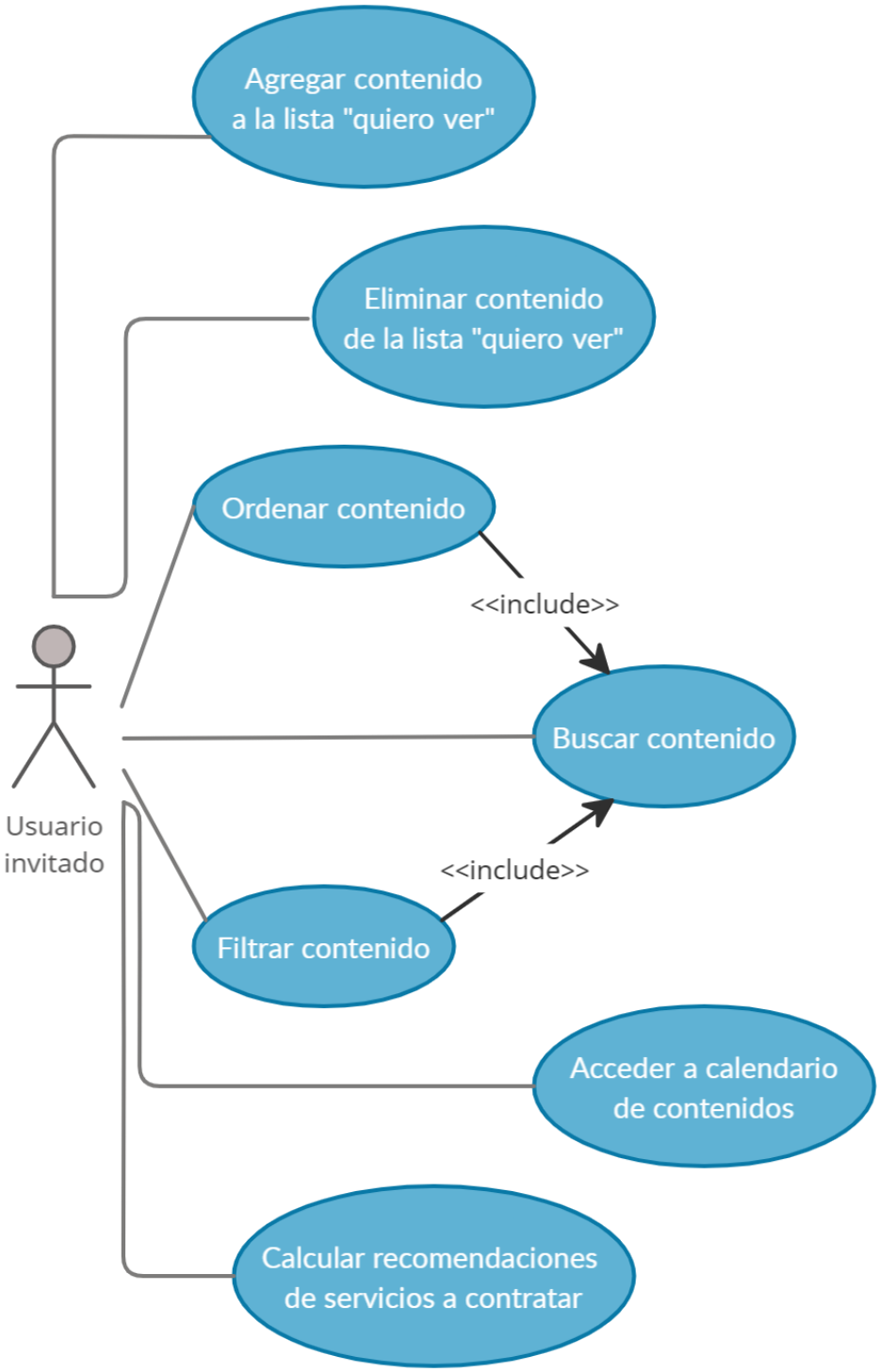

Fig. 12 – Diagrama de casos de uso del usuario invitado

Análisis del marco legal y ético

Dado que vamos a trabajar con datos del usuario y el foco de la aplicación será España, hay una serie de leyes, reglamentos y normativas que deben tenerse presentes.

En primer lugar, debe tenerse en cuenta el Reglamento General de Protección de Datos (En adelante *RGPD*), también conocido por sus siglas en ingles *GDPR*. En el caso de España se reformó la ley de protección de datos para adaptarse a lo indicado en la *RGPD*, por lo que principalmente nos basaremos en la ley española que se detallará más adelante. Dentro de este reglamento, los datos recogidos por StreamBudget al registrar un usuario (nombre, apellidos, fecha de nacimiento y correo electrónico) pertenecerían a las categorías de Datos identificativos y características personales, ninguna de las cuales es una de las categorías especiales de datos que la *RGPD* trata como de mayor sensibilidad, requiriendo mayor protección. En el caso de los usuarios no registrados o invitados, no se recopila ningún tipo de dato identificativo. Por otro lado, este reglamento obliga a que el usuario tenga que dar su consentimiento para el tratamiento de sus datos personales (se indicará en el registro de usuario) y a garantizar su derecho de consulta, modificación y eliminación de datos del sistema de StreamBudget.

La segunda es la **Ley Orgánica 3/2018 de Protección de Datos y Garantía de Derechos Digitales** (En adelante *LOPDGDD*) publicando en el Boletín Oficial del Estado (en adelante BOE) del día 6 de diciembre de 2018 [9]. Como su propio nombre indica, esta ley recoge la legislación relativa al tratamiento de datos personales, así como los derechos digitales de las personas físicas. Esta ley está compuesta de 97 artículos separados en 10 secciones (títulos). También cuenta con disposiciones (adicionales, transitorias, derogatorias y finales) que no son interés para este análisis. Debido al tipo de datos con el que se trabajará en StreamBudget, deben tenerse en cuenta los siguientes artículos de la *LOPDGDD,* que serán descritos de manera resumida a continuación, agrupados por el título en el que aparecen en el BOE.

Título II. Principios de protección de datos.

Artículo 4. Exactitud de los datos: Lo datos de los usuarios deben ser exactos. Debemos garantizar que se guarden correctamente en el sistema, pero dado que es el propio usuario el que los proporciona, no es necesario hacer verificaciones adicionales [9, p. 119801].

Artículo 5. Deber de confidencialidad: Los datos de los usuarios deben permanecer confidenciales. Deben ponerse las medidas necesarias tanto tecnológicas como humanas para evitar filtraciones de datos [9, p. 119802].

Artículo 6. Tratamiento basado en el consentimiento del afectado: Los usuarios deben consentir el tratamiento de sus datos y estos solo pueden tratarse únicamente con los fines descritos en el consentimiento que los usuarios acepten [9, p. 119802]. Lo que se referencia en este artículo es un reflejo de lo que indica la RGPD respecto a los consentimientos que se ha analizado anteriormente.

Artículo 7. Consentimiento de los menores de edad: Este articulo contiene medidas adicionales y más restrictivas complementarias al artículo 6, para los casos en los que los usuarios sean menores de 14 años. Los menores de 14 años no disponen de potestad para consentir el tratamiento de sus datos y requieren la autorización de un tutor legal. Dado que solicitaremos la fecha de nacimiento del usuario, la aplicación negará el registro a los usuarios menores de 14 años, no llegando a guardar sus datos en ningún momento [9, p. 119802].

Título III. Derechos de las personas.

Artículo 11. Transparencia e información al afectado: Este artículo contempla dos casuísticas, pero la relevante para StreamBudget es cuando se obtienen datos directamente por parte del usuario. El derecho de los usuarios es este caso es poder conocer información básica sobre quién o quiénes son los tratantes de sus datos y cuáles son las finalidades del tratamiento de dichos datos, debiendo poder obtener contacto electrónico y/u otros medios [9, p. 119803].

Artículos 13 – 18. Derecho de acceso, de rectificación, de supresión, a la limitación del tratamiento, a la portabilidad y de oposición. Este grupo de artículos dota al usuario de control prácticamente total sobre sus datos [9, pp. 119804 - 119805]. Estos artículos son un reflejo directo de los articulo 15 – 21 de la RGPD, haciéndose referencia directa a ellos dentro del BOE. Los derechos de acceso, rectificación y supresión ya se han tratado anteriormente en el párrafo que analiza la RGPD.

El derecho a limitación de tratamiento permite al usuario contactar al tratador de sus datos para impedir su modificación o eliminación. El derecho de portabilidad permite al usuario obtener sus datos de un tratante de datos para poder cedérselo a otro tratante distinto. Finalmente, el derecho de oposición permite que el usuario indique al tratante de sus datos que únicamente puede realizar su tratamiento en casos excepcionales (interés público o mercadotecnia directa). El proceso para ejercer estos derechos en StreamBudget se haría mediante contacto electrónico (v.g. correo electrónico) que se habilitará para que los usuarios puedan enviar todas sus solicitudes referentes a sus datos.

Finalmente, y si bien este apartado ya no tiene implicaciones legales, se debe mencionar que ninguno de los servicios se opone de manera activa al uso de sus catálogos por terceros ni tampoco se han encontrado declaraciones que se opongan explícitamente al *scraping* de su catálogo, simplemente no se proporciona a los desarrolladores una interfaz de programación de aplicaciones (en adelante *API*) con la que obtener estos catálogos. Concretamente en el caso de Netflix, sí que se ofrecía una *API* que cualquier desarrollador podía usar, pero fue retirada del servicio en el año 2014 salvo para algunos colaboradores importantes de Netflix. Netflix ya no mantiene el blog de desarrolladores donde publicó la entrada del cese de servicio, pero gracias al proyecto *Wayback Machine* todavía puede consultarse [10]. El resto de los servicios principales (HBO Max, Prime Video y Disney+) nunca contaron con *API* para desarrolladores.

Identificación y análisis de soluciones posibles

Dado que no existen APIs públicas para desarrolladoras, un problema crítico para esta aplicación era la obtención de los catálogos de cada servicio. Se valoró el uso de técnicas de *scraping* para obtenerlos, pero esto hubiera desviado demasiado el alcance, ya que se trata de una tarea muy costosa y poco reaprovechable entre distintos servicios de *streaming*. La otra opción era el uso de *API* de terceros que nos ofrecieran estos catálogos.

Se estuvieron investigando distintas *API* que trabajan ofreciendo los catálogos de *streaming*. Las *API* que se investigaron o se intentó obtener acceso fueron JustWatch, Flixed [11] y TMDB. Se utilizó la web RapidApi [12] para buscar otras *API*

que pudieran ser de interés, pero o bien no disponían de todos los servicios (lo cual obligaría a usar más de una *API*) o bien se alimentaban a su vez de otra *API*, lo cual hacía redundante usarlas en lugar de su *API* principal.

Para la arquitectura global del sistema, se valoraron dos opciones:

1. Realizar todo el sistema en la aplicación móvil.
2. Realizar un sistema compuesto por la aplicación móvil y un *backend* que provea de distintos servicios a esta.

El primer caso tiene la ventaja de únicamente tener que desarrollar un sistema, sin embargo, la *API* analizadas ofrecen en algunos casos demasiada información y en otro la ofrecen de manera que requiere mucha lógica procesarla para la finalidad de la aplicación.

El segundo caso es algo más complejo, ya que el sistema estaría compuesto por dos subsistemas, pero las ventajas resultan muy atractivas. Por un lado, en caso de usar una *API* de pago podrían guardarse sus resultados y actualizar únicamente una vez al día, lo cual reduciría el número de peticiones y el coste. Además, tanto si la *API* es de pago como gratuita, podríamos refactorizar los datos en nuestro propio modelo para que fuera más sencillo y eficiente utilizar los catálogos. Finalmente, tener un *backend* propio podría permitir descargar algunas partes de la lógica en este. Con el abanico de dispositivos que acepta Android, podría darse el caso que los cuellos de botella al procesar algunos de los algoritmos fuera el dispositivo en sí, ganando eficiencia si podemos delegar esa carga en un servidor externo.

Solución propuesta

Finalmente, tras todo el análisis se opta por la combinación de aplicación y *backend*, usando la *API* de TMBD (inicialmente en su versión gratuita, pudiendo promocionar a la versión de pago), dado que es la única de las *API* contactadas que permitieron un acceso inicial para realizar el desarrollo.

La aplicación deberá dar solución a todos sus requisitos y cumplir la legislación, tomándose las medidas indicadas en apartados anteriores.

Presupuesto

Para la versión inicial, se tratará de buscar el producto mínimo viable con un presupuesto ajustado (*Véase* Fig. 13) junto con su mantenimiento en línea durante un año. El presupuesto se hará solo sobre la parte de desarrollo de código e infraestructura y no se tendrá en cuentas los costes (horas de trabajo) de los requisitos funcionales ni de QA o *testing*.

Para la infraestructura, se desplegará sobre el servicio en la nube de Microsoft Azure. Debe tenerse en cuenta que los precios de Azure son variables, por lo que los presupuestos son siempre estimados. El siguiente presupuesto se obtuvo el día 23/05/2022.

Microsoft Azure Estimate				
Su presupuesto				
Service type	Region	Description	Estimated monthly cost	Estimated upfront cost
Virtual Machines	West Europe	1 D4as v4 (4 vCPU, 16 GB de RAM) x 730 Horas (Pago por uso), Linux, (Pago por uso); 0 discos administrados: E15, 100 unidades de transacción; Tipo de transferencia interregional, 5 GB de transferencia de datos de salida de Oeste de Europa a Este de Asia	€159,83	€0,00
Storage Accounts	West Europe	Redundancia Almacenamiento de blobs en bloque, Uso general V2 y LRS, Acceso frecuente Nivel de acceso, Capacidad: 100 GB - Pago por uso, 10 x 10 000 operaciones de escritura, 10 x10 000 operaciones de lista y creación de contenedores, 10 x 10 000 operaciones de lectura, 100.000 operaciones de lectura de alta prioridad de Archive Storage, 1 x 10 000 operaciones de otro tipo. 1000 GB de recuperación de datos, 1000 GB de recuperación de alta prioridad de Archive Storage, 1000 GB de escritura de datos	€2,94	€0,00
Azure Database for PostgreSQL	West Europe	Implementación de Servidor flexible, Nivel Uso general, 1 D2v4 (2núcleos virtuales) x 730Horas (pago por uso), 20 GB de almacenamiento, 0 GB almacenamiento de copia de seguridad adicional: redundancia LRS, sin alta disponibilidad	€294,11	€0,00
IP Addresses	West Europe	0 direcciones IP dinámicas, 1 direcciones IP estáticas, 0 reasignaciones	€2,50	€0,00
Support		Support	€95,08	€0,00
		Licensing Program	Microsoft Customer Agreement (MCA)	
		Billing Account		
		Billing Profile		
		Total	€554,46	€0,00
Disclaimer				
All prices shown are in Euro Zone – Euro (€) EUR. This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/ This estimate was created at 5/23/2022 5:35:23 PM UTC.				

