

Universidad De Buenos Aires

Facultad de Ingeniería

Departamento de Computación

Trabajo Profesional - Ingeniería en Informática

Stream Club

Aplicación de Seguimiento y Gestión del Contenido Visto en Plataformas de Streaming

Alumno	Padrón	Mail
LARREA BUENDÍA, Hugo Marcelo	102140	hlarrea@fi.uba.ar
BALESTIERI, Juan Diego	101601	jbalestieri@fi.uba.ar
CAPELLI, Tiago Sebastian	98316	scapelli@fi.uba.ar

Tutor: Prof. Dr. Diego Corsi

2024

Índice

1	Introducción	5
2	Motivación	6
2.1	Objetivo	7
3	Estado del Arte	8
3.1	<i>My Anime List</i> (M.A.L.)	8
3.2	<i>Letterboxd</i>	9
3.3	<i>Just Watch</i>	11
4	Solución Implementada	11
4.1	Alcance	12
4.2	Metodología de Trabajo	14
4.3	Herramientas y Tecnologías	16
4.4	Arquitectura	18
4.4.1	Aplicación	20
4.4.1.1	Gestión de cuenta	21
4.4.1.2	Exploración de películas, series y artistas	22
4.4.1.3	Recomendaciones de películas, series y servicios de <i>streaming</i>	27
4.4.1.4	Red Social	28
4.4.2	Backend	30
4.4.2.1	Process API	31
4.4.2.2	Content API	32
4.4.2.3	User API	33
4.4.2.4	Recommendations API	34
4.4.3	Cálculo de recomendaciones	35
4.4.3.1	Métricas	37
4.5	Desarrollo	38
4.5.1	Especialista UX	39
4.5.2	Especialista <i>Machine Learning</i>	39

4.6	Planificación y Resultado	40
4.7	Proceso de Despliegue	44
5	Dificultades y Lecciones Aprendidas	45
5.1	Dificultades	45
5.1.1	<i>Expo Router</i>	45
5.1.2	<i>Redux</i>	45
5.1.3	<i>TMDB y Just Watch</i>	46
5.1.4	<i>Render y AWS</i>	47
5.2	Riesgos Materializados	48
5.2.1	Licencia de <i>Jira</i>	48
5.2.2	Ausencias	48
5.2.3	Base de datos gratuita de <i>Render</i>	49
5.3	Lecciones aprendidas	49
5.3.1	Organización y distribución de tareas	49
5.3.2	Exceso de funcionalidades	50
5.3.3	Prematura optimización	50
6	Futuras Líneas de Trabajo	51
7	Conclusiones	53
8	Referencias	55
9	Anexos	59
9.1	Encuesta	59
9.2	Repositorios	61

Agradecimientos

En este trabajo, que representa el final de una larga etapa para nosotros, queremos dar las gracias a la Universidad de Buenos Aires por ser nuestro segundo hogar por tantos años. A nuestros docentes por habernos guiado en este camino. A nuestros compañeros por caminar junto a nosotros esta senda y, en especial, a nuestros amigos y familia por darnos su apoyo incondicional.

Resumen

El presente trabajo se enfoca en el desarrollo de *Stream Club*, una aplicación móvil que integra múltiples plataformas de *streaming* con el objetivo de unificar la experiencia de búsqueda y visualización de películas y series. Además, permite llevar un registro del contenido visto en otros medios como el cine o la televisión, y también agregar amigos, formar grupos, enviar recomendaciones y mucho más.

La idea surge a partir de la proliferación de servicios de *streaming*. Con tantas opciones, los usuarios pueden confundirse sobre qué plataformas están utilizando, o en cuáles está disponible lo que desean ver. Además, las recomendaciones recibidas pueden no ser óptimas, ya que solo consideran lo visualizado en una plataforma.

Existen varias alternativas en el mercado, pero suelen enfocarse en un aspecto específico. *Stream Club*, en cambio, cubre un gran abanico de necesidades en una sola aplicación.

Para el desarrollo se siguió una metodología ágil, lo que permitió iteraciones rápidas y una retroalimentación constante del tutor y de diversos colaboradores, facilitando la mejora continua y la adaptación del proyecto en función de las necesidades emergentes.

Luego de un arduo año de trabajo, se ha creado una herramienta integral, funcional y, no menos importante, divertida. Si bien las limitaciones propias de un trabajo práctico implicaron el recorte de ciertas funcionalidades, *Stream Club* tiene mucho potencial para crecer.

1. Introducción

El presente informe tiene como objetivo documentar el desarrollo del *Trabajo Profesional de Ingeniería Informática* de los alumnos *Juan Diego Balestieri, Tiago Sebastián Capelli y Hugo Marcelo Larrea Buendia*. Este consistió en el desarrollo de un sistema unificador de plataformas de *streaming*^[1], con el objetivo de proveer una única aplicación que permita consolidar el contenido de las distintas plataformas suscritas por el usuario.

Debido a la creciente cantidad de servicios de *streaming*, la búsqueda de contenido se volvió compleja y lenta, ya que implica recorrer individualmente las distintas aplicaciones del usuario. Con este trabajo, se buscó aportar una solución que permita acceder al contenido disponible desde un único lugar.

A la vez, las recomendaciones ofrecidas por los servicios de *streaming* están basadas únicamente en lo visto en una sola plataforma y, por lo tanto, son incompletas. Debido a esto, se planteó desarrollar un algoritmo capaz de recomendar contenido basado en todo aquello que fue consumido por el usuario, independientemente de la plataforma utilizada. Ofreciendo así, recomendaciones más completas y precisas.

Hoy en día, existen aplicaciones que permiten hacer seguimiento de películas y series vistas como: *My Anime List*^[2], *Letterboxd*^[3], o *Just Watch*^[4], algunas de las cuales incluso cuentan con recomendaciones personalizadas. Sin embargo, ninguna ofrece una experiencia completa, ya que no incorporan funcionalidades esenciales y son tediosas de utilizar. Por ello, *Stream Club* incorpora también aspectos de red social y juegos, con el objetivo de promover la interacción entre los usuarios, ya que consumir contenido suele ser una experiencia también social. Así mismo, se ha diseñado un sistema de recompensas por el uso de la aplicación, lo que permite que el usuario disfrute finalmente de una experiencia integral y satisfactoria en el consumo de su contenido diario.

2. Motivación

Con el objetivo de confirmar la existencia de una problemática en el consumo de contenido a través de plataformas de *streaming*, el equipo realizó una encuesta a más de 350 personas (ver Encuesta) para conocer sus hábitos de uso y las dificultades que enfrentan diariamente.

A partir de esta encuesta, se observó que el 93 % de los encuestados tiene al menos una plataforma de *streaming* y el 65 % cuenta con más de una (pregunta 1). Esto confirma la popularidad de estos servicios en la actualidad. Además, se evidencia una necesidad de integración ya que la mayoría tiene suscripciones a más de una plataforma. Buscar contenido, por ejemplo, no debería ser una tarea que requiera una gran cantidad de pasos; sin embargo hoy en día resulta tediosa y compleja.

Asimismo, la falta de integración entre las distintas plataformas lleva a los usuarios a concentrar su uso en una sola de ellas. En la encuesta, se observó que el 89 % de los encuestados afirma utilizar una plataforma con mayor frecuencia que las demás, y un 12 % utiliza solo una de las múltiples que tienen contratadas (pregunta 2).

Casi todas las plataformas ofrecen distintos tipos de recomendaciones a sus usuarios. Sin embargo, la encuesta refleja que el 70 % de los encuestados no está completamente satisfecho con las recomendaciones recibidas. Además, solo el 13,8 % afirma que uno de los principales motivos para consumir contenido son las recomendaciones provistas (pregunta 3).

El hecho de que las recomendaciones solo consideren el contenido visto en cada plataforma de forma aislada constituye una limitación a la hora de realizar sugerencias de contenido.

La encuesta también destaca el impacto del círculo social en el consumo de contenido: uno de los principales motivos por los cuales un usuario decide ver contenido es a partir de recomendaciones de familiares o amigos (pregunta 4). Sin embargo, las plataformas

de *streaming* basan sus algoritmos de recomendación solo en los intereses personales, sin considerar el aspecto social. Además, más del 60 % de los encuestados consume contenido frecuentemente con familiares o amigos. Esto indica una necesidad del usuario de recibir recomendaciones para un grupo de personas y no únicamente para sí mismos (pregunta 5).

Aunque ya existen aplicaciones que integran distintos servicios de *streaming*, los encuestados que las han utilizado mencionaron que el proceso de carga de datos suele ser poco práctico y no amigable. Además, al no ofrecer incentivos o recompensas por utilizar la aplicación, esta tarea se percibe como tediosa y monótona. Estos inconvenientes impiden que dichas aplicaciones se popularicen de manera masiva y podrían explicar por qué muchos usuarios abandonan su uso (pregunta 6). Asimismo, la falta de interacción social en estas aplicaciones ofrece pocos incentivos para que los usuarios las recomiendan.

2.1. Objetivo

El objetivo de este trabajo fue construir una aplicación de integración de plataformas de *streaming amigable y superadora* que le permite al usuario:

- *Conocer el contenido* de cada plataforma y particularmente de aquellas que tiene contratadas.
- *Redirigir al contenido* de las distintas plataformas de manera sencilla desde la aplicación.
- *Hacer un seguimiento del contenido* visto o pendiente de ver.
- *Generar informes y estadísticas de uso*, enfocados en el contenido visualizado y consumo de cada plataforma.
- Contar con un perfil, una lista de amigos y demás aspectos orientados a *redes sociales*.
- Disponer de un *sistema de recomendaciones* basado en todo el contenido consumido, independientemente de la plataforma.

- Disponer de un *sistema de recompensas* que promueve el consumo de contenido.
- Disponer de un *sistema de desafíos y juegos* que incentivan el uso y carga de datos.

3. Estado del Arte

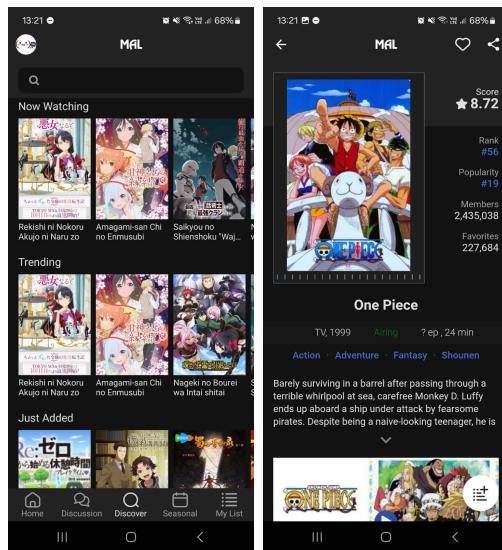
En el mercado existen varias aplicaciones que permiten realizar el seguimiento del contenido visto. Algunas de ellas incluso permiten visualizar en qué plataformas se encuentra disponible dicho contenido. Entre estas alternativas, las más populares son *My Anime List*^[2], *Letterboxd*^[3] y *Just Watch*^[4]. Debido a su relevancia, se analizaron sus funcionalidades para conocer el estado del arte en cuanto a este tipo de aplicaciones.

3.1. *My Anime List* (M.A.L.)

También conocida como M.A.L. debido a sus siglas. Es una de las plataformas más populares para los fanáticos del Animé y cuenta con una aplicación móvil (Figura 1). Desde su fundación en 2004, se ha convertido en el sitio de referencia dentro de este nicho. Sus principales características son:

- Permite hacer seguimiento del contenido siempre y cuando se trate de Animés, lo cual la transforma en una aplicación poco atractiva para un público interesado en otro tipo de contenido.
- Permite ver que contenido está en emisión.
- No muestra en qué servicios se encuentra disponible determinado contenido.
- Aunque permite buscar contenido, no ofrece opciones de filtrado.
- Permite participar en foros de discusión con otros usuarios.
- Maneja la privacidad del contenido visto.

- Permite tener amigos; sin embargo, no es posible buscar usuarios, lo que hace que agregarlos sea una tarea tediosa y compleja.
- Muestra noticias sobre el mundo del Animé, aunque no están relacionadas con las preferencias del usuario.
- No realiza recomendaciones basadas en el contenido visto por el usuario.
- No cuenta con juegos.
- Permite escribir reseñas.



(a) Búsqueda (b) Detalle de Serie

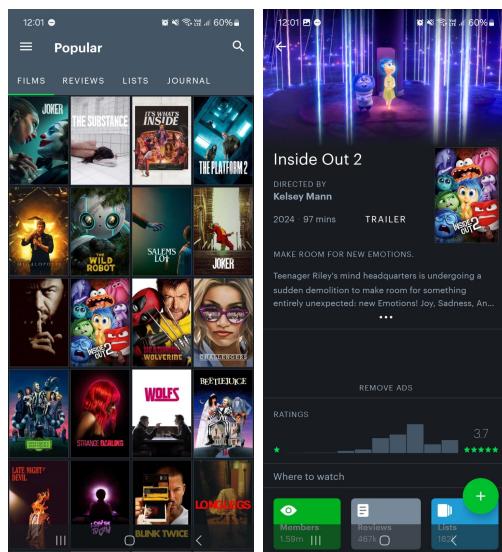
Figura 1

App de *My Anime List*

3.2. *Letterboxd*

Letterboxd es una plataforma centrada en el cine que se ha convertido en una herramienta muy usada por cinéfilos que buscan organizar su historial de visionado. También cuenta con una aplicación móvil (Figura 2). Sus principales características son:

- Permite hacer seguimiento de películas.
- No incluye series en su catálogo.
- Ofrece, por un costo adicional, la posibilidad de saber en qué plataforma está disponible un contenido determinado.
- Permite escribir reseñas.
- No cuenta con juegos.
- Permite agregar amigos y que contenido han marcado como visto. Sin embargo, esta funcionalidad resulta poco amigable para el usuario.
- Permite crear listas de contenido.



(a) Inicio (b) Detalle Película

Figura 2

Aplicación de *Letterboxd*

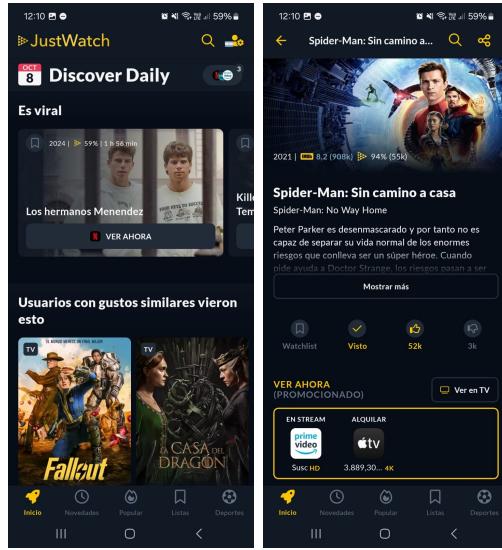
3.3. *Just Watch*

Just Watch es una plataforma de búsqueda y recomendación de contenido diseñada para ayudar a los usuarios a encontrar fácilmente dónde ver películas y series en servicios de *streaming*. Además, ofrece recomendaciones personalizadas basadas en las preferencias del usuario. Cuenta con una aplicación móvil (Figura 3) y sus principales características son:

- Permite gestionar los servicios de *streaming*.
- Ofrece la posibilidad de saber en qué plataforma está disponible un contenido determinado.
- Cuenta con un sistema de calificación de contenido mediante 'me gusta' o 'no me gusta'.
- No permite escribir reseñas.
- No permite la interacción entre usuarios.
- No cuenta con juegos.
- Permite agregar elementos a una lista de contenido (*Watchlist*).
- Aunque permite buscar contenido, no ofrece opciones de filtrado.

4. Solución Implementada

Para cumplir con los objetivos mencionados en la sección 2.1 se desarrolló una aplicación móvil. Se optó por esta alternativa, en lugar de una página web, por tratarse de una opción cómoda para la mayoría de los usuarios debido a que al estar viendo algún contenido



(a) Inicio

(b) Detalle Película

Figura 3

Aplicación de *Just Watch*

audiovisual, es muy probable que tengan su dispositivo móvil cerca para poder marcarlo como visto dentro de la aplicación.

Esta aplicación se comunica con un servidor alojado en la nube que permite a los usuario acceder de forma sencilla a toda su información desde cualquier lugar en cualquier momento. Se utilizó una API externa^[24] capaz de ofrecer información detallada y actualizada de películas, series y artistas. Por último, se realizó un algoritmo de recomendación *batch* capaz de calcular distintos tipos de recomendaciones, algunas de ellas teniendo en cuenta el contenido visto por los usuarios. Todos estos puntos serán explicados con mayor detalle en esta sección.

4.1. Alcance

El alcance definido fue el siguiente:

- *Interacción con contenido:* se permite a los usuarios obtener información de películas,

series y artistas; conocer la disponibilidad del contenido en las diferentes plataformas de *streaming* y ser redirigido a este en alguna de dichas plataformas. Para poder acceder a dicho contenido es posible que el usuario deba tener contratada la plataforma seleccionada (salvo que la misma provea una forma gratuita de visualizarlo).

- *Perfil de usuario:* los usuarios son capaces de crear su perfil utilizando su cuenta de *gmail*, elegir una foto de perfil entre un conjunto de opciones y cambiar su nombre de usuario para ser reconocido más fácilmente dentro de la aplicación.
- *Reseñas:* los usuarios pueden crear, editar y borrar reseñas de películas y series, así como también visualizar aquellas hechas por otros usuarios.
- *Sistema de recompensas:* los usuarios son recompensados con puntos por cada reseña que dejan en la aplicación. Dichos puntos permiten subir de nivel y con esto desbloquear nuevas imágenes para su perfil.
- *Recomendaciones de contenido:* el sistema ofrece a los usuarios recomendaciones de series y películas basadas en su actividad general en la aplicación; de contenido similar a una serie o película; de películas para grupos de amigos basado en la actividad de todos sus miembros y de contenido en base a una selección de hasta 3 elementos visualizados.
- *Red social:* los usuarios pueden agregar amigos a la aplicación y crear grupos con ellos para recibir recomendaciones basadas en el contenido visualizado por ellos o de contenido para ver juntos.
- *Integración con redes sociales:* el sistema permite a los usuario acceder a las redes sociales de un artista desde la aplicación, así como compartir contenido a sus amigos mediante alguna de sus redes sociales.
- *Juegos:* existen trivias sobre distintas películas y series que pueden ser jugadas por los usuarios.

- *Estadísticas de uso:* el sistema recopila información sobre el uso de cada plataforma de *streaming* de los usuarios para permitirles una mayor visibilidad sobre el consumo de cada una de estas. En base a esta información, el sistema recomienda nuevas plataformas a ser contratadas y dar de baja aquellas menos utilizadas.

4.2. Metodología de Trabajo

Para el desarrollo del presente trabajo se utilizó una metodología similar a *Scrum*^[31] debido a su enfoque ágil. Sin embargo, esta fue adaptada a las necesidades presentadas por el equipo de trabajo. Se optó por no implementar ciertos roles de dicha metodología ni tampoco celebrar todas las reuniones que una implementación de libro de la misma implicaría. Durante el desarrollo, se armaron sprints de dos semanas en los cuales se trabajaba sobre ciertas funcionalidades definidas al inicio del mismo.

Para el seguimiento del trabajo el equipo se reunía semanalmente. Estas reuniones podían ser cortas, cuando se trataba de una reunión de medio *sprint*, o largas, cuando era necesario finalizar un *sprint* e iniciar el siguiente. Esto se debe a que las reuniones a mitad de *sprint* cumplían una función similar a las *dailys* dentro de *Scrum*: tener un panorama de la situación de cada miembro del equipo respecto a sus tareas, identificar las dificultades que se hayan presentado y tomar las acciones necesarias al respecto. Las reuniones a final de *sprint* resultaban más largas debido a que eran tanto reuniones de revisión del *sprint* actual como planificación del siguiente.

El equipo, además, tuvo reuniones de retrospectiva en las ocasiones que sus integrantes lo consideraron necesario. Estas resultaron ser una excepción más que una norma dentro de la metodología de trabajo.

Para la organización de las tareas se utilizó un sistema de *tickets* montado sobre la herramienta *Jira*^[13], la cual permite crear tareas y asociarlas a distintas historias. Con el pasar de los sprints, fueron surgiendo *bugs* y deudas técnicas que también fueron agregados a la herramienta. Es por esto que *Jira* se convirtió rápidamente en una herramienta

fundamental para tener una visión del estado y progreso del proyecto durante todas sus etapas.

Para el armado de los *sprints* se seguían los siguientes pasos:

1. Definición de las historias sobre las que se trabajaría durante el *sprint*: para este paso era crucial seguir lo presentado en la propuesta de trabajo, buscando respetar las etapas previamente definidas.
2. Creación de las tareas correspondientes a las historias a realizar: estas normalmente consistían de una parte de *backend* y una o más de *frontend*. También se contó con tareas de diseño donde uno de los integrantes realizaba una propuesta gráfica sobre alguna pantalla a ser desarrollada.
3. Asignación de las tareas: el equipo tomaba la decisión sobre a que miembro le correspondía determinada tarea según sus capacidades, conocimientos y preferencias a la hora de trabajar. Este criterio, que se mantuvo consistente durante todo el desarrollo, permitió una especialización de los integrantes en sus áreas de preferencia y, de ese modo, un aumento de productividad en sus respectivas tareas.
4. Asignación de *bugs* y deudas: durante el desarrollo fueron surgiendo *bugs* y deudas técnicas que fueron cargadas en *Jira*. Según su gravedad y el tiempo disponible del equipo dentro del sprint estas podían o no entrar en el mismo o quedar pendientes para un sprint posterior.
5. Priorización de las tareas: las tareas eran priorizadas por el equipo dentro de una de cinco categorías posibles: *Highest*, *High*, *Medium*, *Low*, *Lowest*. Estas prioridades indicaban el orden en el cual se deberían realizar las mismas. Muchas veces esta prioridad tenía relación con si dicha tarea resultaba bloqueante o no para algún otro miembro; o si el equipo consideraba que la misma tenía o no que ser realizada en su totalidad dentro de dicho *sprint*.

Una vez iniciado el *sprint*, cada miembro se ocupaba de realizar el trabajo, registrar las horas dedicadas y mantener actualizado el estado de cada una de sus tareas. Este método de registro permitió tener una noción más concreta del tiempo dedicado a cada ítem y así tomar decisiones como, por ejemplo, notar si un miembro estaba teniendo dificultades con una tarea o replantear la solución pensada para encontrar una manera más rápida de realizarlo. Al finalizar cada *sprint* se armaba un informe de avance que era enviado al tutor.

Finalmente, cuando una etapa era terminada se coordinaba una reunión con el tutor para mostrar las funcionalidades implementadas durante la etapa y recibir la validación de las mismas. Esto, sumado a los informes enviados, permitió que el tutor tenga en todo momento una visión clara sobre el avance realizado por el equipo, lo desarrollado dentro de cada sprint y sobre el estado del proyecto en general.

4.3. Herramientas y Tecnologías

Durante el desarrollo del trabajo se utilizaron las siguientes herramientas:

- *Github*^[14]: utilizado para alojar el código del proyecto, aprovechando su sistema de control de versiones. El desarrollo se llevó a cabo utilizando una estructura de *branches*: *main*, *develop* y *feature*, lo que permitió un desarrollo en paralelo y un manejo ordenado de las versiones. Se creó una organización en *Github* para gestionar todos los repositorios del proyecto: *mobile-app*, *content-script*, *process-api*, *recommendations-api*, *content-api* y *user-api*.
- *Figma*^[15]: herramienta utilizada para diseñar los prototipos de las pantallas de la aplicación móvil, permitiendo trasmitir una visión clara de la interfaz de usuario a todo el equipo.
- *Postman*^[16]: empleado para la prueba y documentación de las APIs desarrolladas. Se creó un equipo de trabajo que permitió a todos los miembros acceder a los mé-

todos implementados. Además, se utilizó la funcionalidad de guardar ejemplos para documentar el comportamiento de las APIs y los distintos *endpoints*.