Fig. 13 – Imagen del presupuesto estimado de la infraestructura en Azure

Teniendo la estimación de coste para la infraestructura, en Tabla 14 se desglosan junto con este los siguientes costes aproximados para conseguir una versión productiva. En el caso de las horas de trabajo, estas llevan incluidas todos los costes indirectos que podrían esperarse (impuestos, facturas, lugar de trabajo, equipos, etc.):

Tabla 14 – Conceptos del presupuesto inicial

Concepto	**Coste**	**Tipo de coste**	**Total**
120 horas de desarrollo del software	26 €/h	Puntual	**3120,00 €**
48 horas de arquitectura del software	38 €/h	Puntual	**1824,00 €**
40 horas de implantación e infraestructura	24 €/h	Puntual	**960,00 €**
Licencia de desarrollador de Google Play	25 €	Puntual	**25,00 €**
Coste de infraestructura del *backend* (*Véase* Fig. 13)	554,46 €/mes	Periódico	**6653,52 €**
Coste de licencia de *API* para obtención del catálogo	20 €/mes	Periódico	**240,00 €**

Conjuntamente, se requerirían 5.929 € puntuales + 6.893,52 € anuales de los gastos periódicos, **12.822,52 €** en total, que estarían destinados a pagar las horas de trabajo, la infraestructura y la licencia de publicación en la Play Store de Google. También se incluye una bolsa de 20 €/mes en concepto de licenciamiento de *API*, en el

caso que el *scraping* sea inviable ni tampoco pueda usarse una alternativa gratuita. En caso de usar una *API* gratuita, esta bolsa se usará para bonificar el desarrollo de las interfaces que permitan utilizarla.

Adicionalmente, se aprovisionan para mantenimiento y correctivos que puedan surgir a lo largo del año, estos costes no se tendrían en cuenta para el primer mes del año donde se estaría construyendo el software. Esta bolsa de horas representada en Tabla 15 se aprovecharía para la realización de pequeños evolutivos si no fuera consumida en correctivos.

Tabla 15 – Conceptos del presupuesto de mantenimiento

Concepto	Coste	Tipo de coste	Total
64 horas de desarrollo del software	26 €/h	Periódico	**1664,00 €**
16 horas de arquitectura del software	38 €/h	Periódico	**608,00 €**
16 horas de infraestructura	24 €/h	Periódico	**384,00 €**

Sumados obtendríamos 2.656 € que se pagarían mensualmente durante 11 meses, **29.216 €**. Además, y por precaución, se agrega un 10% extra a la suma de los valores anteriores en concepto de imprevistos, siendo **2.921,6 €** adicionales.

Tras agregar todos los costes anteriores, el proyecto anual supondría un coste de **44.960,12 €**, que incluirían los costes de desarrollo, infraestructura y mantenimiento durante un año. En el caso de vender este proyecto a un tercero en lugar de realizarse como proyecto personal, se marcaria un incremento del 50% en concepto de beneficio (22.480,06 €). La factura final tendría un valor de **67.440,18 €**.

4. Diseño de la solución

El sistema está basado en una aplicación para Android, que contacta con un *backend* para obtener la información de las cuentas y los catálogos. Dicho *backend* consta de varios servicios web respondiendo desde un apache tomcat y una base de datos PostgreSQL que almacena la información. Adicionalmente el *backend* contacta con los servicios de la *API* de TMDB para rellenar y actualizar los catálogos de los distintos servicios de *streaming*.

Arquitectura del sistema

Tal como se indicó en el capítulo introductorio, la arquitectura del sistema está basada en la metodología por capas, si bien profundizaremos ahora en la variante específica que se utiliza para cada subsistema.

El *backend* carece de interfaz gráfica, por lo que no tiene una capa de presentación. Por otro lado, al actuar como la API personalizada de StreamBudget, dispone de interfaces en forma de varios servicios HTTPS considerándose esto una de las capas. Las otras dos capas que lo componen son la capa de negocio, donde se encuentra la lógica de negocio del *backend* y la capa de persistencia, que contiene los objetos de acceso a datos (en adelante DAO) y la lógica para interactuar con la base de datos (en adelante *BDD)* de PostgreSQL. Se hace uso de la librería Hibernate, que entre otras cosas permite hacer una equivalencia de las tablas de *BDD* a clases Java, pudiendo codificar la lógica de persistencia de manera más sencilla al poder usar estas entidades y sus métodos de manera programática, en lugar de tener que utilizar sentencias SQL planas.

La aplicación Android utiliza una variante especial para este tipo de software. Partiendo de la arquitectura recomendada desde Android que puede verse en Fig. 14, StreamBudget utiliza las capas de presentación (UI Layer) y la capa de datos (Data Layer).

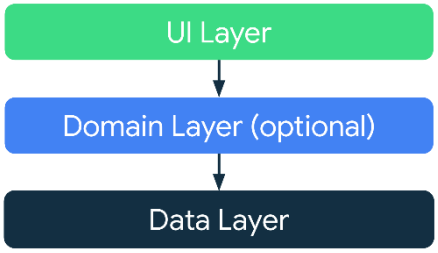

Fig. 14 – Arquitectura recomendada para Android. Obtenido de [23].

La capa de presentación tiene la función indicada con anterioridad, teniendo una especialización en el caso de Android que la divide en View y ViewModel.

Las View (o vistas) contienen únicamente los elementos gráficos que se muestran en la pantalla. Estas vistas se suscriben a los ViewModel, que funcionan como contenedores de estados. Estos contenedores contienen los datos para proporcionárselos a las vistas, además de estar encargados de la lógica detrás de la interfaz gráfica, absorbiendo pues los ViewModel lo que en las arquitecturas clásicas serían la capa de lógica de negocio.

La capa de datos funciona como capa de persistencia. Está compuesta por clases Repository (o repositorio), que pueden manejar distintas fuentes de datos para extraer o persistir información, siendo los orígenes totalmente transparentes para la

lógica. Es necesario crear una clase repositorio por cada tipo de dato, teniendo cierta equivalencia con las entidades con las que se trabaja con Hibernate y reflejándose el modelo de datos de la *BDD* en estos repositorios.

Detalle de la infraestructura

Como se menciona en el capítulo 3, la infraestructura se aprovisionará en el servicio en la nube de Microsoft, denominado Azure. El uso de las tecnologías en la nube nos permite cubrir **RLB-01** de manera robusta. Se indicará en los siguientes párrafos las medidas que ofrece Azure para cada componente de la infraestructura.

El modelo de datos queda almacenado en una *BDD* PostgreSQL 13.7, ofrecida en la modalidad software como servicio (en adelante *SaaS*). El hecho de usar la *BDD* en *SaaS* nos permite abstraer gran parte los procesos de instalación, configuración y mantenimiento de la *BDD*. Las condiciones del *SaaS* para la *BDD* en Azure prometen una disponibilidad de 99,99%, así como opciones de redundancia geográfica locales y globales, respaldos automáticos, actualización de parches menores y una variedad de herramientas de monitorización.

El servidor de aplicaciones utiliza una máquina Virtual en Azure, cuyas características pueden verse en Fig. 13. Se opta por el sistema operativo Ubuntu 18, basado en Linux. El servidor de aplicaciones utilizado es un Apache Tomcat 10.0.22. Además, se configura un Apache HTTPD que actuará como proxy para las peticiones. En Fig. 15 puede verse el diagrama representativo de la infraestructura inicial.

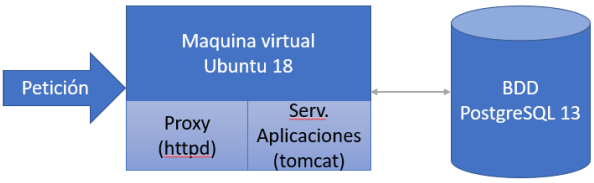

Fig. 15 – Diagrama de infraestructura inicial

En caso de requerirse un escalado del sistema, podría realizarse de manera sencilla. La máquina actual podría replicarse N veces, cada una de ella dispondría de su propio Apache Tomcat, pero no contendrían el proxy. En la maquina original, se agregaría la configuración al proxy para convertirlo en un proxy-balanceador, repartiendo la carga entre los diversos nodos disponibles. Dado que todas las llamadas al *backend* son transaccionalmente unitarias, es independiente qué nodo responda a cada llamada. En Fig. 16 puede observarse un diagrama ejemplificando este escenario.

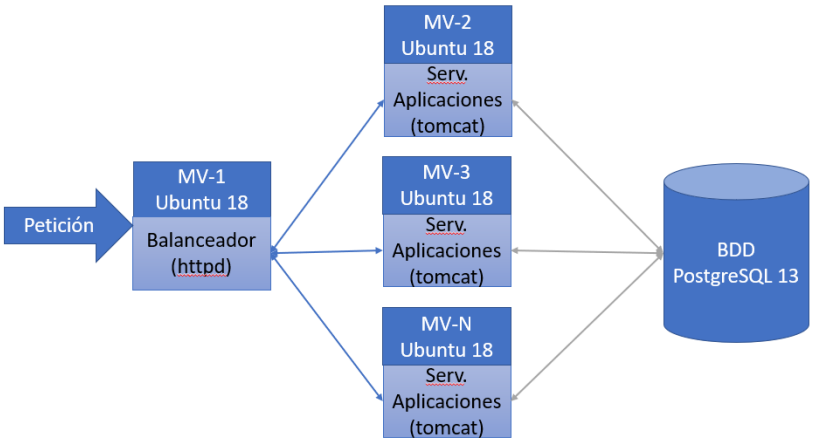

Fig. 16 – Diagrama de ejemplo de infraestructura escalada

Modelo de datos detallado

El modelo de datos que se ha diseñado para StreamBudget tiene el objetivo de reducir la cantidad total de información que se manejará a únicamente la necesaria para poder alimentar las funciones de la aplicación, refactorizándolo además para que sea más sencilla y natural de consultar para los procesos que la requieran. Además, el modelo puede ser ampliado, modificado o extendido de manera sencilla a la hora de mejorar o agregar funcionalidades.

El modelo se ha divido en dos esquemas principales. Por un lado, el esquema **sb_catalog_data** contiene las tablas que representan el submodelo de los catálogos, es decir, la información de los proveedores de streaming, sus planes de pago, y sus contenidos (películas y series). Por otro lado, el esquema **sb_user_data** contiene las tablas representativas del submodelo de datos de usuarios, siendo estas los datos de la cuenta y su lista “quiero ver”. Además, este último esquema dispone de las tablas para una posible funcionalidad social de listas de amigos, si bien finalmente no quedó implementada en la primera versión de la aplicación.

En las siguientes secciones se detallarán cada una de las tablas de cada esquema y se incluirán los esquemas entidad-relación de cada uno de ellos, pero antes es conveniente explicar algunos conceptos que aplican al modelo de manera global.

Obviando lo índices de creación automática, todas las columnas que contienen una referencia foránea (en adelante *FK*) tienen su propio índice para agilizar las consultas.

Los campos de fecha y hora se guardan en la *BDD* sin zona horaria (*timestamp without timezone*). Debido a la arquitectura de la aplicación, el *backend* que es encargado de la persistencia siempre estará en la misma zona horaria que, a nivel de infraestructura, siempre estará configurada como tiempo universal coordinado (en adelante *UTC*). Si estos datos se tuvieran que representar en la aplicación en algún momento, es muy sencillo de manera programática recuperar el uso horario del dispositivo (siendo parte de la configuración regional del dispositivo) y realizar las

operaciones de suma o resta para que se muestre actualizada a la zona horaria del usuario.

Existen 4 columnas que se repiten a lo largo del modelo, que serán referidas como columnas de control. Estas columnas permiten cierto control y trazabilidad temporal sobre los datos.

- **Insert_date**: hace referencia a la fecha donde se insertó el dato.
- **Update_date**: hace referencia a la fecha donde el dato se actualizó por última vez.
- **Deleted:** permite poder hacer borrados lógicos de los datos, es decir, marcar un dato como eliminado para que no se muestre sin necesidad de eliminar el registro de manera completa de la *BDD*.
- **Version:** permite conocer cuantas veces se ha modificado un dato.

Catálogos (sb_catalog_data)

En este modelo, representando en Fig. 17, se aprovecha el concepto de herencia que permite PostgreSQL a la hora de definir sus tablas mediante el uso de *INHERITS.* Similar a los conceptos de herencia en la programación orientada a objetos, la herencia entre tablas nos permite extender una tabla de diversas maneras, manteniendo una tabla con datos comunes y una serie de tablas extendidas para diversos propósitos.

Puede observarse en el diagrama entidad-relación que las claves primarias (en adelante *PK*), normalmente representada en la columna **id** de la tabla, aparecen como tipo SERIAL o tipo INTEGER dependiendo de la tabla. Cabe aclara que el tipo SERIAL es internamente el tipo INTEGER, pero con una secuencia auto incremental para generar claves únicas de manera automática. En los casos donde las *PK* son parte de una tabla de cruce (v.g. provider_content) o se utiliza el id obtenido por la *API de* TMBD para mantener la sincronía se utiliza el tipo INTEGER. En los casos donde deben generarse claves propias, se utiliza el tipo SERIAL. Todas las relaciones mostradas son uno a muchos (1:N), ya que las relaciones muchos a muchos (N:N) se indican mediante las tablas intermedias.