- *React Native*^[17]: *framework* utilizado para el desarrollo de la aplicación móvil, facilitando el desarrollo multiplataforma. Se integró con la librería *React Native Paper*, que provee componentes basados en *Material Design*.
- *TypeScript*: lenguaje utilizado en el desarrollo de las APIs (*User API*, *Process API* y *Content API*), ofreciendo mayor seguridad en el código mediante el tipado estático.
- *Python*: lenguaje de programación utilizado para desarrollar el algoritmo de recomendación y la implementación de la *Recommendations API*.
- *Docker*^[21]: utilizado para contener el código del *backend*, asegurando una mayor facilidad para la ejecución en distintos sistemas operativos. Complementado con *docker-compose* para orquestar los diferentes servicios de los que algunos de los microservicios dependen (principalmente bases de datos).
- *Google Colab*^[18]: herramienta utilizada para el desarrollo del algoritmo de recomendación, brindando flexibilidad en el desarrollo y testeo del mismo mediante el uso de *Jupyter Notebooks* en la nube.
- *Visual Studio Code*: entorno de desarrollo integrado utilizado para escribir y depurar el código de las diferentes partes del proyecto.
- *Render*^[20]: plataforma de *hosting* utilizada para desplegar los servicios del *backend*, proporcionando un entorno gratuito y confiable para mantener los servidores en la nube y accesibles en todo momento.
- *Neon*^[19]: plataforma de bases de datos *SQL* utilizada para almacenar y gestionar datos estructurados.
- *MongoDB*^[22]: utilizado como base de datos *NoSQL* para almacenar datos no estructurados o semi-estructurados, como parte del *backend* del proyecto.

- *Expo*^[23]: plataforma y conjunto de herramientas utilizadas para desarrollar y construir la aplicación móvil con *React Native* de manera eficiente.
- *Google Meet y Gmail*: herramientas de comunicación utilizadas para coordinar reuniones del equipo y mantener la comunicación fluida a lo largo del proyecto con el tutor.
- *Google Slides*: utilizado para preparar los informes de avances presentados al final de cada sprint al tutor.
- *The Movie Database API*^[24]: API utilizada para obtener datos sobre películas y series, necesarios para el funcionamiento de la aplicación.
- *Codecov*^[25]: herramienta utilizada para el análisis de cobertura de pruebas, asegurando que el código de los microservicios esté adecuadamente probado a lo largo del desarrollo.

4.4. Arquitectura

Para la arquitectura del proyecto, se optó por un *backend* basado en microservicios. La idea principal fue lograr una mayor abstracción al acceder a los distintos servicios utilizados. Por este motivo, el sistema cuenta con tres APIs:

- *Content API (CAPI)*: responsable de procesar todo lo relacionado con el contenido (series, películas, servicios de *streaming* y artistas).
- *User API (UAPI)*: responsable de procesar todo lo relacionado con los usuarios.
- *Recommendations API (RAPI)*: responsable de obtener todas las recomendaciones previamente calculadas.

Estas tres APIs fueron esenciales para conseguir una mayor abstracción en el procesamiento de los principales tipos de información manejados (usuarios, contenido y recomendaciones). Sin embargo, aunque esta información es independiente en muchas ocasiones,

también necesita relacionarse en otras. Un ejemplo es la obtención del perfil de usuario, donde se requiere acceder tanto a la información personal del usuario (UAPI) como al contenido (CAPI) visto por él. Para resolver este desafío, se creó un nuevo microservicio encargado de orquestar la información entre las APIs ya mencionadas. Este microservicio se denominó *Process API (PAPI)*. De esta manera, la aplicación móvil solo se comunica con la PAPI, mientras que esta se encarga de acceder a cada una de las otras APIs cuando sea necesario, y combina la información de las mismas.

Finalmente, quedaba un único aspecto de la arquitectura por resolver: el cálculo de las recomendaciones. Este proceso, al ser muy costoso tanto en tiempo como en recursos, se decidió que fuera realizado en un proceso *batch*. El proceso consta de tres pasos principales:

1. *Obtención de los datos*: se desarrolló un *script* de *Python* capaz de obtener el contenido visto por cada usuario y los miembros de cada grupo y datasets con el contenido disponible en TMDB.
2. *Cálculo de las recomendaciones*: se realizó un *Google Notebook* utilizando *Python* capaz de calcular las recomendaciones en base a los datos obtenidos en el paso anterior.
3. *Subida de resultados a RAPI*: finalmente los resultados obtenidos son subidos a través de un *script* de *Python* a la base de datos de la RAPI.

Este proceso *batch* permite que las recomendaciones en la RAPI estén precalculadas. De esta manera, el acceso a las recomendaciones por parte del usuario es mucho más rápido que si fuera necesario calcularlas en tiempo real. Además, es altamente probable que un usuario quiera conservar sus recomendaciones por un periodo de tiempo determinado, en lugar de que estas varíen constantemente. Por este motivo, este proceso se ejecuta cada 24 horas para mantener las recomendaciones actualizadas. Finalmente, la arquitectura completa del proyecto se muestra en la Figura 4.

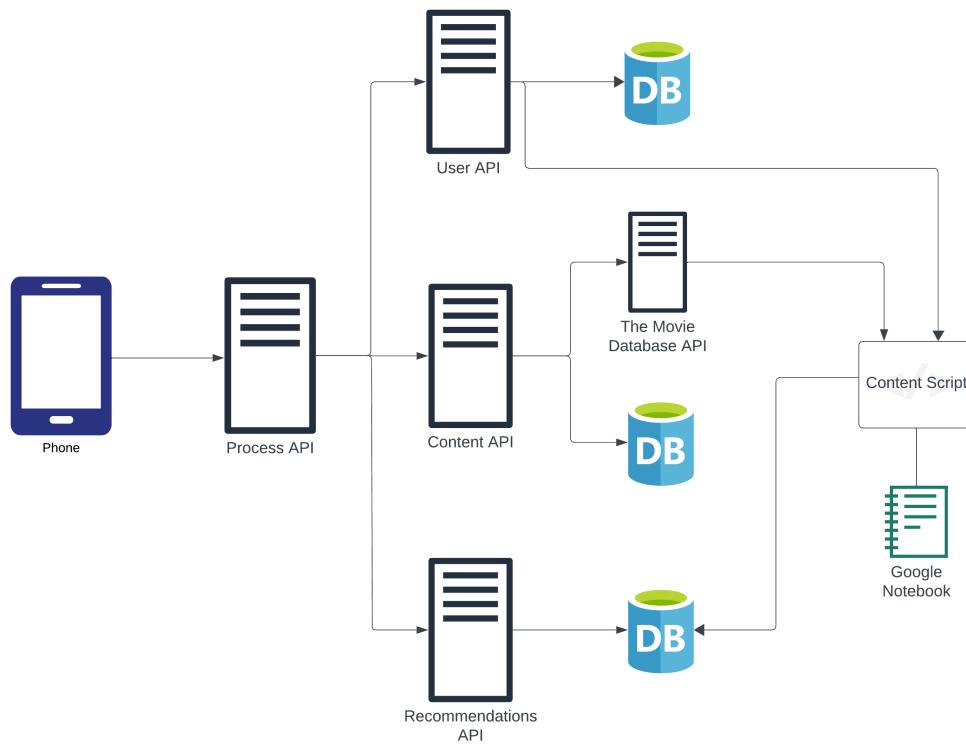
**Figura 4**

Diagrama de Arquitectura de el Sistema

4.4.1. Aplicación

El enfoque principal del trabajo fue el desarrollo de una aplicación móvil, la cual incluye cuatro grandes grupos de funcionalidades:

- Gestión de cuenta.
- Exploración de películas, series y artistas.
- Recomendaciones de películas, series y servicios de *streaming*.
- Red Social.

A continuación, se detallarán las funcionalidades de cada uno de esos ítems:

4.4.1.1 Gestión de cuenta

- *Inicio de Sesión*: el usuario puede crear su perfil en la plataforma e ingresar a la misma mediante su cuenta de *Google* (Figura 5).
- *Perfil*: dentro del perfil el usuario (Figura 6) puede ver sus datos personales tales como correo, nivel, nombre de usuario, foto de perfil, cantidad de amigos, plataformas contratadas, lista de contenido visto y *watchlist*. El nivel del usuario va aumentando según las reseñas que este realice. Tanto el nombre como la foto del usuario son editables. Para esta última, se puede elegir una entre varias opciones que se van desbloqueando según el nivel del usuario.
- *Contenido Visto*: el usuario puede ver una lista de todo el contenido visto, ordenado de más a menos reciente (Figura 7). En esta lista se muestra la fecha en la que se vio cada contenido y, en el caso de las series, el último capítulo visto.



Figura 5

Inicio



Figura 6

Perfil



Figura 7

Vistos

- *Configuración*: dentro de esta pestaña el usuario puede cambiar la visibilidad de su

contenido visto y su *watchlist*, ir a la pantalla de estadísticas de uso o cerrar sesión (Figura 8).

- *Estadísticas de Servicios*: el usuario puede ver cuánto tiempo ha dedicado a ver contenido, tanto dentro como fuera de sus servicios de *streaming* (Figura 9).
- *Gestión de plataformas*: el usuario puede agregar o eliminar las plataformas que posee (Figura 10). Esto permite mostrar si cierto contenido se encuentra disponible en alguna de las plataformas del usuario y calcular las estadísticas de uso.
- *Trivias*: el usuario puede responder trivias sobre su contenido favorito dentro de la aplicación (Figura 11).

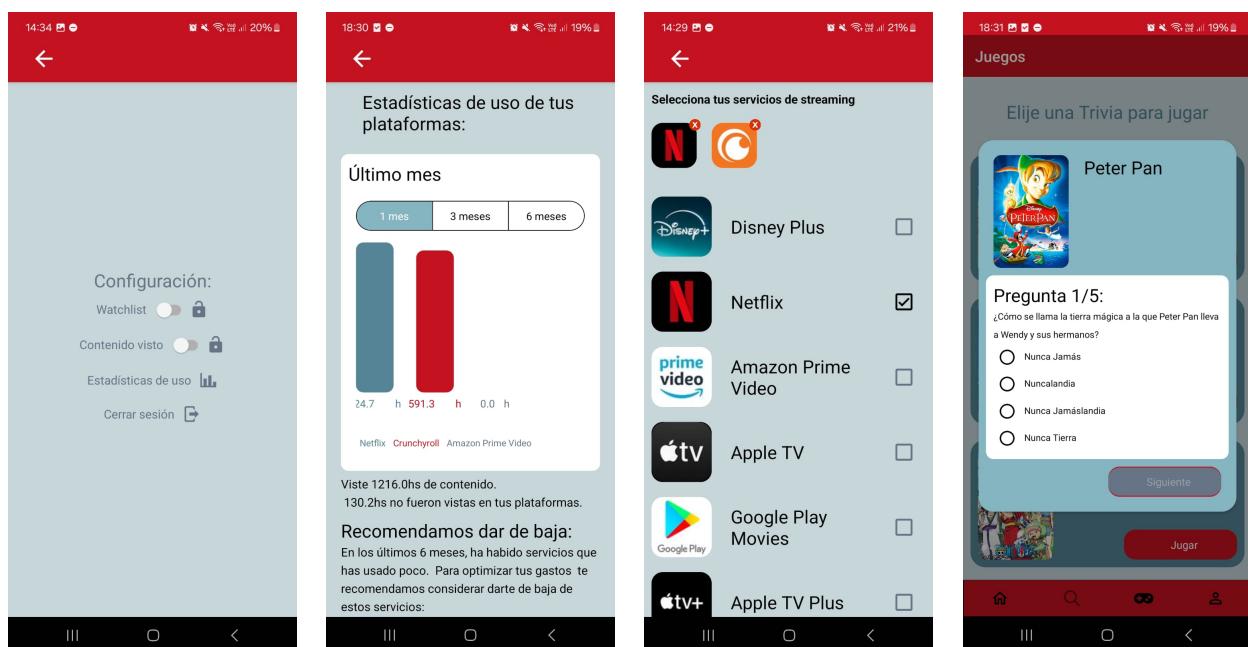


Figura 8

Configuración

Figura 9

Estadísticas

Figura 10

Plataformas

Figura 11

Trivias

4.4.1.2 Exploración de películas, series y artistas

Para explorar el contenido, se optó por utilizar la API de TMDB, que ofrece detalles sobre una amplia gama de películas, series y artistas. En este contexto, un artista se refiere

a cualquier persona involucrada en una producción audiovisual (actor, director, productor, entre otros cargos). Para cada uno de estos tipos, es posible realizar búsquedas por nombre y visualizar los detalles correspondientes.