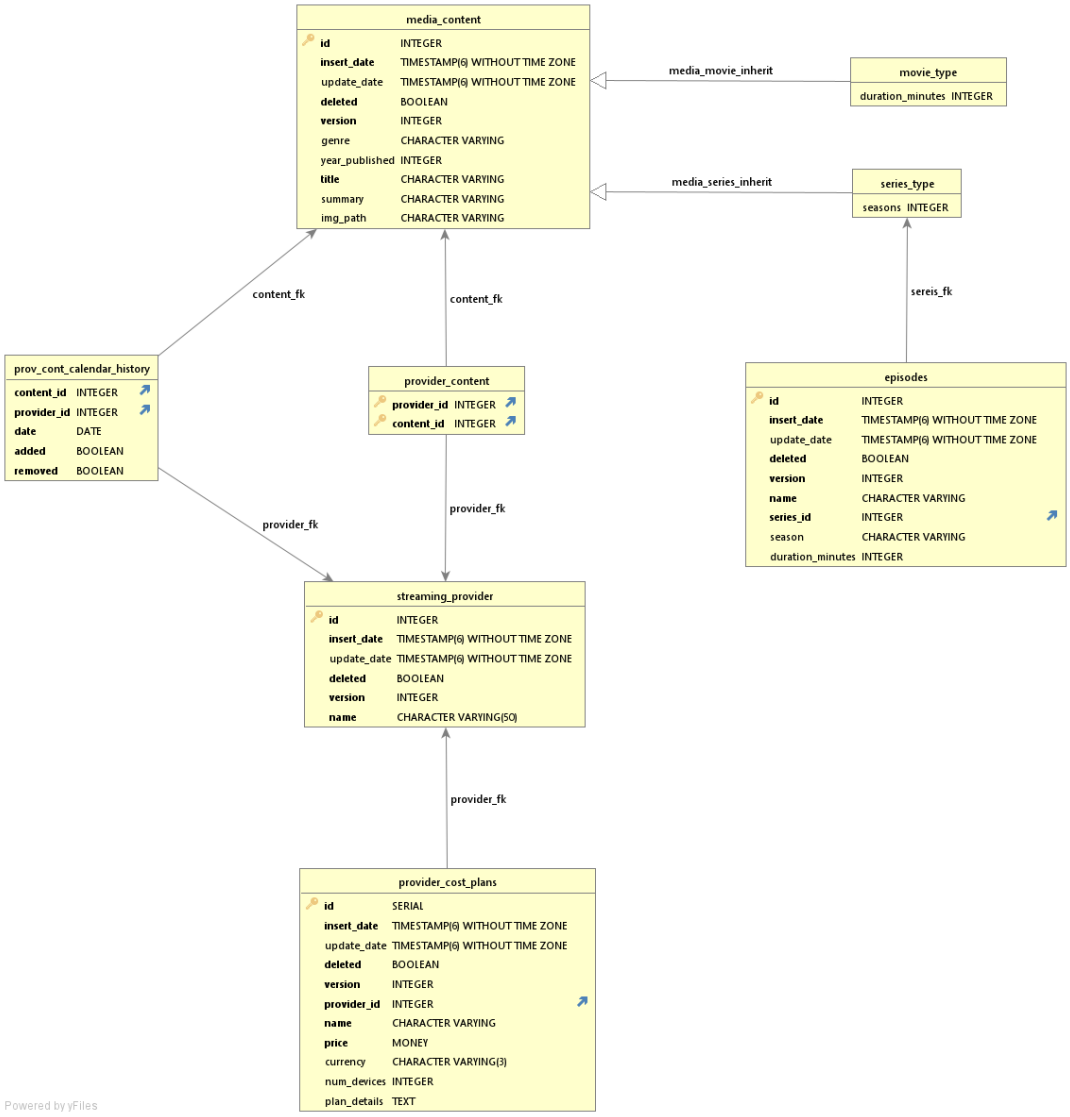

Fig. 17 – Modelo entidad-relación del esquema de catálogos

media_content: representa el contenido multimedia genérico, los tipos específicos de contenido heredan los datos de esta tabla. Contiene el título, géneros, año de publicación, ruta de la imagen de portada y descripción del contenido.

movie_type: representa el tipo película, heredando todos los valores de media_content. Esta tabla de extensión agrega la duración en minutos de la película.

series_type: representa el tipo serie, heredando todos los valores de media_content. Esta tabla de extensión agrega las temporadas que tiene la serie.

episodes: contiene la información de los episodios, nombre del episodio, temporada y una *FK* a la serie a la que pertenece (series_fk).

streaming_provider: contiene la información de los proveedores de *streaming*, necesitándose únicamente su nombre.

provider_cost_plans: contiene la información de los planes y precios de cada servicio de *streaming*. Los datos registrados son nombre del plan, precio, divisa, numero de dispositivos que permite el plan de manera concurrente y un texto con los detalles del plan. Además, guarda la referencia foránea (provider_fk) del servicio de *streaming* al que pertenece cada plan.

provider_content: tabla intermedia que contiene la información de qué contenido está disponible en cada proveedor, requiriendo de *FKs* a la tabla de proveedores (provider_fk) y contenidos (content_fk).

prov_cont_calendar_history: esta tabla se utiliza para la función del calendario de contenidos. Cuando el sistema detecta que se ha agregado o eliminado un contenido a un servicio de *streaming*, crea un registro en esta tabla. La tabla guarda las *FKs* a la tabla de proveedores (provider_fk) y contenidos (content_fk), así como la fecha del registro y dos columnas booleanas para indicar si es una adición o una eliminación. Estas columnas contienen una restricción (*Véase* Fig. 18) que impide que ambas tengan el mismo valor, debiendo ser siempre una verdadera y otra falsa. Dado que la funcionalidad **CU-08** solo muestra datos del último mes, la tabla es purgada al inicio de cada mes para no contener información innecesaria.

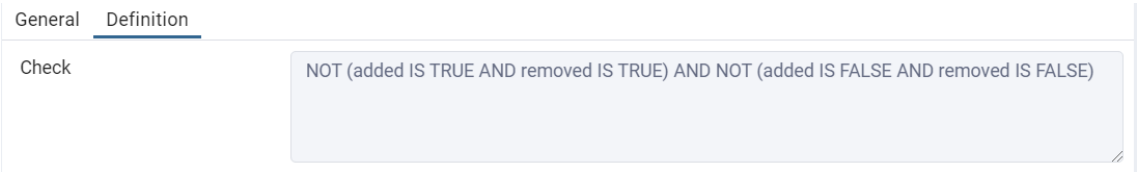

Fig. 18 – Restricción en las columnas added y removed

Datos de usuario (sb_user_data)

En este modelo, representando en Fig. 19, es más simple que el anterior. No se ha requerido el uso de herencias y, dado que todos los datos de este modelo se generan a raíz del uso de la aplicación, todas las *PKs* son autogeneradas. Del mismo modo que en el modelo anterior, todas las relaciones mostradas son 1:N, usándose tablas intermedias para las N:N.

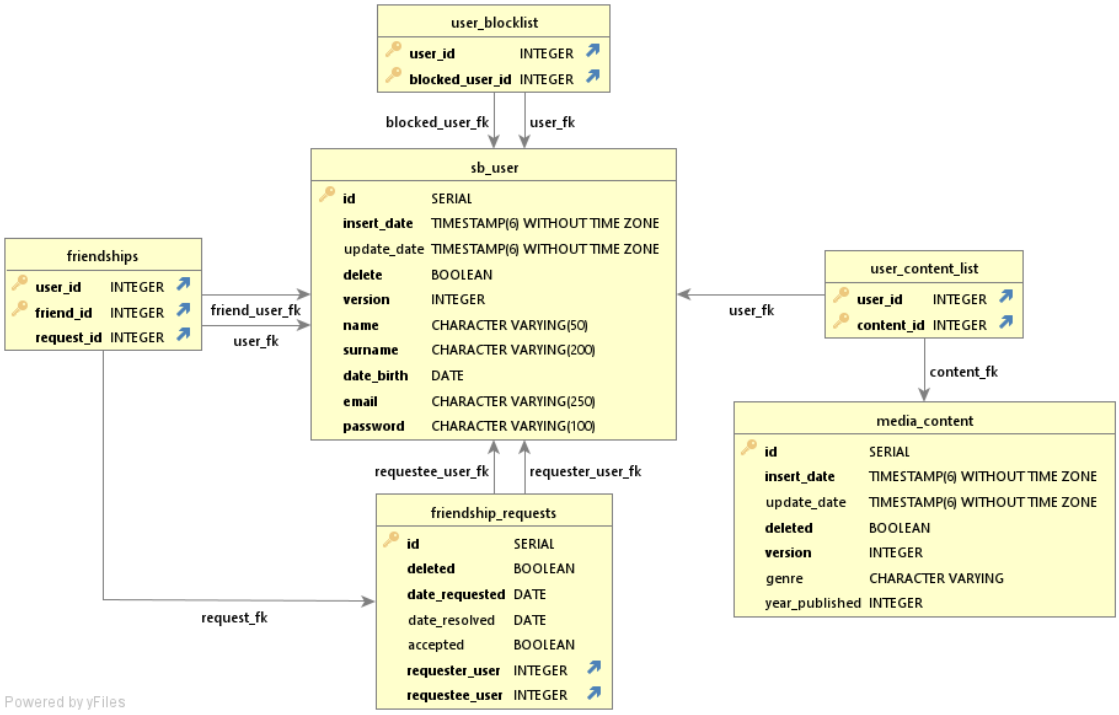

Fig. 19 – Modelo entidad-relación del esquema de datos de usuario

sb_user: representa a un usuario y contiene sus datos personales. Estos datos son su nombre, apellidos, fecha de nacimiento, email y contraseña. Se ha optado por un único campo de apellido pensando en futuras expansiones a otros países, donde culturalmente o bien solo se tiene un apellido o se tienen más de dos. En el caso de España, ambos apellidos quedarán registrados en esta tabla. El borrado lógico en el caso del usuario (campo deleted) se utilizaría en caso de que ejercieran su derecho de oposición o limitación de tratamiento. En caso de ejercer su derecho de supresión siempre se usaría un borrado físico, eliminado el registro por completo.

user_content_list: tabla intermedia que contiene la lista “quiero ver” de cada usuario. Permite la relación entre usuario (user_fk) y contenido (content_fk), siendo estas las *FKs* que se incluyen la tabla.

friendship_requests: contiene las peticiones de amistad. Cada petición consta de una fecha de solicitud, una fecha de resolución y un booleano que indica si fue aceptada o rechazada, además contiene las *FKs* al usuario solicitante (requester_user_fk) y al usuario solicitado (requestee_user_fk). Esta tabla se utilizaría como histórico de peticiones, quedando las relaciones de amistad guardas en la tabla friendships. Aún no utilizada.

friendships: lista de amistades entre usuarios. Formada por una relación compuesta por las *FKs* que representan a cada usuario (user_fk y friend_user_fk) y una *FK* (request_fk) a la petición de amistad que se hizo. Tiene redundancia de datos con la anterior, pero es beneficiosa ya que al usarla no es necesario filtrar por tantos campos y se agiliza la consulta. Aún no utilizada.

user_blocklist: lista de bloqueos entre usuarios, para impedir que se sigan enviando solitudes de amistad. Contiene las *FKs* que representan al usuario (user_fk) y al usuario que es bloqueado (blocked_user_fk). Aún no utilizada.

Maquetas de las interfaces de usuario

Se han diseñado una serie de maquetas (también conocidas como *mockups*) de las pantallas de la futura aplicación móvil.

Fig. 20 – Maquetas de inicio de sesión, registro y cuenta de usuario

Las pantallas mostradas en Fig. 20 satisfarían los siguientes requisitos funcionales:

- CU-02 y CU-10 (pantalla inicio de sesión).
- CU-01 (pantalla registro).
- CU-11 y CU-12 (pantalla cuenta).

Fig. 21 – Maquetas de catálogo, calculador de recomendaciones y calendario

Las pantallas mostradas en Fig. 21 satisfarían los siguientes requisitos funcionales:

- CU-03, CU-04, CU-05, CU-06 y CU-07 (pantalla catálogo).
- CU-09 (pantalla calculadora de recomendaciones).
- CU-08 (pantalla calendario).

Adicionalmente, en Fig. 22 se muestra el diagrama de navegación entre las distintas pantallas.

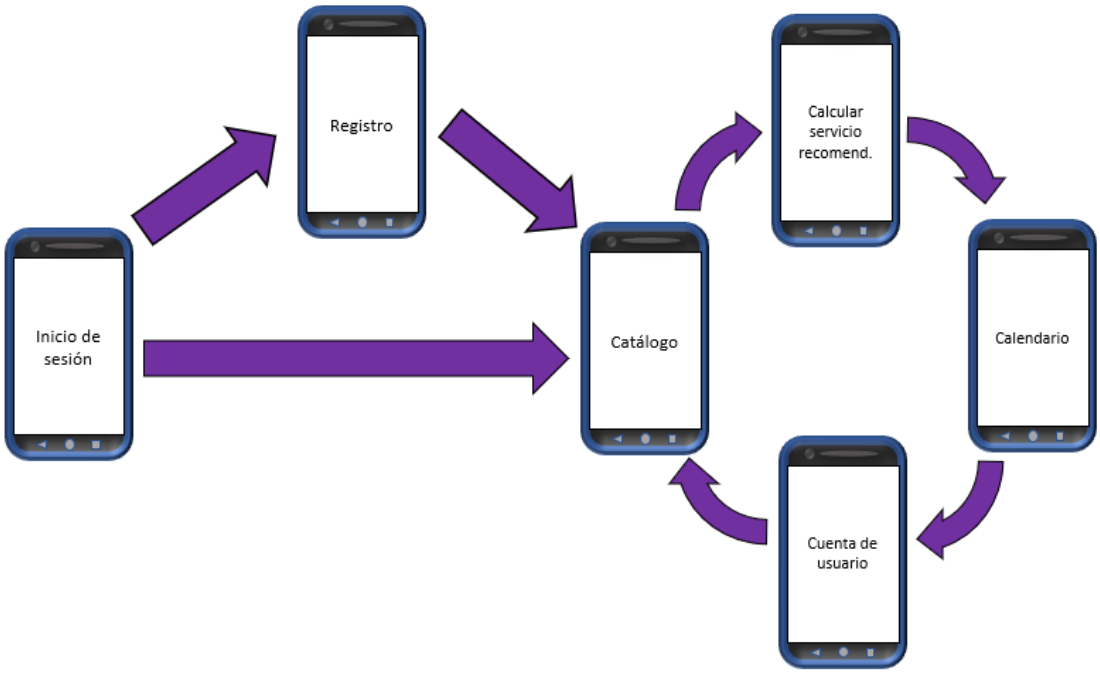

Fig. 22 – Diagrama de navegación entre pantallas

Tecnologías y herramientas utilizadas

Las diferentes tecnologías utilizadas se agruparán por categorías para facilitar su lectura.