- *Buscar contenido por nombre:* el usuario puede realizar búsquedas por título y obtener información resumida del contenido que coincide con el texto ingresado (Figura 12). Cada entrada de la lista de resultados redirige a su pantalla de detalle.

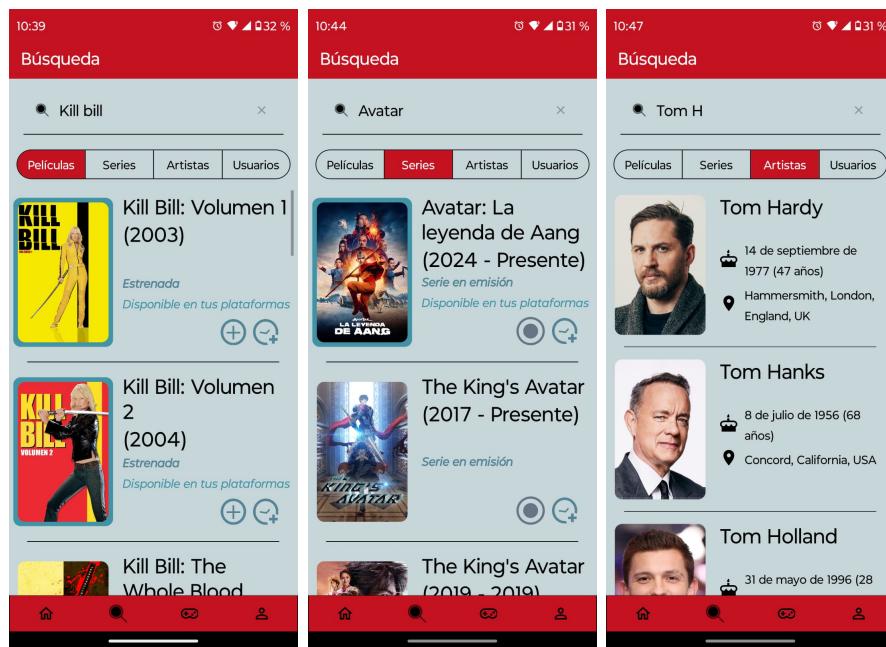


Figura 12

Búsqueda

- *Detalle de película:* se muestra el título de la película, el póster, el año de estreno, la duración y los directores (Figura 13). Seguido, se muestra información sobre las plataformas de *streaming* en las que está disponible y el estado de la película, que puede ser: estrenada, programada o en posproducción. El botón *Ver ahora* permite redirigir a la plataforma deseada para visualizar el contenido. También se incluye la descripción de la película y los géneros a los que pertenece. Finalmente, se muestra el reparto, con los actores de la película (que redirigen al detalle del artista), las reseñas

(tanto propias como de otros usuarios) y una serie de recomendaciones de películas similares incluyendo el póster, nombre y año de estreno (que redirigen al detalle de cada una).

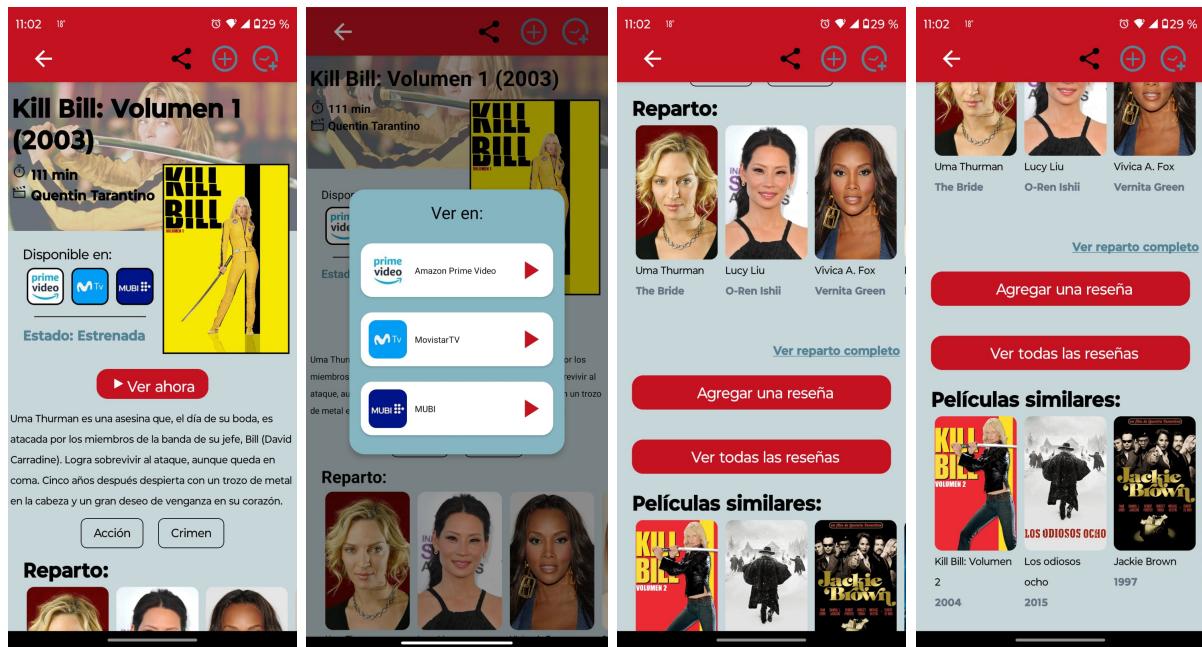


Figura 13

Detalle de Película

- *Detalle de serie:* Se puede ver el título de la serie, el póster, el año de estreno, la cantidad de episodios y temporadas, así como los creadores (Figura 14). Más abajo, aparece información sobre las plataformas de *streaming* en las que está disponible y el estado de la serie, que puede ser: finalizada, en emisión, piloto, cancelada o planificada. Se muestra información sobre el próximo capítulo que el usuario debe ver. Se incluyen el nombre del capítulo, la fecha de estreno y el botón *Ver ahora*, que redirige a las plataformas de *streaming* disponibles. También, la descripción de la serie, los géneros a los que pertenece y el listado de temporadas, cada una con su póster, nombre y fecha de estreno, los cuales llevan al detalle de la temporada. Por último, se muestra el reparto con los actores de la serie que llevan al detalle de cada artista, las reseñas tanto propias como de otros usuarios, y una lista de

recomendaciones de series similares con su póster, nombre y año de estreno, que llevan al detalle de cada una.

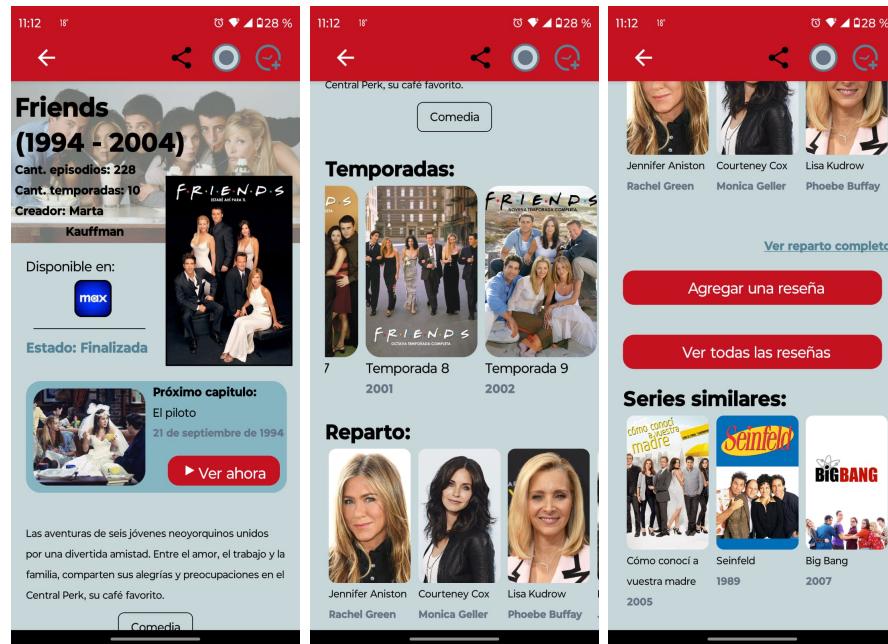


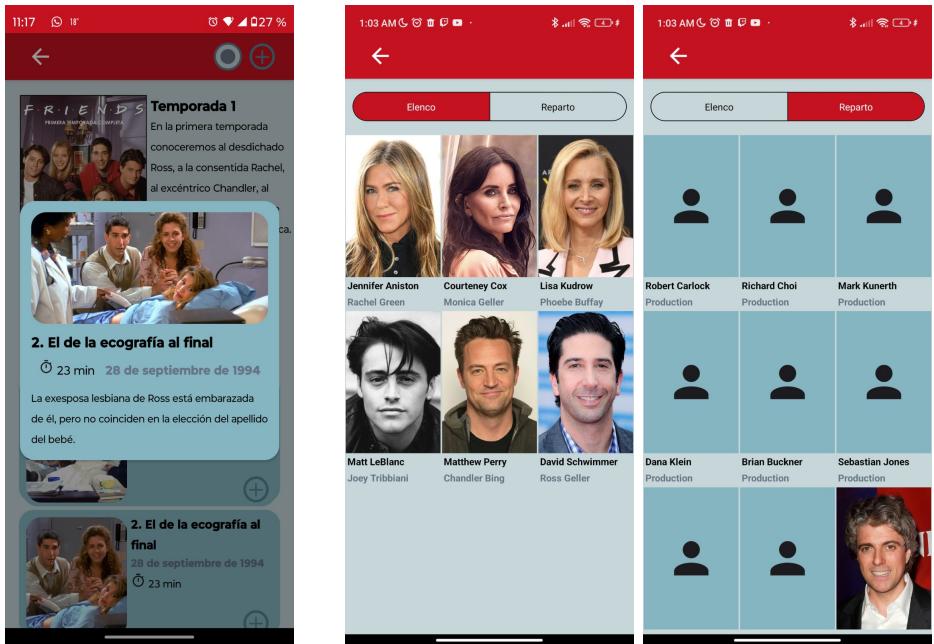
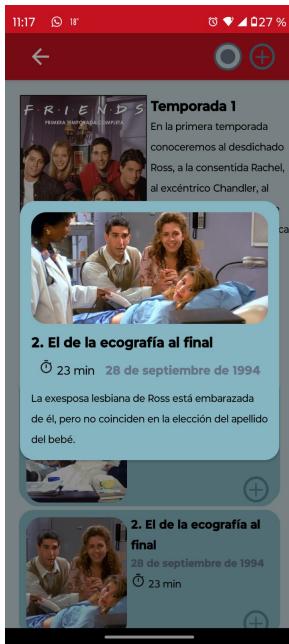
Figura 14

Detalle de Serie

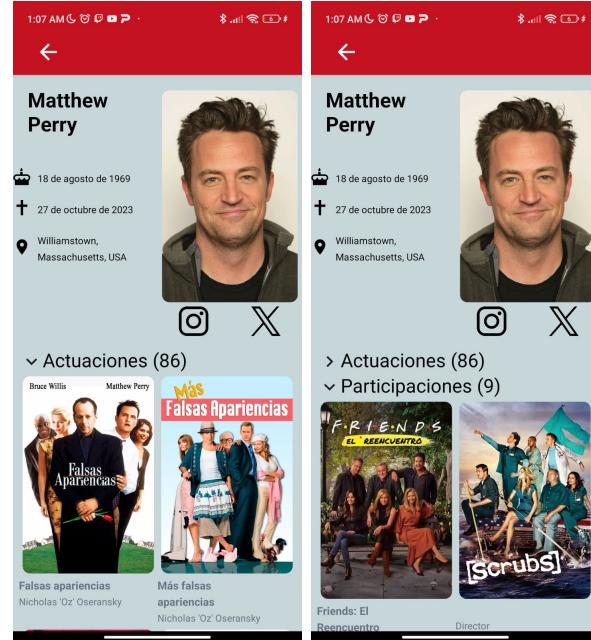
- *Detalle de temporada:* se muestra el título de la temporada, su descripción, un botón *Ver serie* para redirigir y un listado de capítulos con la cantidad indicada (Figura 15). Cada tarjeta de capítulo se puede abrir para visualizar su descripción.
- *Reparto completo:* se puede ver a los actores en la solapa *elenco* y a todo el resto de involucrados en la solapa *reparto* (Figura 16).
- *Detalle de artistas:* se muestra la fecha, lugar de nacimiento y, de corresponder, la fecha de defunción (Figura 17). También se incluyen las redes sociales del artista, que redirigen a sus respectivas plataformas. En *Actuaciones*, se presentan las películas o series en las que ha estado, y en *Participaciones* se muestran sus colaboraciones en otros roles.