Desarrollo

Se ha valorado la utilización de diferentes plataformas de desarrollo en Android como React Native[1], Ionic[2] y Flutter[3], pero finalmente se ha optado por la utilización de Android SDK por no necesitar generar código multiplataforma y por experiencia previa del desarrollador además de tener de las mejores documentaciones en español y su propio entorno de desarrollo, Android SDK Studio[4], que incluye emuladores de teléfonos Android fácilmente configurables para hacer cualquier prueba.

[1] https://reactnative.dev/
[2] https://ionicframework.com/
[3] https://flutter.dev/
[4] https://developer.android.com/studio

La programación nativa en Android puede realizarse en Java o Kotlin, decidiéndose para este proyecto el uso de Java, utilizándose en algunos casos algunas clases auxiliares en Kotlin. Si bien Kotlin es un lenguaje más nuevo que está ganando cada vez más popularidad sus características más fuertes no aportan gran beneficio a este proyecto y el desarrollador cuenta ya con experiencia en Java. Siendo algo más detallados, Kotlin es compatible con el desarrollo multiplataforma y con respecto a Java tiene un rendimiento superior y sería el candidato perfecto para aplicaciones que demanden mucho del dispositivo, como por ejemplo la edición audiovisual, pero ninguna de estas características es requerida por este proyecto. Por otro lado, Java es un lenguaje más asentado, con años de uso para desarrollo en Android además de otras plataformas, que cuenta con una comunidad mayor y una documentación considerada muy buena y fácil de entender, esto sumando a la actual experiencia en desarrollo Java se decanta el uso de este lenguaje.

Es importante destacar el uso de Gradle[5] como tecnología de compilación y construcción de código, similar a lo que se haría con Apache Ant[6] o apache Maven[7]. Gradle viene incorporado en Android SDK Studio para realizar las tareas de construcción del código.

Respecto al uso de librerías, se han utilizado Ion en el proyecto Android, ya que contiene una funcionalidad que permite cargar las imágenes en los ImageView de manera dinámica y asíncrona desde una dirección web, pudiendo así utilizar las imágenes almacenadas en los servidores de *TMBD* en lugar de tener que almacenarlas en la aplicación o en el *backend*, que hubiera ocasionado un aumento importante del tamaño de la aplicación o del tráfico de red. La otra librería utilizada es Hibernate[8] en el *backend*, ya mencionada con anterioridad, que permite trabajar en código con Entidades que representan cada tabla en la BDD, haciendo más sencilla el manejo de datos.

Durante el desarrollo de la aplicación se ha emulado un smartphone Google Píxel 2 con un sistema operativo Android 11.0 (Android R) mediante Android Studio para poder hacer pruebas de su funcionamiento.

El software se ha desarrollado para una versión mínima Android 8.0 y una versión objetivo Android 11.0. La versión mínima hace referencia a la versión más baja de Android que la aplicación es capaz de soportar y funcionar correctamente con toda su funcionalidad, aunque el desarrollo se haga para una versión objetivo más actual. Como se puede ver en Fig. 23, al elegir Android 8.0 como versión mínima debería hacer posible llegar al 82,7% de los dispositivos Android globales, ya que la distribución acumulada identifica el porcentaje de dispositivos que utilizan una versión o superior. Un casi 83% de dispositivos se considera un alcance de mercado suficiente, no mereciendo la pena reducir la versión objetivo a una más baja (con la penalización de compatibilidad al codificar).

[5] https://gradle.org/
[6] https://ant.apache.org/
[7] https://maven.apache.org/
[8] https://hibernate.org/

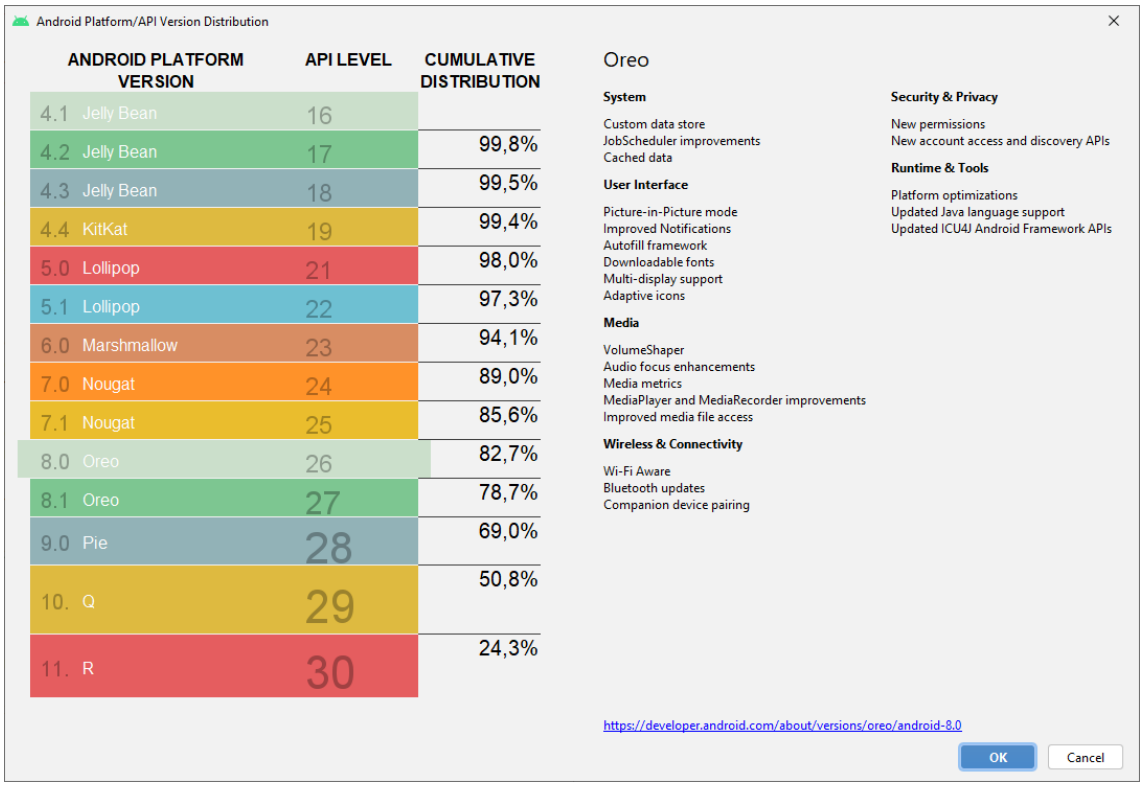

Fig. 23 – Imagen de la distribución de dispositivos por versión de Android

Para poder probar las respuestas de las *API* se utilizó la herramienta Postman[9]. En sus inicios esta herramienta era un complemento de navegador, pero ha evolucionado hasta ser una aplicación completa. Además de su función principal de consulta, dispone de una serie de utilidades que facilitan el trabajo de pruebas, como guardar variables para poder usar en distintas llamadas, pudiendo incluso utilizar la respuesta de una llamada para refrescar una variable y usarla en otra llamada. También permite emular una API, es decir, esperar una petición y dar una respuesta predefinida, que, si bien no se usó, es una funcionalidad muy útil para poder probar software si aún no está disponible la API final.

Base de datos

Como se ha mencionado anteriormente, el motor de BDD utilizado es PostgreSQL[10], un proyecto de código abierto con más de 30 años de desarrollo activo. Es un motor robusto, ofreciendo una solución gratuita y libre capaz de competir con motores empresariales como el de Oracle[11].

Se utilizaron dos sistemas gestores de base de datos (en adelante *SGDB*). El primero es pgAdmin4, el *SGDB* por defecto ofrecido por PostgreSQL. Este software se utilizó para la creación del modelo de datos y la gran mayoría de interacciones SQL que se requirieron. El segundo fue DbVisualizer[12], un *SGDB* multiplataforma que cuenta con una versión gratuita y una versión de pago. El autor tenía experiencia previa con la herramienta y se utilizó para generar los modelos entidad-relación del modelo de datos,

[9] https://www.postman.com/
[10] https://www.postgresql.org/
[11] https://www.oracle.com/es/database/
[12] https://www.dbvis.com/

ya que los que genera esta herramienta son considerablemente más legibles que los generados por pgAdmin4.

Infraestructura y despliegue

La infraestructura se aprovisionó en Microsoft Azure[13], que ya ha sido descrito anteriormente.

A nivel de despliegue, cabe mencionar el uso de dos productos de apache. El primero sería el servidor de aplicaciones, tomcat[14], que tiene un uso muy extendido en el mercado y se prefirió a otras alternativas como Jetty[15] de la fundación Eclipse o Weblogic[16] de Oracle. Para la gestión del tráfico de red, se eligió httpd[17] que actúa como proxy y podría actuar como balanceador, en este caso se eligió por contar con experiencia previa, pero podrían haberse utilizado otras alternativas, como ngix[18].

Para poder acceder a los servidores en la nube, se utilizaron las herramientas WinSCP[19] para el acceso FTP (protocolo de transferencia de ficheros) y PuTTY[20] como cliente SSH para la ejecución de comandos de consola. Estas dos herramientas funcionan muy bien en conjunción ya que WinSCP cuenta con integraciones específicas para PuTTY.

[13] https://azure.microsoft.com/es-es/
[14] https://tomcat.apache.org/
[15] https://www.eclipse.org/jetty/
[16] https://www.oracle.com/es/java/weblogic/
[17] https://httpd.apache.org/
[18] https://www.nginx.com/
[19] https://winscp.net/eng/index.php
[20] https://www.putty.org/

5. Desarrollo de la solución propuesta

A lo largo de las implementaciones, se siguieron los diseños, las arquitecturas y se satisficieron los requisitos indicados en los capítulos anteriores.

Uno de los puntos críticos en el desarrollo ocurrió durante la implementación de lógica de cálculo del servicio óptimo (requisito CU-09). Inicialmente esta lógica se incluyó en la capa de datos de la aplicación móvil y si bien funcionaba y con pocos datos no existía problema, cuando el número de elementos de la lista “quiero ver” era grande, los tiempos de espera hasta que se obtiene respuesta eran superiores a 2 segundos si el dispositivo era de gama baja, incumpliéndose el requisito RND-01. Con el tiempo disponible y tras valorar opciones se optó por mover la lógica al *backend*, ya que al estar ambos codificados en Java era sencillo refactorizar el código para usarlo en el *backend*. Se tuvo en consideración que, si el usuario dispone de una conexión capaz de usar servicios de *streaming*, no debería haber problema en solicitar el cálculo al *backend* en lugar de realizarse en local. Además, dado que no se guardan los catálogos localmente, ya se requiere una conexión a internet para el uso de la aplicación. Esto además nos proporciona una serie de ventajas adicionales.

- Dado que el hardware del *backend* es siempre el mismo, los resultados de rendimiento deberían ser más deterministas. Delegar la lógica en el *backend* debería dar rendimientos más igualados entre distintas gamas de dispositivos.
- Al hacerse el cálculo en el *backend*, se tiene ya la base para una funcionalidad futura donde se notifique al usuario de cambios en sus recomendaciones. El *backend* podría hacer el cálculo y enviar la notificación push al dispositivo o un correo electrónico al usuario.

Continuando con el *backend*, su capa de persistencia está compuesta por todas las clases DAO que se utilizan para interactuar con la BDD. En su capa de negocio se han implementado la lógica que consulta la *API* de TMDB y toda la lógica necesaria para poder resolver las solicitudes de la capa de interfaces.

Para la implementación de los algoritmos, se realiza un preprocesado al arrancar el *backend* y se persiste en un objeto HashMap<integer, mediaData>. La clave utilizada es el mismo identificador de la película y el objeto mediaData es una estructura de datos personalizada, que contiene el peso que tiene el contenido para cada servicio y algoritmo. La estructura tiene forma de árbol, siendo la navegación algoritmo > servicio > peso. Este objeto se carga al inicio y se utiliza el patrón Singleton, ya que necesitamos que solo exista una instancia de este objeto en memoria. Es importante mencionar que, si un contenido no existe para un servicio concreto, se guarda como peso 0 dentro de mediaData.

A la hora de calcular la puntuación de cada servicio y con la lista de identificadores y el código del algoritmo recibidos desde la aplicación, se recorre esta

lista en bucle utilizando una variable numérica para guardar el peso total por cada servicio. El bucle va accediendo al dato del peso concreto para cada película con el algoritmo elegido y simplemente se suman para cada servicio. Esto elimina la necesidad de lógicas comparativas dentro del bucle y únicamente tiene que realizarse la suma, siendo el tiempo necesario para cada calculo bastante deterministas, siendo el factor que afecta a los tiempos el tamaño de la lista recibida.

Las respuestas que ofrece la *API* de TMDB se actualizan una vez al día, por lo que desde el *backend* las peticiones para actualizar los catálogos se hacen también una vez al día, concretamente a las 3 am. Se utiliza un cliente HTTP para hacer las peticiones a la API, procesando los resultados para extraer los datos necesarios para StreamBudget para que después sean persistidos. Se recuerda que el valor de los identificadores de las tablas de los catálogos se registra con el mismo valor que devuelva TMBD en sus campos “ids”, lo cual facilita las interacciones posteriores con la API. Los puntos que se utilizan de la API de TMBD son los siguientes:

- **/discover/movie** y **/discover/tv** se utilizan para obtener el grueso del catálogo de películas y series respectivamente, ya que es el único servicio que ofrece el catálogo completo. Respecto a los argumentos enviados, se solicita el contenido del lenguaje español y cuya región de visionado sea España, sin incluir contenido adulto. Se filtra por popularidad, ya que este campo es requerido, aunque no sea relevante para StreamBudget. El argumento más importante, watch_providers, permite enviar una lista separada por comas de los proveedores de streaming, solicitándose aquí los que son relevantes para StreamBudget.
- **/tv/{tv_id}/season/{season_number} (TV Season – Get Details)** se utiliza para obtener la información de las temporadas y los episodios de cada serie.
- **/watch/providers/movie** se utiliza para obtener los proveedores de *streaming* de películas. Si bien existe un método específico para obtener proveedores de series, los proveedores que se utilizarán en StreamBudget ofrecen ambos tipos de contenido y sería redundante consultar ambas, puesto que los identificadores son los mismos (v.g. el identificador de Netflix es 8, tanto para películas como para series).

Finalmente, en la capa de interfaces, se expone una API propia. Como nombre del dominio, se utiliza el gratuito generado por Azure (streambudget. westeurope.cloudapp.azure.com). Los puntos más significativos expuestos en la API de StreamBudget se describirán a continuación en Tabla 16 a Tabla 20.

Tabla 16 – Tabla descriptiva de /media/catalogs

Tipo	GET			
Ruta	/media/catalogs			
Parámetros	provider genres year_pub	integer string integer	Proveedor streaming. Géneros cinematográficos. Año de estreno.	opcional opcional opcional
Respuesta	resultados[] películas[] título descripción proveedor			

	géneros[] año ruta_imagen duración proveedores[] series[] título descripción proveedor[] géneros[] año ruta_imagen temporadas num_episodios
Descripción	Obtener los catálogos de contenido, sus detalles y proveedores.

Tabla 17 – Tabla descriptiva de /media/calendar

Tipo	GET
Ruta	/media/calendar
Parámetros	Ninguno.
Respuesta	resultados[] proveedor[] adiciones[] título eliminaciones[] título
Descripción	Obtener la lista de adiciones y eliminaciones de los proveedores desde principio de mes.

Tabla 18 – Tabla descriptiva de /user/register

Tipo	POST			
Ruta	/user/register			
Parámetros	email	string	Correo electrónico.	requerido
	password	string	Contraseña (md5).	requerido
	date_birth	string	Fecha de nacimiento.	requerido
	name	string	Nombre del usuario.	requerido
	surname	string	Apellido/s del usuario.	requerido
Respuesta	resultado estado token_sesion			
Descripción	Registrar una cuenta de usuario. El estado indica si ha sido exitosa o errónea.			

Tabla 19 – Tabla descriptiva de /user/login

Tipo	POST			
Ruta	/user/login			
Parámetros	email	string	Correo electrónico.	requerido
	password	string	Contraseña (md5).	requerido
Respuesta	resultado estado token_sesion			

Descripción	Iniciar sesión en una cuenta de usuario. El estado indica si ha sido exitosa o errónea.

Tabla 20 – Tabla descriptiva de /media/getbestprovider

Tipo	POST			
Ruta	/media/getbestprovider			
Parámetros	to_view_list algorithm token	integer[] integer string	Lista de ids de contenido a ver. Código del algoritmo a usar. Token de sesión.	requerido requerido opcional
Respuesta	resultados[] proveedor puntuación			
Descripción	Solicita el cálculo de la puntuación de los servicios de *streaming* según el algoritmo seleccionado. Si se envía un token, la lista se guardará en *BDD* asociado al usuario al que pertenezca el token.			

Respecto a la aplicación móvil, se debe destacar un apartado importante del desarrollo en Android y es el hecho de que los dispositivos pueden estar en posición vertical (retrato o *portrait*) u horizontal (paisaje o *landscape*), debiendo desarrollarse en muchos casos para ambas orientaciones. Con las restricciones temporales en el desarrollo y al no verse especial ventaja en disponer del modo paisaje para esta aplicación, se utilizan unos parámetros en el manifiesto de Android (en adelante manifest) que le indican al sistema que esta aplicación solo puede usarse en modo retrato, forzando esta orientación mientras se usa la aplicación. Estos parámetros deben configurarse para cada actividad de la aplicación:

- android:screenOrientation="portrait"
- android:configChanges="orientation|keyboardHidden">

Entrando en la capa de ui, está compuesta por 3 actividades principales (Inicio de sesión, registro y menú de navegación). Las dos primeras, como su nombre indica, son las utilizadas para el inicio de sesión si ya se dispone de cuenta y para el registro de una cuenta de usuario. La más interesante es el menú de navegación, con un estilo de 4 iconos inferiores, nos permite movernos entre los distintos fragmentos que componen el resto de la aplicación (Catalogo, Recomendador, Calendario y Cuenta).

En Fig. 24, puede verse el árbol de código que representa la lógica capa de ui, se han desplegado solo un par de nodos a modo de ejemplo. Puede apreciarse que se ha respetado la arquitectura de Vista y Modelo-Vista (View y ViewModel).

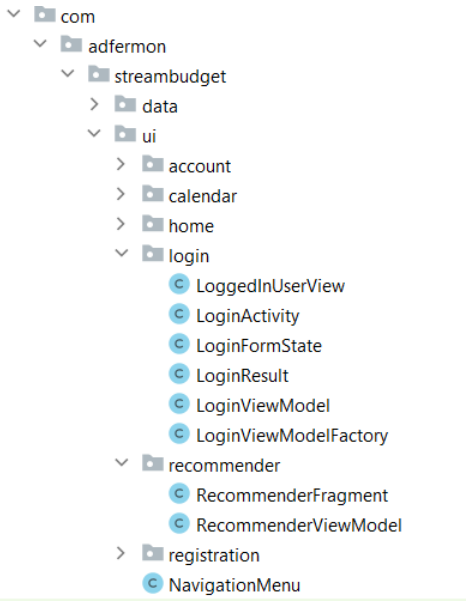

Fig. 24 – Muestra del árbol de la capa ui de la aplicación Android

Al iniciar la aplicación, se muestra la pantalla de inicio de sesión y de forma paralela el sistema trata de recuperar un token de sesión y los datos de la cuenta de la memoria local. Si estos datos existen, se redirige automáticamente al menú principal de navegación. En caso contrario, el usuario puede optar por iniciar sesión, que contactará con la *API* para recibir el token o registrarse, que si se completa con éxito también recibirá un token de sesión.

Una vez en el menú de navegación principal, el usuario puede saltar a cualquier apartado del menú en cualquier momento y desde cualquier pantalla. El primer apartado contiene el catálogo de contenido de los servicios de *streaming*. Aquí el usuario puede buscar, ordenar y filtrar este contenido y, lo más importante, marcarlo para su lista “quiero ver”. Esta lista no está disponible de manera directa en la aplicación, pero el usuario puede filtrar por los contenidos que ha marcado para ver como filtro del catálogo. Se hace uso de una RecyclerView, con un adaptador personalizado. Hay que destacar aquí el uso de la librería Ion que, como se menciona en capítulos anteriores, nos permite cargar las imágenes desde URLs de internet de manera asíncrona, usando las almacenadas en *TMBD*.

El apartado del Recomendador contiene la funcionalidad núcleo de la aplicación. Tras pulsar el botón calcular, se envía al *backend* la lista “quiero ver” (se envía la lista de identificadores de contenido) y el token de sesión de manera opcional. Si se envía el token, la lista quedará guardada en el *backend* asociada al usuario al que pertenece el token, pero no es obligatorio ya que se puede utilizar esta función como invitado. Una vez recibida la respuesta, se muestra la lista de los servicios de *streaming*, junto con el porcentaje de contenido que se podría visionar y un plan recomendado, visualizándose con otra RecyclerView similar a la del catálogo. La función de recomendar un plan concreto aún no está completamente desarrollada y en el futuro podría tener en cuenta los costes divididos al usar planes compartidos.

El apartado calendario muestra una lista ordenada por servicios de *streaming*, indicando los contenidos que se han agregado y eliminado de cada uno desde el principio del mes. Finalmente, el apartado de cuenta muestra los datos de la cuenta con la que se inició la sesión, o datos genéricos en caso de estar usando la aplicación como invitado.

En el caso de la capa de datos no hay nada especialmente destacable. En esta capa están los DataSources (orígenes de datos) que son los encargados de contactar con la API del *backend* o con el almacenamiento local y los correspondientes Repository (repositorios) que almacenan la información en memoria para su uso en la aplicación.

6. Resultado final

Arte y funcionalidad final

A continuación, se mostrarán y describirán capturas del arte final de la aplicación.

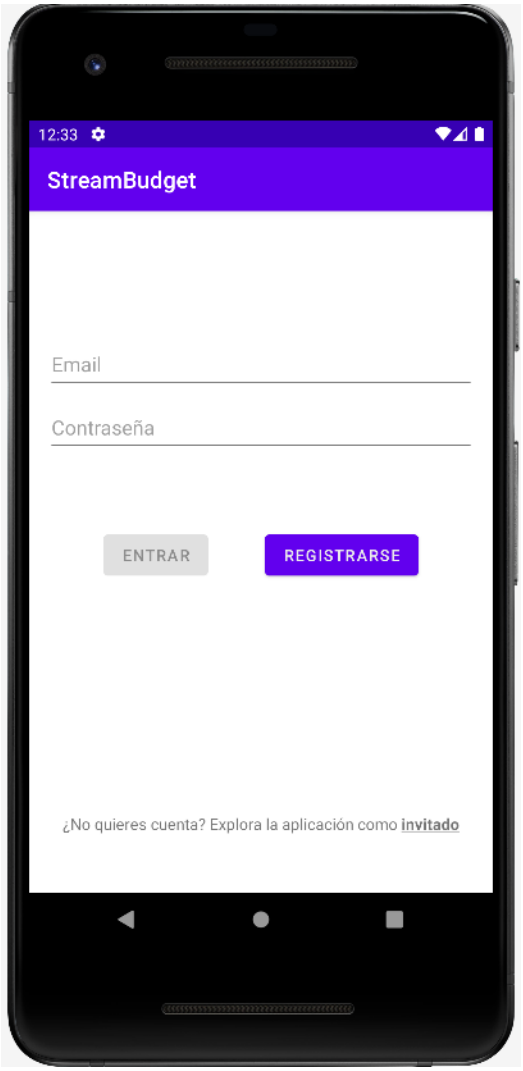

Fig. 25 – StreamBudget, pantalla de inicio de sesión

Fig. 26 – StreamBudget, pantalla de registro

En Fig. 25 y Fig. 26 pueden observarse las pantallas de inicio de sesión (login) y registro de StreamBudget. En el caso del login, el foco está en entrar con una cuenta o registrarse, si bien como se indica en el requisito **CU-02**, se permite el acceso a la aplicación como invitado (pulsando en el texto inferior). La pantalla de registro permite crear la cuenta de usuario, satisfaciendo el requisito **CU-01**. El check inferior solicita el consentimiento del usuario para el tratamiento de sus datos, no pudiendo completarse el registro si no se acepta. En el futuro, el texto de este check podría incluir un enlace a los términos y condiciones de la aplicación, pero en la aplicación actual si el usuario desea saber el uso que se dará a sus datos debe solicitarlo por correo electrónico. En cumplimiento del requisito **UX-03**, puede observarse que los botones “Entrar” de Fig. 25

y “Registrarse” de Fig. 26 aparecen inhabilitados al no estar cumplimentados todos los campos necesarios.