**Figura 15**

Detalle de Temporada

**Figura 16**

Reparto Completo

**Figura 17**

Detalle de Artista

4.4.1.3 Recomendaciones de películas, series y servicios de *streaming*

- *Contenido similar*: es posible buscar contenido similar al visto previamente. Para esto, existe dentro de la pantalla principal la sección 'Algo similar a esto' donde si se toca el botón 'Seleccionar contenido' es posible elegir hasta 3 películas o series vistas para obtener contenido parecido (Figura 18).

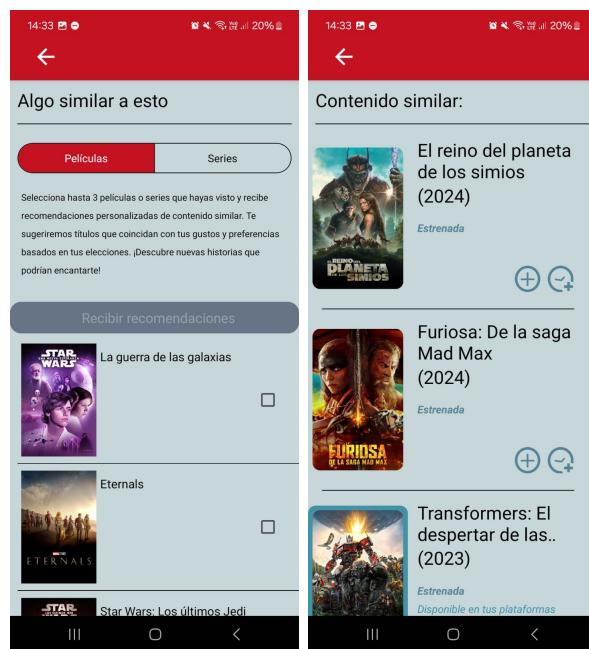


Figura 18

Contenido Similar

- *Contenido visto por amigos*: en la pantalla principal se reciben recomendaciones de contenido visto por amigos que no haya visto el usuario (Figura 19).
- *Recomendaciones personalizadas*: en la pantalla principal, se reciben recomendaciones de películas y series que el usuario no haya visto, basadas en su historial de visualizaciones (Figura 20).
- *Recomendación de Servicios*: según el tiempo de visualización en las plataformas contratadas, el sistema recomienda al usuario cancelar los servicios poco utilizados. También sugiere contratar aquellos servicios que usan sus amigos (Figura 21).



Figura 19

Recomendaciones basadas en contenido visto por amigos



Figura 20

Recomendaciones basadas en contenido visto



Figura 21

Recomendación de servicios populares entre amigos

4.4.1.4 Red Social

- *Grupos*: el usuario puede crear y eliminar grupos con sus amigos (Figura 22). Estos recibirán recomendaciones personalizadas en base a las visualizaciones de todos los miembros, tras 24 horas de la creación del mismo.
- *Amigos*: el usuario puede enviar y recibir solicitudes de amistad que pueden ser aceptadas o rechazadas (Figura 23). Además, un usuario puede explorar el perfil de otros usuarios y según la privacidad, ver su listado de contenido visto y su watchlist.

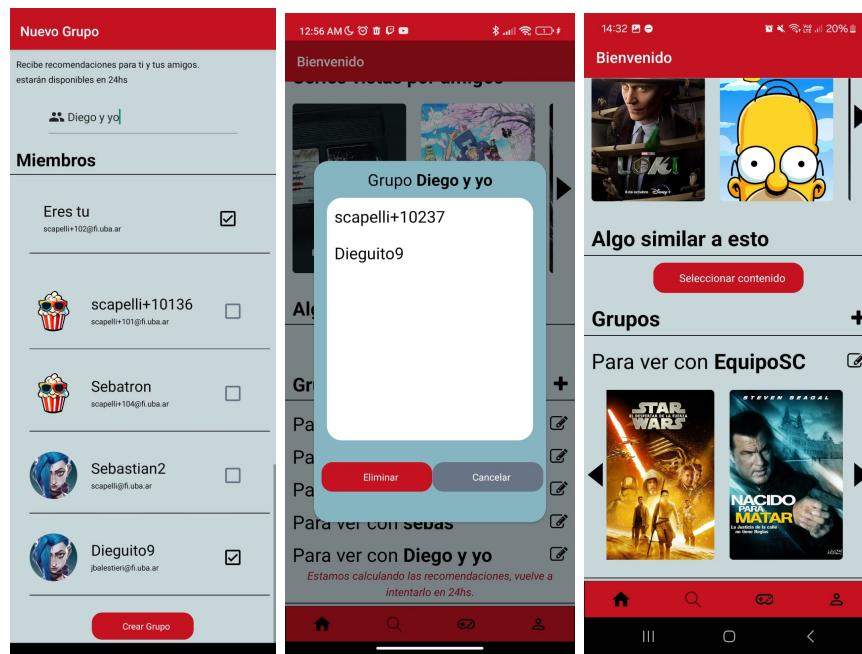


Figura 22

Grupos

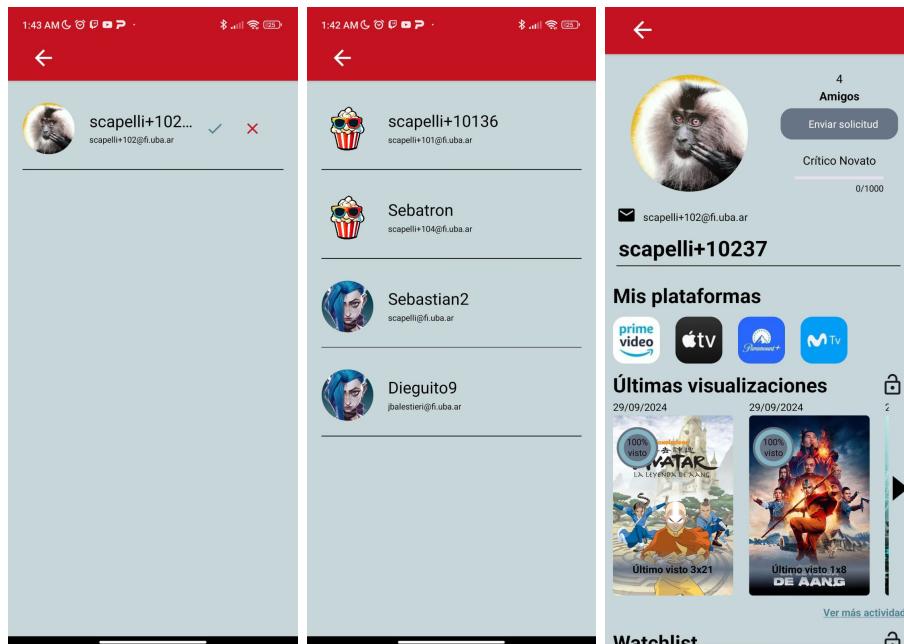


Figura 23

Solicitudes de Amistad

4.4.2. Backend

Con el objetivo de diseñar una arquitectura escalable, se optó por dividir las responsabilidades del *backend* en varios microservicios. Esto permite que cada uno de ellos se enfoque en una lógica particular del problema, reduciendo la complejidad general y facilitando su mantenimiento.

A excepción de la Recommendations API, todos los microservicios fueron desarrollados en *TypeScript*. Esta decisión se tomó debido a que el uso de *JavaScript* (JS) con *Node.js* es un estándar en la industria, y *TypeScript* ofrece las mismas cualidades que JS, con el valor añadido de ser un lenguaje tipado. El tipado estático de *TypeScript* mejora la claridad del código y permite predecir posibles errores con mayor facilidad. Además, todos los microservicios siguen una estructura de capas bien definida, como se muestra en la Figura 24.

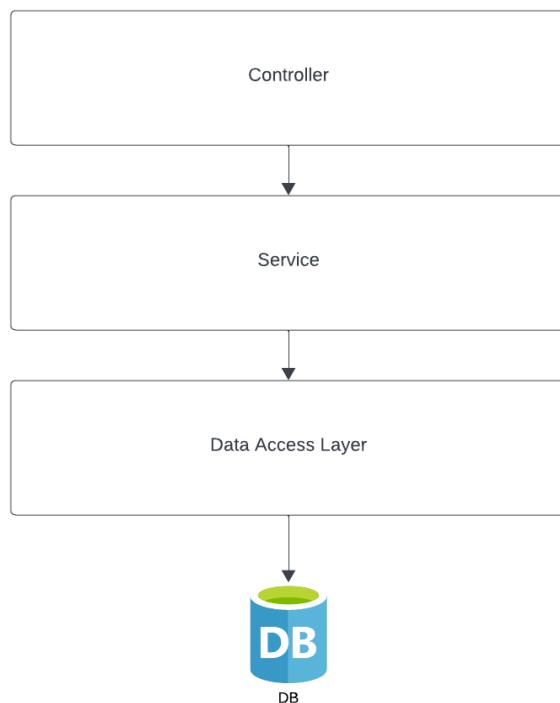


Figura 24
Estructura de Capas de Micro Servicios

- Controller: esta capa ofrece un servicio de alto nivel para el cliente del microservicio. Se encarga de invocar los servicios correspondientes para realizar la tarea solicitada y procesa los datos recibidos para dar una respuesta acorde al cliente.
- Service: en esta capa se validan las reglas de negocio de una operación y se llama a la capa de acceso de datos (DAL) cuando es necesario. Es responsable de manejar la lógica de negocio del microservicio.
- Data Access Layer (DAL): esta capa gestiona la interacción con la base de datos, encargándose de la recuperación, modificación o eliminación de datos.

Es importante destacar que, debido a su simplicidad y a la ausencia de una base de datos, la *Process API* solo cuenta con la primera capa de esta arquitectura. En este caso, la *Process API* actúa como una especie de capa de control para los demás microservicios, orquestando las operaciones necesarias entre ellos.

4.4.2.1 Process API

La *Process API (PAPI)* representa el punto de entrada de todas las peticiones provenientes del cliente hacia el resto de la arquitectura *backend*. Su función principal es delegar dichas peticiones a uno o varios microservicios correspondientes y, cuando es necesario, procesar los datos recibidos para devolverlos de forma consolidada a la aplicación móvil. La existencia de este microservicio permitió una mayor abstracción en los demás (*CAPI*, *UAPI* y *RAPI*), ya que estos no necesitan preocuparse por compartir información entre ellos, sino únicamente con la PAPI.

Además, tener un único punto de entrada para todo el sistema *backend* brinda una capa adicional de seguridad. La PAPI es responsable de validar la sesión de cada usuario, siendo el único servicio público del sistema. El resto de las APIs podrían estar alojadas en una red privada (VPC), lo que limitaría su acceso externo y aumentaría la seguridad del sistema. Sin embargo, debido a las limitaciones del proveedor de *hosting* utilizado, esta

configuración no fue implementada en el proyecto.