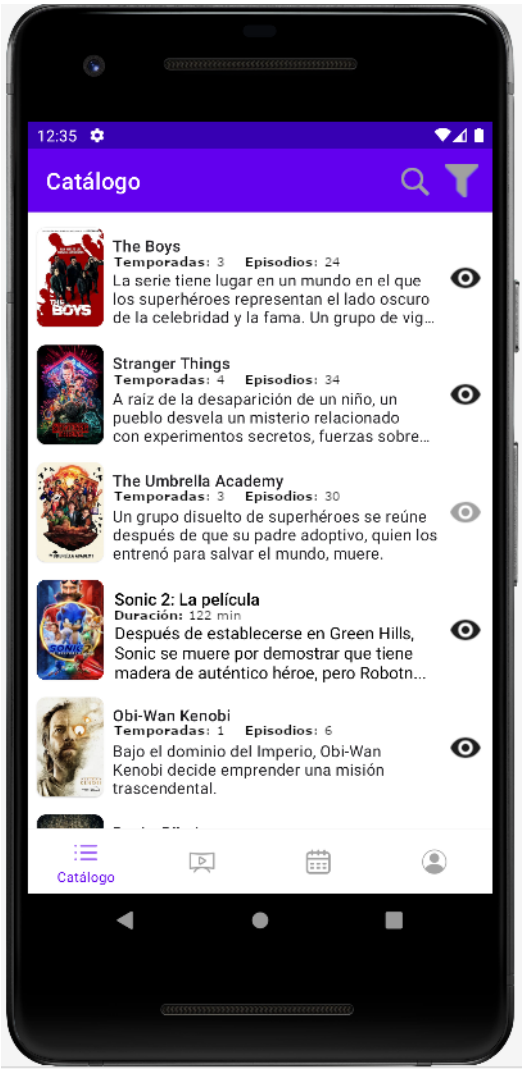

Fig. 27 – StreamBudget, pantalla de catálogos

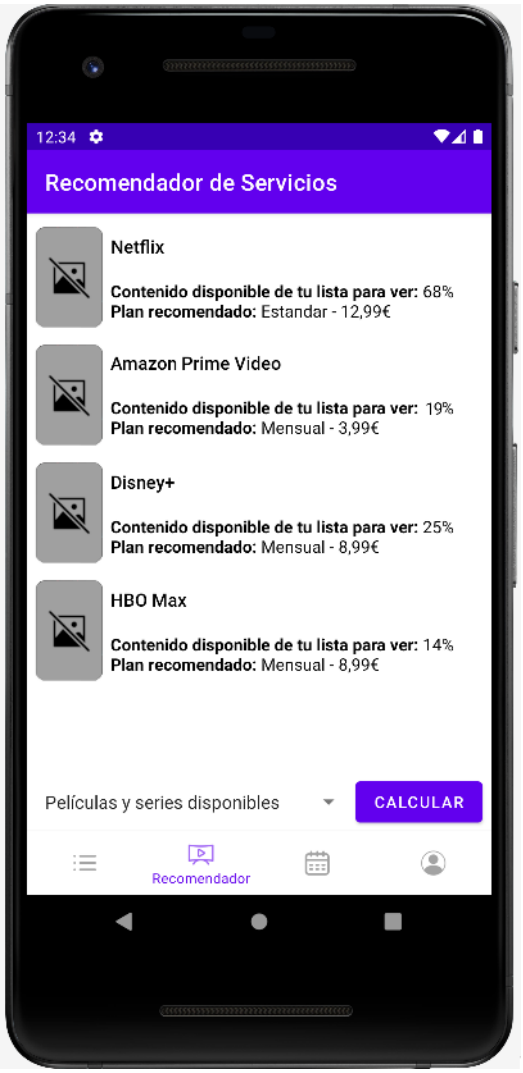

Fig. 28 – StreamBudget, pantalla de servicios recomendados

Las siguientes capturas muestran la funcionalidad principal de la aplicación. En Fig. 27 se puede ver la pantalla de catálogo, que contiene todo el contenido multimedia (series y películas) del que dispone la aplicación. Utilizando los iconos con forma de ojo puede agregarse o eliminarse contenido de la lista “quiero ver”, satisfaciendo los requisitos **CU-03** y **CU-04** respectivamente. Mediante el uso del icono de la lupa en la esquina superior derecha, el usuario puede buscar contenido dentro del catálogo (requisito **CU-05**), mientras que el icono con forma de embudo permite elegir el modo de ordenación y agregar opciones de filtrado de contenido (requisitos **CU-06** y **CU-07**). Los datos mostrados son el título del contenido y un resumen, agregándose el número de episodios y temporadas para las series y la duración para las películas. Como trabajo futuro, se ampliaría esta pantalla para mostrar más datos del contenido en una pantalla propia (detalles del contenido), pudiendo mostrar así año de publicación, género u otros datos que actualmente están disponibles en el *backend,* pero no se muestran en la aplicación; pudiendo agregarse también más datos si se acompaña de una ampliación del modelo de datos.

En Fig. 28 aparece el “Recomendador de servicios”, que permite elegir un algoritmo y calcular que servicio sale más rentable en base al criterio elegido (requisito **CU-09**). Lo primero que llama la atención es el uso de imágenes, ya que en un principio no se tenía pensado utilizarlas y no hay soporte en el modelo, pero como trabajo futuro prioritario se modificaría el *backend* para guardar el logo de cada servicio y poder mostrarlo aquí. Los resultados se muestran tal como se describió en el apartado de diseño, pudiendo verse en la parte inferior el selector de algoritmo y el botón que solicita el cálculo. El cálculo de posición se realiza dividiendo el coste entre el porcentaje disponible, cuanto menor sea el valor de dicha operación, más rentable se considera el servicios y más alto aparece. El porcentaje mostrado depende del algoritmo usando:

- Para la opción “Películas y series disponibles” muestra el porcentaje del número total de elementos de la lista disponibles en el servicio.
- Para la opción “Número de minutos totales” muestra el porcentaje del total de minutos tras sumar la duración de las películas y las duraciones de los episodios de las series seleccionadas disponibles en el servicio.
- Para la opción “Películas y episodios disponibles” muestra el porcentaje del número total de películas + total de episodios de cada serie seleccionada disponibles en el servicio.

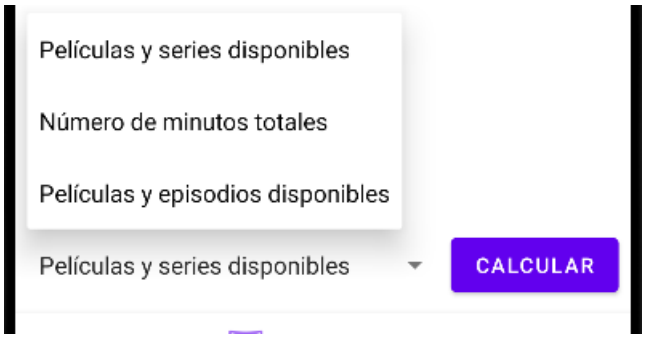

Fig. 29 – StreamBudget, selector de algoritmo

Finalmente, se muestran las pantallas del calendario en Fig. 30 y de cuenta de usuario en Fig. 31. Respecto al calendario, se muestra el mensaje que indican que no hay nada que mostrar. Esto se debe a que, dado que el *backend* ha estado funcionando menos de un mes natural, todo el contenido se consideraba nuevo, por lo que se optó por purgar la tabla, no usando tampoco datos de prueba. Esto además daba pie a poder mostrar una pantalla sin contenido, que simplemente muestra un TextView con un mensaje indicándolo. Todas las pantallas tienen un mensaje similar para indicar que no recibieron datos, pero este se oculta en el supuesto de que sí que se reciban. En caso de recibirlos, se mostraría una lista con los cambios por cada servicio (requisito **CU-08**), siendo todo un único texto (muy similar al diseño maquetado de la pantalla).

En el caso de la pantalla de cuenta de usuario, podemos observar los datos de la cuenta que tiene la sesión iniciada, con un botón que permite editar cada uno de los campos (requisito **CU-11**). Para persistir estos cambios, el usuario debe presionar el botón del disquete (guardar). En caso de querer eliminar la cuenta (requisito **CU-12**), el usuario puede pulsar en esta pantalla el icono de la papelera en la esquina superior derecha, tras lo cual se le solicitara confirmación de que realmente quiere eliminar su

cuenta y en caso afirmativo, se llama a la lógica del *backend* para purgar los datos de la cuenta.

Fig. 30 – StreamBudget, pantalla de calendario

Fig. 31 – StreamBudget, pantalla de cuenta de usuario.

Pruebas

Respecto a las pruebas, se considera que se han realizado pruebas suficientes para asegurar el funcionamiento de la aplicación y el cumplimiento de los requisitos. Sin embargo, en ciclos futuros, deberían introducirse una serie de mejoras en el proceso de prueba, incluyéndose entre otras la realización y cumplimiento de las pruebas de calidad básica de las apps [12] además de la realización de pruebas en dispositivos físicos además de virtuales.

A continuación, se hablará sobre lo requisitos no funcionales que no hayan sido mencionados con anterioridad en este capítulo. En el grupo de experiencia de usuario y usabilidad se considera **UX-01** parcialmente cumplido, ya que todas las navegaciones entre menús cumplen el requisito de necesitar 4 acciones o menos, pero algunas interacciones pueden no ser intuitivas, por ejemplo, el uso de los iconos de acción que se encuentra en la zona superior derecha en las pantallas de catalogo (Fig. 27) y cuenta

de usuario (Fig. 31). El requisito **UX-02** no ha podido validarse, al no contarse con un grupo de usuarios suficientemente grande para poder sacar conclusiones.

En el apartado de rendimiento, se da por válida **RND-01** aunque en las pruebas realizadas los tiempos oscilaron entre 0,3 y 2,1 segundos, siendo las peticiones con listas “quiero ver” muy grandes (100 o más elementos) las que más tiempo consumen. El requisito **RND-02** también queda validado al ocupar la aplicación menos de 1GB aun incluyendo los datos de cache.

El grupo de seguridad, **SEC-01** y **SEC-03** se cumplen ya que se ha realizado el diseño e implementación que requerían. **SEC-02** y **SEC-04** se consideran satisfechos, ya que no se han encontrado formas de obtener datos de un usuario al azar y se han diseñado e implementado medidas para cumplir la legislación actual.

En referencia al grupo de confiabilidad, ha podido validarse **RLB-02**, ya que se ha tratado de controlar todos los posibles errores mostrándose una respuesta controlada y en las pruebas funcionales no se han detectado errores que bloquearan la aplicación. Los requisitos **RLB-01** y **RLB-03** son válidos por defecto, pudiendo invalidarse en caso de darse sus condiciones en el futuro.

El requisito **PTB-01** también se considera validado, al ser la aplicación compatible con Android 8.0.

Los requisitos no funcionales de soporte (**SUP-01** a **SUP-03**) requieren un uso productivo continuado para poder validarse, por lo que quedarían pendientes de revisarse en el futuro.

A modo de resumen, en Tabla 21 se listan todos los requisitos funcionales y no funcionales y su estado de cumplimiento.

Tabla 21 - Cumplimiento de los requisitos

Requisito	Cumplido			
	Si	Parcialmente	No	No valorable
CU-01	X			
CU-02	X			
CU-03	X			
CU-04	X			
CU-05	X			
CU-06	X			
CU-07	X			
CU-08	X			
CU-09	X			
CU-10	X			
CU-11	X			
CU-12	X			
UX-01		X		
UX-02				**X**
UX-03	X			
RND-01	X			
RND-02	X			
SEC-01	X			
SEC-02	X			

SEC-03	X			
SEC-04	X			
RLB-01	X			
RLB-02	X			
RLB-02	X			
PTB-01	X			
SUP-01				X
SUP-02				X
SUP-03				X

7. Conclusiones

Con los resultados obtenidos, el autor considera que los objetivos principales detallados en la introducción se han cumplido con éxito, si bien hay ciertas mejoras que serían muy interesantes al futuro.

El principal objetivo era desarrollar una aplicación Android que tras indicarle qué se quiere ver, nos permita conocer qué servicio nos es más rentable en función de cómo le demos valor, pudiendo elegir distintas opciones. Es innegable que la versión final cumple este cometido y el autor y su entorno cercano han utilizado ya esta función para evaluar si deben seguir manteniendo todas sus suscripciones o alguna no está siendo realmente rentable o útil.

Algo inesperado fue el tamaño final del *backend*, requiriéndose bastante lógica y tiempo de codificación para el funcionamiento completo de la versión definitiva de la aplicación. Sin embargo, no se considera que el planteamiento fuera errado, ya que, de no invertir tiempo en el *backend*, la codificación de Recomendador de servicios habría sido más costosa temporalmente de lo que se requirió para el *backend*.

Respecto al aspecto final, se han echado en falta ciertos conocimientos de diseño de interfaces, ya que esta, aunque funcional, podría ser estéticamente mejor (combinación de colores, proporciones, …).

A nivel personal, el autor esperaba ganar más conocimiento en el desarrollo en Android. Todo el proceso de desarrollo en Android, el uso de las herramientas de desarrollo que habían evolucionado desde el último uso por parte del autor en el año 2016 han sin duda reforzado todos los conocimientos del autor sobre el desarrollo móvil. Además, con el añadido de haber tenido que desarrollar el *backend* y haber tenido que valorar dónde invertir el tiempo y cómo gestionarlo han refrescado y mejorado los conocimientos del usuario en infraestructura, redes y gestión.

Además de todos los desarrollos a futuro, algo que no se ha llegado a incluir y que al autor le habría gustado explorar es la publicación en la Google Play Store. La cuenta de desarrollo se creó, se configuró parcialmente el perfil de desarrollador y se dio de alta la aplicación, pero no se llegaron a completar las configuraciones necesarias para poder publicar una versión de prueba.

A futuro, no se descarta el continuar con el desarrollo e incluso finalizar las configuraciones en la Play Store y publicar una versión beta para valorar el tipo de recepción en un grupo de gente más grande que el que rellenó la encuesta de mercado, contando además los usuarios de prueba con un producto probable.

Relación del trabajo desarrollado con los estudios cursados

Durante el tiempo en que se cursó el grado, se cursaron diversas asignaturas que tienen relación con este proyecto, destacándose las siguientes:

- Estadística.
- Programación.

- Concurrencia y sistemas distribuidos.
- Deontología y profesionalismo.
- Estructuras de datos y algoritmos.
- Interfaces persona computador.
- Redes de computadores.
- Bases de datos y sistemas de información.
- Calidad de software.
- Desarrollo de software dirigido por modelos.
- Diseño de software.
- Gestión de proyectos
- Ingeniería del software
- Proceso de software
- Tecnología de sistemas de información en la red
- Análisis, validación y depuración de software
- Análisis y especificación de requisitos
- Integración e interoperabilidad
- Mantenimiento y evolución de software
- Soluciones informáticas para dispositivos móviles

8. Trabajos futuros

En el siguiente capitulo se detallan evolutivos y perfectivos sobre el sistema propuesto que no se han podido implementar en el alcance inicial.

Perfectivo – Mejora del *backend* para actualizar los precios de los servicios de manera automática.

En la solución actual, los planes de precio de cada servicio deben introducirse manualmente a la base de datos del *backend*. Este compromiso se alcanzó ya que los planes no suelen actualizarse con frecuencia y es una operación sencilla de realizar de manera manual. Sin embargo, una vez realizados otros puntos más importantes, se podrían destinar recursos a mejorar el *backend* para que obtuviese estos precios automáticamente, ya sea mediante *scraping* web llamando a las páginas de precios de cada servicio o buscando una *API* de terceros que pueda proporcionar esta información.

Evolutivo – Incluir el cálculo de los distintos planes de precios y la compartición de cuentas.

En la solución actual, solo se tiene en cuenta el plan básico de cada servicio. Mejorar la lógica de cálculo del *backend* para poder tener en cuenta todos los planes de precios, dividendo el coste en el caso de los planes anuales para compararlo con los mensuales y también el precio de los servicios que permiten varios dispositivos concurrentes teniendo en cuenta que pueden compartirse las cuentas.

Perfectivo – Añadir un acceso a términos y condiciones en el registro.

En la solución actual, al registrarse únicamente hay un check para aceptar el tratamiento de datos. Modificar este check para incluir un enlace que al pulsar nos lleve a los términos y condiciones detallados, ya sea dentro de la aplicación o en un enlace externo.

Evolutivo – Crear una pantalla de detalle del contenido.

En la lista de catálogos de la solución actual no es posible mostrar todos los datos. Añadir una pantalla adicional que al pulsar un elemento nos permita mostrar los detalles del contenido. Adicionalmente puede combinarse con extensiones al modelo de datos para mostrar nuevos datos que puedan obtenerse de TMDB.

Perfectivo – Añadir logos a los servicios de *streaming*.

En la lista de servicios de la solución actual se muestra un espacio para el logo, que actualmente no muestra nada. Expandir el modelo de datos y permitir la carga del logo del servicio.

Evolutivo – Agregar la funcionalidad de amigos.

El modelo existente ya cuenta con varios apartados dedicados a esta funcionalidad. Implementar en la aplicación y el *backend* la funcionalidad que permita enviar solicitudes de amistad a otros usuarios y poder aceptar o bloquear las solicitudes. Incorporar funciones sociales para poder ver las listas “quiero ver” de las amistades y

valorar la creación de una opción que dé mayor peso a un servicio en función de las listas “quiero ver” de las amistades del usuario.

Evolutivo – Agregar calificación energética de los servicios.

Para mostrar mayor compromiso con los ODS, incluir en el logo la calificación energética del servicio y así como una opción que permita al usuario dar más peso a aquellos servicios con mejor calificación energética.

9. Referencias

[1] R. León Sanz y R. Galán López, Introducción a la movilidad: 4G/LTE y el desarrollo de aplicaciones con Android, Madrid: Escuela Técnica Superior de Ingenieros Industriales, 2013.

[2] Kantar, «Android vs. iOS – Smartphone OS sales market share evolution,» Diciembre 2021. [En línea]. Available: https://www.kantarworldpanel.com/global/smartphone-os-market-share/. [Último acceso: Mayo 2022].

[3] M. Baeza Escámez, Análisis comparativo entre Netflix y HBO: cómo usan sus distintas estrategias comerciales, Valencia: Universitat Politècnica de València, 2021.

[4] A. A. Andrade Muñoz, Análisis sobre el uso de la nostalgia en plataformas streaming (Netflix), Valencia: Universitat Politècnica de València, 2021.

[5] I. Bespal, Gestor de películas, Valencia: Universitat Politècnica de València, 2019.

[6] The Movie Database, «Movie Discover,» 2 Enero 2021. [En línea]. Available: https://developers.themoviedb.org/3/discover/movie-discover. [Último acceso: Junio 2022].

[7] P. Sornosa Pérez, Base de datos cinematográfica con sistema de recomendaciones por perfiles de usuarios, Valencia: Universitat Politècnica de València, 2021.

[8] M. Terol Lloret, CineMapp: Una red social de cine para dispositivos móviles, Valencia: Universitat Politècnica de València, 2019.

[9] Gobierno de España, «Boletín Oficial del Estado,» 6 Diciembre 2018. [En línea]. Available: https://www.boe.es/boe/dias/2018/12/06/pdfs/BOE-A-2018-16673.pdf. [Último acceso: 12 Junio 2022].

[10] Netflix, «Retiring the Netflix Public API (via Wayback Machine),» 14 Junio 2014. [En línea]. Available: http://developer.netflix.com/blog/read/Retiring_the_Netflix_Public_API. [Último acceso: Junio 2022].

[11] Flixed, «Flixed Streaming Metadata API,» 2021. [En línea]. Available: https://flixed.io/. [Último acceso: Junio 2022].

[12] RapidApi, «RapidApi,» 2022. [En línea]. Available: https://rapidapi.com/es/hub. [Último acceso: Junio 2022].

[13] Google LLC, «Core app quality,» 2022 Marzo 2022. [En línea]. Available: https://developer.android.com/docs/quality-guidelines/core-app-quality. [Último acceso: 24 Junio 2022].

[14] S. López, «La fatiga decisoria: cómo afrontar el desgaste psicológico invisible que nos hace infelices,» 17 Febrero 2021. [En línea]. Available: https://smoda.elpais.com/belleza/la-fatiga-decisional-como-afrontar-el-desgaste-psicologico-invisible-que-nos-estresa-y-nos-hace-infelices/. [Último acceso: 8 Mayo 2022].

[15] The Carbon Trust, «Carbon impact of video streaming,» Junio 2021. [En línea]. Available: https://prod-drupal-files.storage.googleapis.com/documents/resource/public/Carbon-impact-of-video-streaming.pdf. [Último acceso: 28 Junio 2022].

[16] Greenpeace, «Clicking Clean: Who is winning the race to build a green Internet,» Enero 2017. [En línea]. Available: http://www.clickclean.org/downloads/ClickClean2016%20HiRes.pdf. [Último acceso: 28 Junio 2022].

[17] M. T. Goodrich y R. Tamassia, Data structures and algorithms in Java (5th. ed.), Hoboken: John Wiley & Sons, 2011.

[18] C. Lasa Gómez, A. Álvarez García y R. de las Heras del Dedo, Métodos ágiles: Scrum, Kanban, Lean., Madrid: Anaya Multimedia., 2017.

[19] H. Podeswa, UML, Madrid: Anaya Multimedia, 2010.

[20] P. E. Fernández Casado, UX design: hazlo fácil pensando en el usuario, Madrid: Ra-ma, 2021.

[21] P. Stevens y R. Pooley, Utilización de UML en ingeniería del software con objetos y componentes (2a ed.), Madrid: Pearson Educación, 2007.

[22] B. Momjian, PostgreSQL: introduction and concepts, Boston: Addison-Wesley, 2001.

[23] S. McCracken, Android: curso de desarrollo de aplicaciones, Barcelona: Inforbook's, 2012.

[24] Google LLC, «Guide to app architecture,» 28 Abril 2022. [En línea]. Available: https://developer.android.com/topic/architecture. [Último acceso: 21 Junio 2022].

[25] Google, LLC, «Documentation for app developers,» 2022. [En línea]. Available: https://developer.android.com/docs. [Último acceso: 18 Junio 2022].

[26] Google, LLC, «Android platform,» 2022. [En línea]. Available: https://developer.android.com/about. [Último acceso: 2 Junio 2022].

[27] JustWatch GmbH, «JustWatch,» 2022. [En línea]. Available: https://www.justwatch.com/. [Último acceso: Junio 2022].

[28] Spliiit SAS, «Spliiit,» 2022. [En línea]. Available: https://www.spliiit.com/es. [Último acceso: Junio 2022].

[29] Together Price Limited, «Together Price,» 2022. [En línea]. Available: https://www.togetherprice.com/es/. [Último acceso: Junio 2022].

10. Glosario

Algoritmo: en informática, se refiere al conjunto de operaciones ordenas que permiten resolver o dar solución un problema.

API: Acrónimo (ingles) de Application Programming Interface. Es una interfaz intermediaria que permite la comunicación estructurada entre dos programas.

Backend: parte de una la aplicación a la que el usuario no accede de manera directa. Es la parte contraria al Frontend. Habitualmente contiene lógica de negocio y capacidad de manipular y almacenar la información de la aplicación.

Compilar: proceso por el cual se traduce el código escrito por el programador en su lenguaje de programación de alto nivel a un lenguaje máquina de bajo nivel que permite que sea interpretado y procesado por la computadora.

Desplegar: proceso por el cual se disponibilidad el software para su uso. Consta de varias fases en función de si ya existe una instalación (actualización) o no (despliegue desde cero).

Emular: hacer funcionar un programa o sistema de la misma manera que otro, simulando todas sus funciones de manera equivalente.

Entorno de desarrollo: hace referencia al programa o conjunto de programas que el desarrollador utiliza a la hora de implementar su código. Cuenta con diversas herramientas para facilitar el trabajo de desarrollo.

Fatiga decisoria: "*(…) cansancio mental que experimentamos cuando ya estamos sobrecargados tras tomar muchas decisiones, lo que provoca que éstas ya no sean tan buenas*" [13].

Frontend: parte de una la aplicación a la que el usuario accede de manera directa y utiliza para interactuar con esta. Es la parte contraria al *Backend*. Habitualmente contiene las interfaces gráficas, pero también puede contener parte de la lógica de negocio e incluso acceso directo a los datos.

GDPR: acrónimo (inglés) de General Data Protection Regulation. Véase también RGPD.

Infraestructura: hace referencia al conjunto de hardware, redes y herramientas software sobre las que se despliega una solución informática. En el caso concreto de StreamBudget, hace referencia a las máquinas virtuales que actúan como servidor, la base de datos y la red sobre la que esta desplegado el *backend.*

Interfaz: parte del software que permite comunicar dos sistemas, componentes o programas distintos. Puede referenciar tanto a la parte visual del software con la que el usuario interactúa (Interfaz de Usuario o IU) o a una parte programática que permita la comunicación entre dos programas distintos (intercomunicación o integración).

Nivel (ref. lenguaje de programación): nivel de abstracción que tiene un lenguaje. Cuando se habla de un lenguaje de alto nivel, se hace referencia a un lenguaje directamente entendible por una persona, cuanto más alto sea, menos abstracto y más cercano al lenguaje natural humano. Por el contrario, los niveles de bajo nivel o lenguaje máquina, tienen un gran nivel de abstracción, son complejos de entender a simple vista y pueden llegar a ser completamente incompresibles para una persona, solo puedo ser procesados por una computadora.

RGPD: acrónimo de Reglamento General de Protección de Datos. Véase también GDPR.

Piratear: obtención de una copia ilegal de productos protegidos con derecho de autor (software o material audiovisual).

QA: abreviatura de *quality assurance*. En español, aseguración o control de calidad. Hace referencia al conjunto de procesos y tareas planificadas para que los requisitos de calidad del software satisfechos. Incluyen desde pruebas de errores (testing), pruebas de usabilidad y claridad de la interfaz, pruebas de rendimiento y estrés, etc.

Scraping web: método o técnica utilizado para extraer información de un sitio web mediante programas de software.

Streaming: tecnología que permite acceder a contenidos multimedia tales como películas o música en cualquier momento y lugar, utilizando un dispositivo móvil o un ordenador personal. El envío de contenido se realiza a través de Internet y no está sujeto a horarios, pudiendo consumir los contenidos bajo demanda.

Método MoSCoW: el método MoSCoW, inventado por Dai Clegg, se utilizada para priorizar características en proyectos con limitaciones temporales. Es especialmente apreciado en las metodologías agiles. Es un acrónimo de sus cuatro categorías Must have, Should have, Could have, Won't have, siendo estas en español Debe, Debería, Podría y No tendrá.

Testing: hace referencia al proceso de pruebas y verificaciones por las que pasa un software para validar que funciona correctamente y libre de errores.

Timeout: tiempo máximo que el sistema solicitante esperara para recibir una respuesta. Pasado este tiempo, se asume que la respuesta no va a llegar y se trata como un error.

TMDB: abreviatura de The Movie Database.

11. Anexos

Anexo I – Objetivos de desarrollo sostenible

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

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

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

Debido al objetivo y funcionalidad de StreamBudget, no es posible relacionarlo con la mayoría de ODS, marcados como no procede en el apartado anterior, ya que tienen un alcance muy ambicioso en campos muy distintos al del entretenimiento en *streaming*. Sin embargo, si es posible encontrar una relación con los ODS 12 y 13, además de una relación muy ligera con ODS 4.

En la encuesta de mercado realizada, se preguntó sobre el comportamiento al entrar al servicio de *streaming* y solo el 31% accedía directamente a ver el contenido, el 69% restante da un “vistazo” por el servicio antes de iniciar la reproducción haya o no decido ya lo que va a reproducir (*Véase* Fig. 32)

Fig. 32 – Gráfico del comportamiento al entrar al servicio de streaming

Según los datos del estudio Carbon impact of video streaming [15], que indica que el 50% de la energía consumida es por parte del dispositivo que utiliza el usuario, siendo la televisión el dispositivo que más consume con 58g CO2/h. Si asumimos que cada usuario de media "pierde" 15 minutos navegando sin rumbo por el servicio de *streaming* y pensando en un servicio con unos tres millones de suscripciones activas, esta navegación sin rumbo supondría un consumo de 450,2 toneladas de CO2.

$$3.000.000\ suscripciones \times \frac{69}{100} \times 15\ min \times 58g\frac{CO2}{h} \times \frac{1}{4}h = 450.225.000g$$

Aun reduciendo a solo el 40% que entra sin saber que ver, seguirían siendo 261 toneladas de CO2. No todo el mundo consume desde el televisor, por lo que el valor real sería menor, pero en cualquier caso es una cantidad que considerar.

StreamBudget introduce de manera indirecta un cambio en la manera de consumir los servicios de *streaming*. Al tener el usuario que elegir los contenidos que quiere ver y, si acepta la recomendación, contratar el servicio donde más contenido tiene para ver, le da al usuario un objetivo de visualización atacando a ese 40% de indecisos y en general al tiempo muerto dentro del servicio de *streaming*, lo cual estaría relacionado con ODS 12 ya que fomentaría un uso más responsable de la energía y ODS 13 al buscar una reducción del CO2 emitido a la atmosfera.

Además, como una funcionalidad futura, podría incluirse la calificación energética de cada servicio, basándose en los datos de #ClickClean Who is winning the race to build a green Internet [16]. Esta calificación podría mostrarse al lado del nombre del servicio o incrustado en su logotipo. Para completar, se podría añadir una nueva opción con un check para dar peso extra a los servicios con mejor calificación energética.

Con respecto a ODS 4, hay un género dentro del contenido relacionado con la educación, los documentales. Dadas las funciones de StreamBudget, podría utilizarse

por familias con presupuestos más ajustados para buscar el servicio más rentable que más películas o series documentales y de contenido educativo les ofrezca. Esto reduciría ligeramente la brecha entre rentas y permitiría a esas familias acceder a este contenido educativo que teniendo que hacer la búsqueda o el cálculo de manera manual podrían haber perdido.