4.4.2.2 Content API

La *Content API (CAPI)* es la responsable de gestionar todo lo relacionado con el contenido. Su principal función es actuar como un *wrapper* de la API de TMDB, transformando la información recibida en un formato más acorde a lo utilizado por la aplicación. Es capaz de realizar búsquedas y obtener detalles de películas, series, artistas (actores, productores, etc.) y servicios de *streaming*. Este último no es retornado directamente por la API, sino que se obtiene a través de un trabajo de *web scrapping*.

Además, la CAPI es responsable de gestionar la información relacionada con el contenido propio de la aplicación, como reseñas, trivias, contenido visto por cada usuario, *watchlists* y la privacidad de estas dos últimas. Para la persistencia de dicha información, se utiliza una base de datos *MongoDB*.

El uso de una base de datos *Mongo* se debe a su simplicidad y capacidad de crear *clusters*, lo que resuelve la necesidad de acceso rápido a una potencial gran cantidad de datos almacenados (principalmente contenido visto y *watchlist*). Por otro lado, las desventajas de este tipo de base de datos no generan un impacto significativo, ya que la información almacenada no requiere ser combinada ni es crítica. Por lo tanto, los problemas de consistencia que podrían llegar a surgir no afectarían gravemente el funcionamiento de la aplicación.

La base de datos contiene las siguientes colecciones:

- *Stream providers*: almacena las horas vistas por cada usuario correspondientes a las plataformas de *streaming* que posee. Esta información se utiliza para calcular las estadísticas de uso y para determinar si un contenido específico se encuentra en alguna de las plataformas del usuario.
- *Other stream providers*: almacena las horas vistas por cada usuario que no corresponden a ninguna plataforma de *streaming* que posea. Esta información se utiliza

para calcular las estadísticas de uso.

- *Privacies*: almacena la información de privacidad de *watchlist* y contenido visto de cada usuario.
- *Reviews*: almacena las reseñas de películas y series.
- *Seen contents*: almacena el contenido visto por cada usuario.
- *Trivias*: almacena las trivias, sus preguntas y respuestas.
- *Watchlists*: almacena la *watchlist* de cada usuario.

4.4.2.3 User API

La *User API (UAPI)* es la responsable de gestionar la información de los usuarios. Sus principales funciones incluyen gestionar la autenticación, mantener la información personal de cada usuario (nombre, foto de perfil y correo electrónico), así como administrar sus amigos, grupos y los puntos obtenidos. A diferencia de la CAPI, esta API almacena información crítica, por lo que es necesario asegurar la consistencia de los datos. Por este motivo, se optó por utilizar una base de datos *SQL*, estructurada en las siguientes tablas:

- *Friends requests*: almacena las solicitudes de amistad pendientes.
- *Friends*: almacena los amigos de cada usuario.
- *Groups*: almacena los miembros y la información de cada grupo (*no* las recomendaciones).
- *Levels*: almacena los niveles y el rango de puntos necesario para pertenecer a cada uno de ellos. Mantener esta información en una tabla permite una mayor flexibilidad al momento de realizar cambios en esa estructura (por ejemplo, agregar un nuevo nivel o modificar los rangos de puntos de los niveles existentes).

- *Photos*: almacena las imágenes y sus identificadores, con el formato: {nivel}{identificador de imagen}.
- *Points*: almacena la cantidad de puntos que posee cada usuario.
- *Tokens*: almacena los *tokens* de sesión.
- *Users*: almacena la información personal de cada usuario.

La Figura 25 se presenta un diagrama que muestra cómo se relacionan las tablas mencionadas:

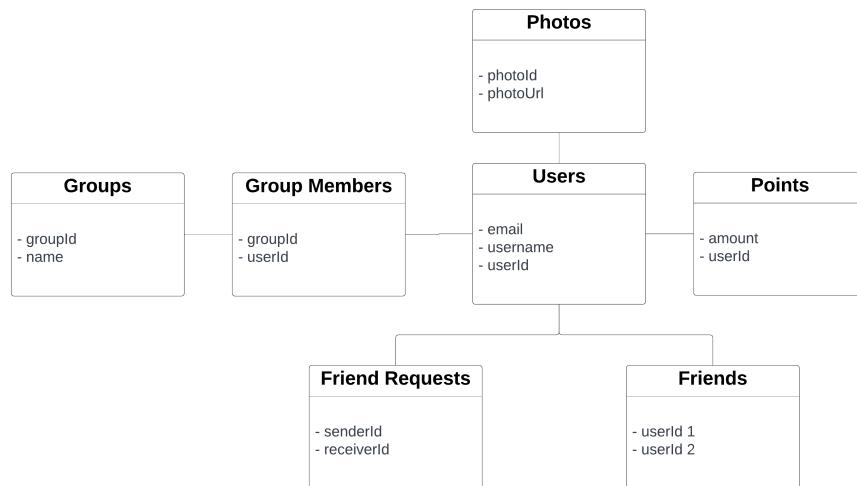


Figura 25

Relación de Tablas en UAPI

4.4.2.4 Recommendations API

Esta API es responsable de entregar las recomendaciones calculadas a través del proceso *batch*. Se comunica con una base de datos *SQL* que almacena los identificadores (*IDs*) del contenido recomendado. La RAPI es la más sencilla de todas y su única función es proporcionar los *IDs* de recomendaciones asociados a usuarios, películas, series y grupos a la PAPI, que se encarga de procesar estos *IDs* y entregar las recomendaciones con los

datos relevantes asociados (título, póster, año). Con la RAPI, se logra separar el problema de las recomendaciones del resto de la arquitectura, lo que disminuye la complejidad de los demás microservicios.

La base de datos asociada a la RAPI cuenta con las siguientes tablas:

- *Group movie recommendations*: almacena las recomendaciones de películas para cada grupo.
- *Movie movie recommendations*: almacena las recomendaciones de películas para cada película (películas similares).
- *Series series recommendations*: almacena las recomendaciones de series para cada serie (series similares).
- *User movie recommendations*: almacena las recomendaciones de películas para cada usuario.
- *User series recommendations*: almacena las recomendaciones de series para cada usuario.

4.4.3. Cálculo de recomendaciones

Se utilizó un sistema de recomendaciones basado en contenido, esto es, el modelo analiza determinados atributos del contenido consumido por el usuario para determinar sus preferencias y así realizar las recomendaciones.

Inicialmente, se implementó una versión básica utilizando distancia coseno, que permitía recomendaciones de tipo ítem-ítem. Este algoritmo inicial consistía en transformar cada ítem en un arreglo numérico. Para el caso de las películas, se utilizaron los siguientes campos: duración, año de estreno, votos promedios de TMDB, popularidad en TMDB, top palabras clave, lenguaje, país, productora y géneros. Dado que los géneros son un campo de texto con un número limitado de opciones, se realizó *One Hot Encoding*^[28]. Este proceso

consiste en transformar cada género en una columna y completar con 1 o 0, dependiendo de si la película pertenece a ese género o no.

Un proceso análogo se llevó a cabo para las series, con algunas diferencias en los campos utilizados: no se utilizó la duración y se incorporó la fecha de fin de la serie, con la salvedad de que, si la serie no está finalizada, se utiliza la fecha de estreno del último capítulo.

Una vez convertido cada ítem en un arreglo, se procede al cálculo de las recomendaciones. Este paso consiste en tomar cada ítem y calcular la distancia coseno con todos los otros ítems procesados. Una vez calculadas esas distancias, se seleccionan los N ítems más cercanos. Aunque este proceso era funcional, no era óptimo, ya que implicaba calcular todas las distancias posibles, lo que resultaba en un elevado tiempo de procesamiento. Por esta razón, en futuras etapas del desarrollo, se cambió a un modelo basado en *K-Nearest Neighbors*^[30] (K-vecinos más cercanos) de la librería de *Python Scikit-learn*^[29]. Este cambio permitió reducir considerablemente los tiempos de cálculo de las recomendaciones.

Una vez finalizado el proceso de desarrollo de las recomendaciones de tipo ítem-ítem, fue necesario realizar las recomendaciones usuario-ítem, tanto para películas como para series. En este caso, era importante transformar al usuario en un arreglo para luego calcular las distancias ya explicadas anteriormente. Para esto, se optó por considerarlo como una combinación de su contenido visto, se suman los arreglos generados para cada ítem visto por el usuario y se normalizan los valores. Convirtiéndolo, así, en un nuevo ítem, comparable y medible con el resto.

Cabe aclarar que se realiza una separación entre las películas y las series. Esto se debe principalmente a que los campos utilizados para el cálculo de las recomendaciones son distintos entre sí. Además, los géneros son diferentes para series y películas. Por este motivo, las recomendaciones de series para el usuario se calculan en base a sus series vistas, y lo mismo ocurre para las películas. Esto trae la desventaja de que, si un usuario solo ve películas, el algoritmo difícilmente haga buenas recomendaciones de series por no contar con información suficiente.

Una de las principales dificultades fue obtener los *datasets* de películas y series de TMDB, ya que resultaron necesarios para entrenar el modelo y realizar los cálculos. Dado que TMDB no ofrece acceso directo a su base de datos de películas y series, era necesario realizar consultas individuales a través de su API para obtener información. Inicialmente, se optó por desarrollar un *script* que realizaba *data scraping* de TMDB para construir los datasets de películas y series. Sin embargo, esta opción fue descartada debido a que se trataba de un proceso demasiado lento e ineficiente. Por este motivo, se utilizó un *dataset* de TMDB encontrado en la plataforma *Kaggle*^[8]. Esto simplificó el problema de la falta de datos. Sin embargo, utilizar un *dataset* fijo implica que los datos no serán actualizados; por ejemplo, ocurren casos de IDs que fueron cambiados, generando conflictos en la aplicación. Además, el *dataset* obtenido en *Kaggle* no contenía información sobre las plataformas de *streaming* en las que estaba el contenido, limitando la versatilidad de las recomendaciones basadas en las plataformas del usuario.

En este punto, se planteó que obtener información actualizada implicaría tener una relación más estrecha con TMDB, y por tratarse de un trabajo académico, no era algo posible. Por otro lado, se planteó que utilizar un *dataset* de estas características sería suficiente para alcanzar los objetivos propuestos por el trabajo profesional y se continuó con esta opción.

4.4.3.1 Métricas

De la mano de cualquier modelo de recomendaciones, se recomienda que exista algún método para medir la calidad de los resultados. En el caso de este modelo, lo más adecuado es utilizar métricas que indiquen la calidad de un subconjunto de recomendaciones. Para ello, se debe disponer de un *dataset* de usuarios con contenido visto y con los mismos atributos presentes que en el *dataset* utilizado para el entrenamiento del modelo.

En este caso, eso resultó una gran dificultad, ya que los *datasets* encontrados con estas características resultaban incompatibles con los utilizados en el entrenamiento. Además,

carecían de características relevantes para el cálculo de las recomendaciones, como la cronología de los visionados.

Otro gran problema era la conversión de IDs, ya que los datasets obtenidos de *Letterboxd* trabajan con IDs de IMDB, lo que requería convertir los IDs hacia los de TMDB. En el proceso, podían surgir incompatibilidades no contempladas que producían la pérdida de información y que hacían que el *dataset* resultante no fuera confiable.

Finalmente, se planteó que, en lugar de intentar forzar métricas a partir de *datasets* incompatibles, estos inconvenientes se solucionarían de manera *natural* en un entorno de producción, donde se disponga de retroalimentación proporcionada por usuarios reales de la aplicación. Los datos generados en ese contexto serían mucho más valiosos para ajustar y mejorar el modelo, ya que reflejarían el uso real de la aplicación y proporcionarían una referencia ideal para optimizar las recomendaciones en futuras iteraciones.

4.5. Desarrollo

Inicialmente, se llevó a cabo una etapa de investigación y pruebas de concepto para estudiar la viabilidad del proyecto. A partir de *octubre de 2023*, ya con la propuesta aprobada, se inició la etapa formal de desarrollo mediante *sprints* quincenales.

Durante todo el proceso, se mantuvieron reuniones regulares con el tutor Diego Corsi para mantenerlo actualizado sobre el estado del proyecto. En algunas ocasiones, fue necesario realizar reuniones especiales para reajustar el alcance de ciertas funcionalidades y sus criterios de aceptación. Además, al final de cada *sprint* se enviaron informes con detalles de los últimos progresos.

Durante el avance del proyecto surgió la necesidad de conseguir asesoramiento sobre ciertos aspectos que resultaron problemáticos. A continuación se destacan las colaboraciones destacadas.

4.5.1. Especialista UX

En relación con el diseño de las primeras pantallas de la aplicación, el proceso experimentó varios ajustes. Al principio, se incluía demasiada información en cada pantalla y se intentaba abarcar múltiples funcionalidades a la vez. Esto se debía principalmente a dos razones: primero, no se tenía una noción clara de cuántos elementos podían caber en una pantalla de celular en comparación con lo que se veía en el editor de *Figma*; segundo, se buscaba crear un diseño casi final de una sola vez, lo que complicaba el proceso iterativo de desarrollo.

En este contexto, se realizaron consultas por videoconferencia y correo electrónico con Maia Naftali, especialista en UX. Con su asesoramiento, se ajustó el proceso de diseño de pantallas a uno basado en los flujos de navegación del usuario. Este enfoque implica comenzar con bocetos simples de las pantallas que participan en una funcionalidad específica, añadiendo o eliminando secciones según sea necesario. Posteriormente, estos bocetos se convierten en diseños más detallados en *Figma*. Un ejemplo de uso, es la funcionalidad de búsqueda de contenido, donde el flujo del usuario incluye ingresar al menú de búsqueda, escribir el título de interés y recibir los resultados correspondientes. Este nuevo enfoque simplificó el proceso de diseño, evitando la dispersión en detalles relacionados con otras características que serían abordadas más adelante.

4.5.2. Especialista *Machine Learning*

La puesta en marcha del modelo de recomendaciones fue particularmente difícil, ya que inicialmente se intentaba utilizar un modelo desplegado en la nube, listo para realizar recomendaciones bajo demanda. En este punto, se contactó a Jorge Collinet, especialista en *Machine Learning*, para explorar alternativas. Con su ayuda, se simplificó el problema de los recursos al implementar una versión local del algoritmo de recomendación, utilizando los recursos locales y realizando el cálculo de las recomendaciones en un proceso *batch*.

Más adelante, se llevaron a cabo otras reuniones con el especialista para realizar peque-

ños ajustes en el algoritmo, aunque esta primera reunión se considera la más importante, ya que permitió desbloquear el avance.

4.6. Planificación y Resultado

El desarrollo del proyecto fue planificado en seis etapas, cada una de ellas con una duración estimada. Debido a diversas situaciones imprevistas, el tiempo planificado no se cumplió con precisión. Sin embargo, mediante diversos ajustes aplicados durante el proceso, fue posible mantener bajo control las desviaciones de los resultados respecto de las estimaciones, logrando efectivamente alcanzar la sexta y última etapa durante el mes en que estaba previsto. El objetivo de esta sección es contrastar la planificación inicial con lo que realmente sucedió. Para ello, se realizó el diagrama de la Figura 26.



Figura 26
Comparación entre planificación y resultado alcanzado

A continuación, se detallan los principales acontecimientos de cada una de las etapas. Se optó por ignorar la etapa 6, debido a que consiste en el cierre del presente informe y pruebas de la aplicación, y por lo tanto no cuenta con acontecimientos a destacar.

- *Etapa 1. Definición de historias de usuario y requerimientos. Diseño de la arquitectura e implementación del "walking skeleton". Diseño de las primeras pantallas de la aplicación.*

Esta primera etapa se extendió a casi dos meses, uno más de lo estipulado. Los principales motivos fueron la necesidad de adaptación del equipo al ritmo de trabajo, a la falta de definición de fechas límite y a problemas con la selección de las plataformas de despliegue de los servicios (ver sección Render y AWS).

El diseño inicial del prototipo de la aplicación también requirió más tiempo del esperado, ya que el equipo contaba con poca experiencia en ese ámbito. Sin embargo, con la ayuda de una especialista en UX, se logró definir un sistema de diseño basado en flujos de navegación del usuario, lo que permitió reducir la cantidad de pantallas necesarias y dar mayor espacio al desarrollo (ver sección Especialista UX).

En cuanto a las historias de usuario y requerimientos, inicialmente se gestionaban mediante hojas de cálculo en *Excel*. Sin embargo, rápidamente se advirtió que esta metodología no era sostenible a lo largo del proyecto, principalmente debido a la gran cantidad de historias y tareas y su tasa de cambio. Por este motivo, se migró la gestión del proyecto a *Jira*, lo que facilitó la administración de las tareas y permitió organizar el desarrollo en *sprints*.

A partir del tercer *sprint*, se incorporaron informes de avance para documentar el estado del proyecto y el progreso del último *sprint*, lo que ayudó a mantener un contacto más estrecho con el tutor y a tener una mejor visibilidad del avance total del proyecto.

- *Etapa 2. Integración con TMDb. Búsqueda y detalles de contenido. Redirección hacia las plataformas de streaming desde la aplicación. Implementación inicial del modelo de recomendaciones. Perfil de usuario.*

Esta etapa fue la más extensa, ya que consistía en sentar las bases de toda la aplicación. El tiempo de desarrollo fue erróneamente estimado, lo que generó diversas demoras que fueron necesarias para desarrollar correctamente las funciones básicas, facilitando luego la expansión e incorporación de nuevas características. Los principales factores que provocaron estas demoras fueron:

- La funcionalidad de búsqueda de contenido fue la más compleja y demorada, debido a numerosas dificultades no anticipadas. El principal desafío fue la implementación de las solapas (Películas, Series y Artistas) y la integración con la API de TMDB. Además, fue necesario realizar retrabajo para incorporar *Redux* (ver sección *Redux*) y reducir la complejidad del código.
- Aunque se trabajó durante las vacaciones de verano, se acordó una reducción del esfuerzo en enero. Esta disminución fue compensada en febrero, cuando se revaluó la dedicación del equipo y se decidió modificar la organización y distribución de tareas, marcando un punto de inflexión importante (ver sección Organización y distribución de tareas).
- Las funcionalidades principales de la aplicación, como la búsqueda de contenido, marcar contenido como visto, agregar a la *watchlist*, gestión de plataformas de *streaming* contratadas y perfil de usuario, eran esenciales para el uso general de la aplicación. Por ello, estas funcionalidades se desarrollaron con un alto nivel de detalle para garantizar la mejor experiencia posible. Posteriormente, el tiempo extra invertido en estas áreas fue recuperado de funcionalidades menos esenciales.
- Las investigaciones acerca del algoritmo de recomendación fueron lentas debido a la falta de conocimiento del equipo. Para superar ese obstáculo, fue necesario coordinar reuniones con un especialista para esclarecer dudas.

Durante esta etapa, también se mejoró la comunicación dentro del equipo y el compromiso con el tablero de *Jira*, ya que se había detectado que las actualizaciones sobre el estado de cada tarea no se refrescaba con frecuencia.

- *Etapa 3. Sistema de puntos. Mejora del modelo de recomendaciones. Privacidad. Lista de amigos. Integración con otras redes sociales. Inicio de sesión con Google. Reseñas.*

A partir de esta etapa, el inicio de cada fase se superpuso con el final de la anterior, con el objetivo de evitar bloqueos en el desarrollo del *frontend*. Se iniciaba el desarrollo

del *backend* y el diseño de las pantallas mientras se ultimaban los detalles pendientes de la etapa anterior.

Además, antes de comenzar esta fase, se realizó una reunión con el tutor para redefinir el alcance del proyecto y repriorizar las distintas funcionalidades planificadas, dando mayor importancia a los aspectos más superadores de la aplicación. Un mejor entendimiento por parte del equipo sobre los elementos a trabajar también permitió realizar estimaciones más realistas y precisas.

- *Etapa 4. Trivias. Recomendaciones basadas en amistades. Función descubrir. Estadísticas de uso de servicios de streaming y recomendaciones de alta y baja de plataformas.*

Debido a la efectividad de la reunión de reajuste de alcance y repriorización realizada con el tutor la etapa anterior, al comenzar esta etapa se repitió esta reunión, de forma más breve y enfocada en ciertas funcionalidades que habían quedado pendientes de la reunión anterior.

En esta etapa se implementó la funcionalidad de filtrado de contenido que inicialmente, se había pensado como un complemento a la búsqueda de contenido por título. Sin embargo, la API de TMDB no permite realizar este tipo de filtros junto a la búsqueda por título, por este motivo, se ajustó la funcionalidad a una llamada *Descubrir* que permite realizar búsquedas por año, duración y género, pero no por título.

Para las recomendaciones basadas en amistades, se optó por recomendar contenido visto por sus amigos no visto por el usuario. Además, para la sección de *Juegos Interactivos en Base a Contenido Visto* se optó por ofrecer un *set* de trivias basadas en distintos tipos de contenido.

- *Etapa 5. Alta y baja de grupos. Recomendaciones para subconjunto de contenido.*

El cálculo de recomendaciones para grupos se realizó sin mayores inconvenientes ya que consistió en adaptar los datos de los grupos al modelo de recomendaciones basados en usuarios.

Otra de las funcionalidades desarrolladas, fue la recomendación por subconjunto de contenido. Para esto, se reutilizó lo desarrollado para filtrado realizando una combinación de las características de cada uno de los elementos seleccionados. Esto se debe a que, a diferencia de las otras recomendaciones, este proceso no puede ser *batch*.

4.7. Proceso de Despliegue

Para el despliegue de los sistemas que componen la aplicación, se utilizaron varias herramientas clave, siendo *Github* la principal, ya que es responsable de mantener todo el código desarrollado y facilita la colaboración entre los distintos miembros del equipo. La integración continua fue uno de los pilares del desarrollo, y para esto, se empleó *Render*, que se conectó directamente a *Github*. Gracias a esta integración, cualquier *merge* realizado en la rama principal (*main*) desencadena automáticamente un nuevo despliegue de los microservicios, lo que asegura que la versión más reciente del código siempre esté en producción. Este enfoque minimiza errores manuales y permite una mayor agilidad en el ciclo de vida del desarrollo.

En cuanto a la infraestructura de bases de datos, se adoptaron soluciones en la nube para garantizar la escalabilidad y la disponibilidad de los datos. *MongoDB* está alojado en *MongoDB Atlas*, una plataforma gratuita que facilita la gestión de bases de datos *NoSQL*. Las bases de datos *SQL*, por otro lado, están desplegadas en *Neon*, una plataforma gratuita que ofrece una infraestructura óptima para manejar datos estructurados. Algunas de las bases de datos fueron creadas manualmente directo desde sus respectivas plataformas mientras que otras, debido al uso de ORMs^[32], se creaban de forma automática al momento de desplegar el microservicio. Todas son accedidas de manera segura por los microservicios a través de conexiones a internet. Este modelo distribuido asegura que la aplicación pueda crecer sin problemas a medida que aumente la cantidad de usuarios y los volúmenes de datos, proporcionando una arquitectura robusta y escalable.

5. Dificultades y Lecciones Aprendidas

5.1. Dificultades

Durante el desarrollo del proyecto, se enfrentaron diversos desafíos técnicos y organizativos que impactaron tanto el proceso de implementación como la planificación general. Estas dificultades surgieron en distintas etapas del desarrollo y requirieron soluciones creativas y ajustes en la metodología para mantener el avance del proyecto. A continuación, se describen las dificultades encontradas más destacadas y las estrategias empleadas para superarlas.

5.1.1. *Expo Router*

Durante la implementación del *Walking Skeleton*, en la primera etapa del proyecto, uno de los desafíos fue decidir el tipo de enrutamiento a utilizar. En este contexto, *Expo Router* se presentó como una opción novedosa, caracterizada por su enrutamiento basado en archivos y directorios, lo que representaba una innovación dentro del ecosistema de desarrollo móvil. Aunque el equipo contaba con una familiaridad considerable con *React Navigation*, una solución establecida y probada a lo largo del tiempo, la transición hacia *Expo Router* implicó un esfuerzo significativo. Fue necesario invertir tiempo en la lectura de la documentación y en la realización de pruebas para adaptarse a las nuevas características introducidas por *Expo Router*.