Anexo II – Encuesta de mercado

A continuación, se presentarán las preguntas que contenía la encuesta realizada en Google Forms (Véanse Fig. 37 a Fig. 40). Si bien se hizo una versión en español y otra en inglés, se incluyen las capturas del idioma español únicamente. También se presentan los datos de la distribución demográfica de la muestra desde Fig. 33 a Fig. 36.

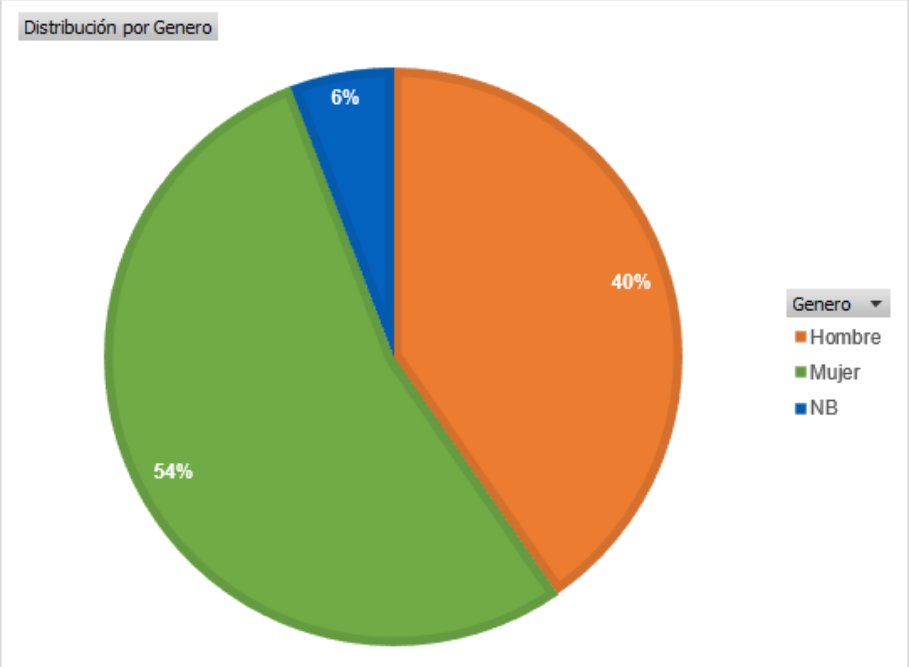

Fig. 33 – Distribución demográfica por genero

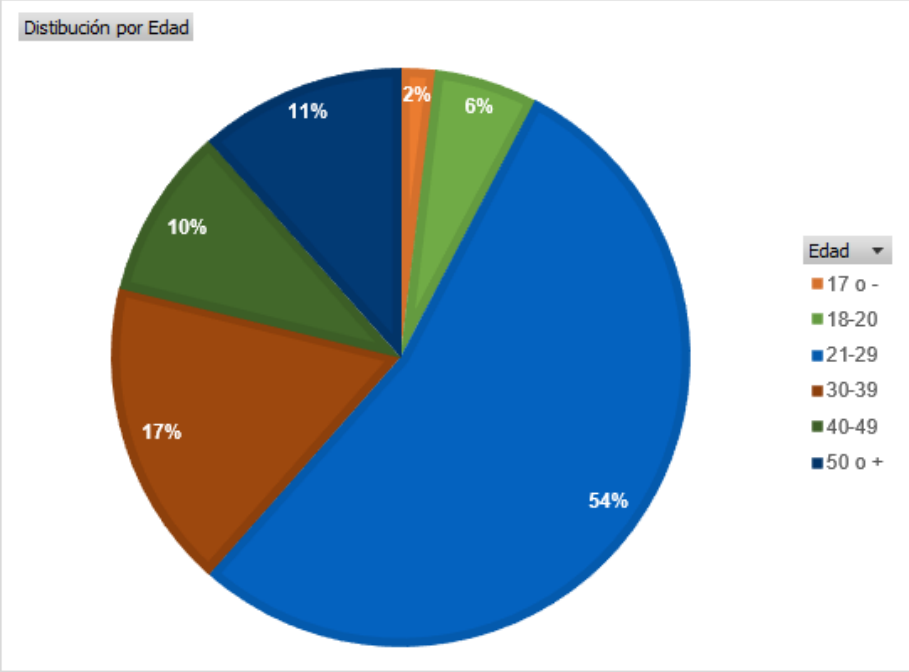

Fig. 34 – Distribución demográfica por edad

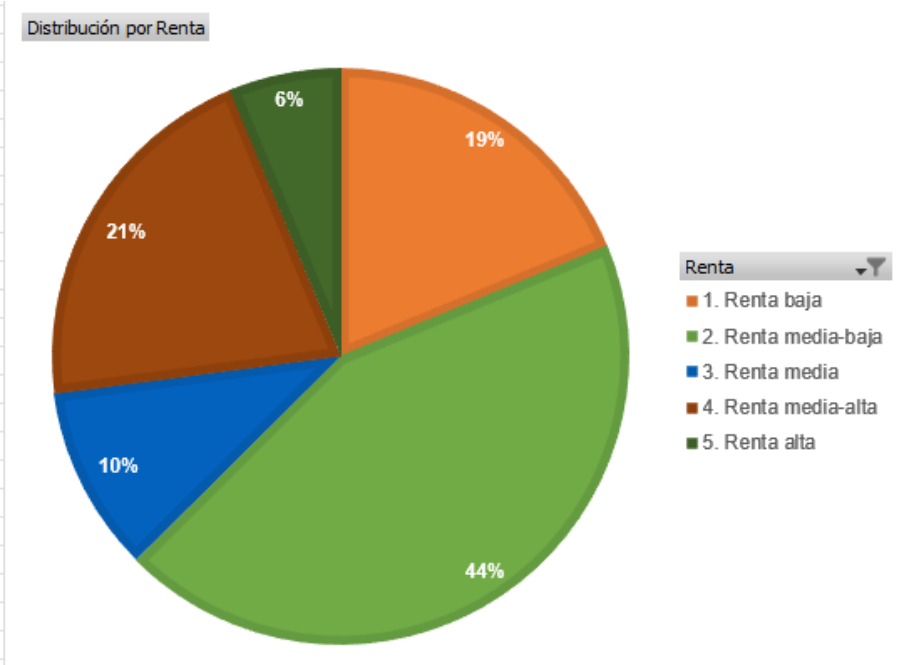

Fig. 35 – Distribución demográfica por renta

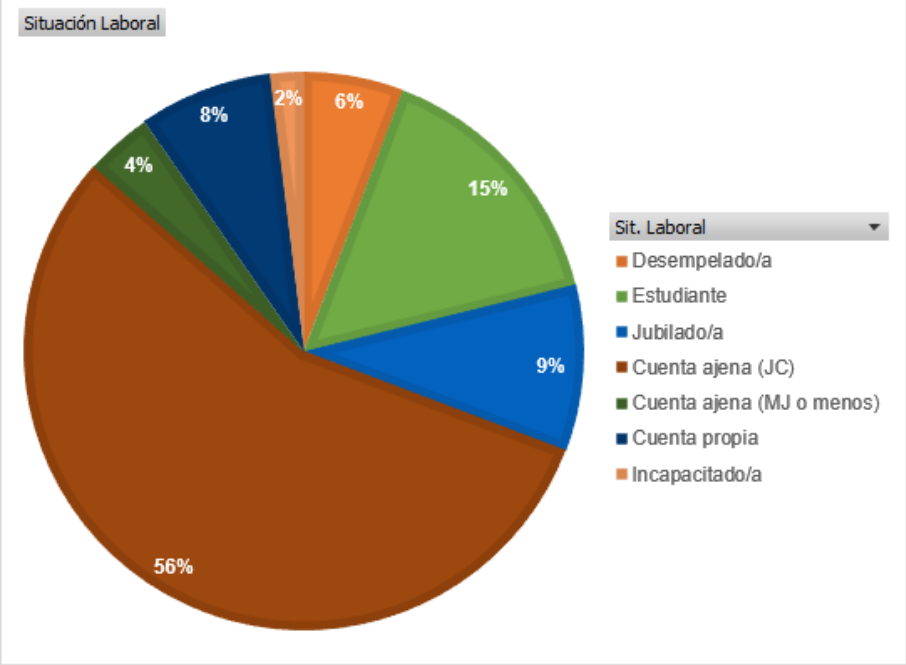

Fig. 36 – Distribución demográfica por situación laboral

Se lanzó una encuesta en español y otra en inglés y posteriormente se agregaron los resultados de ambas. De manera agregada se recopilaron 52 resultados, con dicho valor de muestra y un nivel de confianza del 95%, obtenemos un 13,59% de margen de error.

Encuesta StreamBudget

*Obligatorio

1. ¿Eres...? *

Marca solo un óvalo.

- ◯ Hombre
- ◯ Mujer
- ◯ Otro: _____

2. ¿Cuál es tu edad? *

Marca solo un óvalo.

- ◯ 17 años o menos
- ◯ 18 años a 20 años
- ◯ 21 años a 29 años
- ◯ 30 años a 39 años
- ◯ 40 años a 49 años
- ◯ 50 años o más

3. ¿Cuál de las siguientes categorías describe mejor tu situación laboral? *

Marca solo un óvalo.

- ◯ Estudiante
- ◯ Empleado/a por cuenta ajena, media jornada o menos
- ◯ Empleado/a por cuenta ajena, jornada completa
- ◯ Empleado/a por cuenta propia
- ◯ Desempelado/a
- ◯ Jubilado/a
- ◯ Incapacitado/a para trabajar

4. Indica el nivel de ingresos de tu unidad familiar *

Marca solo un óvalo.

- ◯ Menos de 20.000€
- ◯ 20.000€ - 39.999€
- ◯ 40.000€ - 60.000€
- ◯ Más de 60.000€
- ◯ Prefiero no contestar

Fig. 37 – Imagen encuesta de mercado parte 1

5. Indica las opciones que se identifiquen con su tipo de acceso al contenido de la suscripción *

Considera que eres propietario si eres la persona que paga el recibo

Marca solo un óvalo por fila.

	Soy propietario de la suscripción	Pertenezco a un grupo y pago una parte	Pertenezco a un grupo sin pagar nada	No utilizo este servicio
Netflix	◯	◯	◯	◯
Prime Video	◯	◯	◯	◯
Disney+	◯	◯	◯	◯
HBO Max	◯	◯	◯	◯
Apple TV	◯	◯	◯	◯
Filmin	◯	◯	◯	◯
Rakuten TV	◯	◯	◯	◯

6. Independientemente de las respuestas anteriores, pensando en sus preferencias a la hora de contratar una suscripción... *

Selecciona el que más se aproxime

Marca solo un óvalo.

- ◯ Prefiero la suscripción básica y usarlo individualmente
- ◯ Prefiero la suscripción más completa y usarlo individualmente o compartirla sin pedir la parte proporcional
- ◯ Prefiero la suscripción mas completa y compartirlo para distribuir el coste
- ◯ Prefiero no contratarlo directamente y que alguien lo comparta conmigo

7. Indique el consumo aproximado de cada uno de los servicios *

Selecciona el que más se aproxime en cada caso

Marca solo un óvalo por fila.

	Todos los días	Varios días por semana	Algunos dias al mes	No utilizo este servicio
Netflix	◯	◯	◯	◯
Prime Video	◯	◯	◯	◯
Disney+	◯	◯	◯	◯
HBO Max	◯	◯	◯	◯
Apple TV	◯	◯	◯	◯
Filmin	◯	◯	◯	◯
Rakuten TV	◯	◯	◯	◯

Fig. 38 – Imagen encuesta de mercado parte 2

8. A la hora de ver una película o serie... *

Selecciona el que más se aproxime

Marca solo un óvalo.

- ◯ Se lo que quiero ver y accedo directamente.
- ◯ Aunque se lo que quiero ver, doy un vistazo a otras cosas antes.
- ◯ Normalmente doy un vistazo hasta que algo me llame para verlo.

Pensando en una aplicación móvil donde pudieras indicar películas y series que quieres ver y esta te recomendara el servicio más rentable en función de tus preferencias...

9. Indica cuanto te interesaría este tipo de aplicación *

Marca solo un óvalo.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Mucho

10. A la hora de visualizar el resultado, ¿Cuál seria tu preferencia? *

Marca solo un óvalo.

- ◯ Ver el servicio recomendado según mis preferencias
- ◯ Ver una lista de todos los servicios, ordenadas porcentualmente según la coincidencia con mis preferencias
- ◯ No tengo preferencia con la visualización del resultado

11. Indica cual de las siguientes opciones se aproxima más a tu modo de dar valor a un servicio *

Marca solo un óvalo.

- ◯ Valoraría equitativamente las películas y las series. Es decir, dos películas y una serie tendría un valor de 3.
- ◯ Valoraría en función de los minutos. Aunque el servicio tenga menos cosas, preferiría el que mayor numero de minutos de contenido.
- ◯ Valoraría los episodios de las series y las películas. Es decir, dos películas y una serie de diez episodios tendría un valor de 12.
- ◯ Ninguna de las anteriores.

Fig. 39 – Imagen encuesta de mercado parte 3

12. Además de la función principal, indica tu grado de interés con cada una de las siguientes funciones adicionales *

Donde 1 significa nada interesante y 5 muy interesante

Marca solo un óvalo por fila.

	1	2	3	4	5
Poder guardar mi perfil y ver actualizaciones mes a mes del mejor servicio	◯	◯	◯	◯	◯
Características de red social, poder agregar amistades, ver que servicios y series ven mis amigos.	◯	◯	◯	◯	◯
Ofertas especiales para los servicios de streaming	◯	◯	◯	◯	◯
Calendarios del contenido que se agrega o elimina de los servicios	◯	◯	◯	◯	◯
Buscador de grupos para poder compartir la suscripción	◯	◯	◯	◯	◯

13. ¿Hay alguna funcionalidad no indicada anteriormente que te gustaría tener en este tipo de aplicación?

14. A continuación puede añadir cualquier comentario que considere relevante

Fig. 40 – Imagen encuesta de mercado parte 4