5.1.2. *Redux*

En la segunda etapa del proyecto, durante la implementación de la *Integración de Contenido de Plataformas de Streaming*, surgió un problema relacionado con el manejo de estados en *React*. Se observó que los estados estaban siendo pasados de forma dispersa a lo largo de la aplicación, lo que dificultaba su mantenimiento. Esta situación resaltó la

necesidad de utilizar un sistema de estados globales, por lo que se decidió integrar *Redux* y su *store*.

Redux, una tecnología para el manejo centralizado de estados, era desconocida para el equipo, lo que requirió una adaptación significativa. En este punto, se llegó a considerar la posibilidad de volver a utilizar *React Navigation* debido a la complejidad que implicaba envolver toda la aplicación en el componente principal de *Redux*. No obstante, se logró adaptar la solución de manera que fuera compatible con *Redux*, lo que permitió un flujo de estado más ordenado, eficiente y escalable.

Como se muestra en la figura 27, *Redux* permite gestionar el estado de la aplicación de manera predecible y centralizada. Se basa en el concepto de un único *store* que contiene todo el estado de la aplicación, lo que permite a los componentes acceder a los datos de manera sencilla y eficiente. Además, mediante el uso de *actions* y *reducers*, *Redux* proporciona un flujo de datos unidireccional, lo que hace que el estado de la aplicación sea más fácil de entender y depurar.

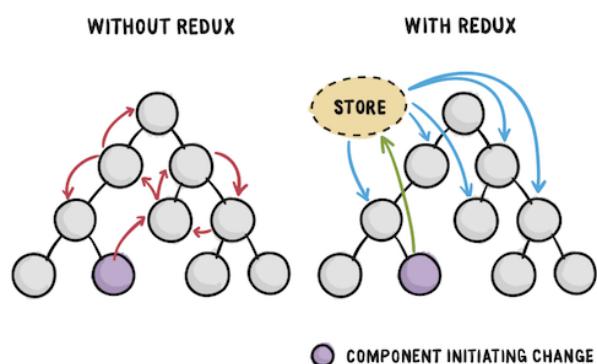


Figura 27

Redux

5.1.3. TMDB y Just Watch

Inicialmente se consideró utilizar la API de *Just Watch*, que provee acceso a la base de datos más grande y precisa de servicios de *streaming* en más de 46 países, pero se descartó

esta opción debido a su alto costo mensual, pues no se consideró un gasto justificado debido a la naturaleza del presente trabajo.

Entre las alternativas accesibles, se optó por TMDB: se trata de una base de datos muy popular de películas y series que ofrece una API gratuita. Sin embargo, esta elección presentó algunas desventajas, ya que resultó ser considerablemente lenta. Las consultas, al realizarse directamente a TMDB, sufren de tiempos de respuesta elevados, lo que afecta la fluidez de la interacción con el sistema.

Otra complicación con TMDB fue la falta de enlaces directos a las plataformas de *streaming* en su API. Sin embargo, como estos enlaces están disponibles en su sitio web, se optó por combinar los IDs del contenido con las URLs correspondientes y extraer los enlaces directamente del HTML de la página mediante un proceso de *web scrapping*.

5.1.4. *Render y AWS*

Inicialmente se intentó utilizar *Amazon Web Services* (AWS) con *Elastic Beanstalk*, una herramienta que facilita la gestión y el despliegue de aplicaciones. Sin embargo, se encontraron dificultades para realizar su configuración de forma correcta. Además, durante estos intentos de despliegue, un error en dichas configuraciones resultó en cargos inesperados en la tarjeta de crédito de uno de los integrantes.

Debido a estos problemas, se decidió cambiar a una plataforma gratuita. La elegida fue *Render*, con la cual ya se tenía experiencia en proyectos previos durante la carrera. *Render* permitió desplegar todos los servicios y APIs de manera directa y gratuita. Aunque ofreció una opción más accesible, se encontraron algunos inconvenientes, principalmente en los tiempos de arranque de los servidores y en la latencia de las consultas. Durante el desarrollo *frontend*, se interactuó directamente con los servicios alojados en *Render*, lo que hizo que las demoras fueran aún más evidentes, afectando la fluidez del proceso de desarrollo.

5.2. Riesgos Materializados

Durante el proyecto ocurrieron varios eventos que impactaron en su desarrollo. A continuación, se detallan cuáles fueron y cómo se gestionaron.

5.2.1. Licencia de *Jira*

Debido a un error, el proyecto de *Jira* estaba asociado a la cuenta de un alumno que no formaba parte del equipo. Al ya no pertenecer a la Universidad, este usuario perdió su licencia gratuita, lo que impidió el acceso al proyecto en *Jira*, provocando la pérdida de acceso a todas las historias, *bugs* y tareas que habían sido creadas.

Para resolver el problema, se contactó al equipo de soporte de *Atlassian*, que ofreció una extensión del periodo de prueba. Durante ese tiempo, se logró migrar lo creado a un nuevo proyecto registrado a nombre de *Stream Club*.

5.2.2. Ausencias

Durante el desarrollo del proyecto, algunos miembros del equipo tomaron ausencias programadas por vacaciones, lo que requirió ajustes en la organización interna para asegurar la continuidad de las tareas críticas. Para mitigar el impacto, se organizaron reuniones de transferencia de conocimiento, donde se compartieron los detalles clave de las tareas en curso con otros miembros del equipo. Además, se reajustaron los *sprints* de acuerdo a la capacidad restante del equipo, priorizando las tareas más urgentes. Posteriormente, se procuró recuperar el tiempo perdido en los *sprints* siguientes, optimizando la planificación para cumplir con los plazos establecidos.

5.2.3. Base de datos gratuita de *Render*

Inicialmente, se utilizó la plataforma *Render* (la misma que se empleó para los micro-servicios) para alojar las bases de datos. Sin embargo, *Render* ofrece este servicio de forma gratuita solo durante un mes. Debido a esta limitación, se decidió migrar la base de datos a *Neon*, una plataforma similar que ofrece un servicio gratuito ilimitado. Esta migración fue esencial, ya que la persistencia de los datos era crucial tanto para el desarrollo como para las pruebas de la aplicación, asegurando que no hubiera interrupciones en el almacenamiento ni en el acceso a la información.

5.3. Lecciones aprendidas

La ejecución de este proyecto enfrentó al equipo a diversos desafíos, de los cuales se obtuvieron valiosos aprendizajes. A continuación, se describen las lecciones más importantes que surgieron a lo largo del proceso, principalmente en lo que respecta a la gestión del equipo y del proyecto.

5.3.1. Organización y distribución de tareas

En la sección *Metodologías de trabajo*, se detalla cómo se gestionaban las tareas dentro del equipo. Sin embargo, este proceso no fue siempre constante, sino que evolucionó a lo largo del tiempo debido a distintos aspectos observados por el equipo.

Inicialmente, el progreso se evaluaba únicamente en función de la cantidad de tareas completadas. No obstante, esta información resultaba incompleta, ya que era difícil identificar las causas detrás del incumplimiento de las tareas por parte de algún miembro del equipo: si se debía a un inconveniente que le impidió dedicar el tiempo necesario al proyecto, o si surgió algún problema que hizo que la tarea fuera más compleja de lo estimado inicialmente. Por esta razón, se decidió comenzar a registrar la cantidad de horas dedicadas a cada tarea, con el objetivo de identificar claramente ambos factores y poder actuar en

consecuencia.

Las reuniones semanales fomentaron una comunicación abierta y colaborativa dentro del equipo. Este enfoque permitió que todos los miembros fueran conscientes tanto del progreso como de las dificultades de los demás, facilitando la búsqueda conjunta de soluciones.

Los informes de avance de *sprint* se convirtieron en una documentación muy valiosa sobre la evolución del proyecto, ya que fueron esenciales para mostrar al tutor los avances realizados y obtener una mejor visibilidad del tiempo transcurrido y el que restaba para completar el proyecto. Este análisis permitió detectar una subestimación del esfuerzo total necesario, un punto que será abordado en la siguiente sección.

5.3.2. Exceso de funcionalidades

En la propuesta del proyecto, se buscó incluir una amplia gama de funcionalidades para desarrollar una aplicación móvil que fuera muy abarcativa para diversos usuarios. Sin embargo, este enfoque llevó a comprometerse con la implementación de demasiadas características, lo que generó dificultades en la gestión del tiempo y los recursos.

Para abordar esta situación, se llevaron a cabo reuniones de reajuste con el tutor, donde se utilizó el método MoSCoW (*Must, Should, Could y Wont*) para definir el alcance de cada punto de la propuesta. Las funcionalidades de menor prioridad fueron desarrolladas de manera más superficial mientras que las esenciales recibieron un tratamiento más detallado. Este enfoque permitió mantener la esencia de la aplicación y, al mismo tiempo, cumplir con los plazos de entrega establecidos.

5.3.3. Prematura optimización

Al inicio del desarrollo, se buscaba finalizar cada funcionalidad cumpliendo con todos los detalles estipulados. No obstante, este enfoque generaba tareas demasiado largas, lo que provocaba el agotamiento en los desarrolladores. Además, cualquier cambio en la funcionalidad implicaba un gran retrabajo posterior.

Para solucionar estos problemas, se decidió reducir el nivel de detalle requerido para dar por finalizada una tarea. Los pendientes se registraban en *Jira*, clasificados como DEUDA (si era algo pendiente de hacer) o *BUG* (si era un error en el desarrollo). Estos pendientes eran priorizados y trabajados en los sprints siguientes. Este cambio permitió al equipo avanzar con mayor fluidez en el desarrollo de las tareas y también gestionar mejor los pendientes que surgían.

6. Futuras Líneas de Trabajo

A medida que el desarrollo del proyecto avanza y se consolidan las funcionalidades básicas, surgen nuevas oportunidades y desafíos que pueden enriquecer la experiencia del usuario y mejorar la eficiencia del sistema. En esta sección se exploran las futuras líneas de trabajo que se pueden abordar para ampliar y optimizar el sistema presentado aquí.

- *Diversidad de trivias e impacto en las recompensas:* ofrecer trivias basadas en el contenido visto y su impacto en el sistema de puntos al completarlas. Integrarse con una inteligencia artificial para que estas puedan ser generadas bajo demanda.
- *Sistema de notificaciones:* las notificaciones resultarían de gran valor para algunas funcionalidades de la aplicación. Algunos ejemplos donde se podrían incorporar notificaciones son: avisar si un amigo vio contenido recomendado por el usuario, notificar si contenido de la *watchlist* está disponible en alguna de las plataformas del usuario, o alertar al recibir una solicitud de amistad.
- *Recomendaciones que consideren las plataformas del usuario:* actualmente, el contenido recomendado podría no estar disponible en las plataformas del usuario. Se considera deseable que las recomendaciones estén dentro de los servicios de streaming que el usuario posee, asegurando que tenga una forma sencilla de acceder a este.

- *Sistema C-BOT de guía interactiva inicial por la aplicación:* plantear un sistema que facilite el uso inicial de la aplicación, permitiendo que nuevos usuarios comprendan rápidamente las funcionalidades clave, lo que ayudaría a reducir la curva de aprendizaje y facilitaría la adopción del sistema.
- *Manejo del cold start* [12]: actualmente, las recomendaciones basadas en el contenido visto por el usuario presentan el problema del *cold start*. Se puede incorporar, dentro del tour de *C-BOT*, una encuesta donde se realicen preguntas al usuario con el objetivo de obtener información sobre sus gustos y mitigar este problema.
- *Importación de contenido visto a partir de un archivo:* esto sería útil para que aquellos usuarios que ya disponen de sistemas de seguimiento de contenido visto puedan migrar a *Stream Club* de manera sencilla. Además, facilitaría el desarrollo, permitiendo una configuración más rápida de un entorno de pruebas.
- *Filtrado de contenido en conjunto con la búsqueda:* la búsqueda de contenido y el filtrado *discover* funcionan de forma separada debido a las limitaciones de TMDB. Si se utiliza otro proveedor de información que permita trabajar con filtros de manera más versátil, se podría unificar estas funcionalidades.
- *Soporte a otros países:* detección de la región desde donde se utiliza la aplicación, relevante para determinar la disponibilidad del contenido en las plataformas de *streaming*. Por ejemplo, el catálogo de *Netflix* varía entre Argentina y Estados Unidos, por lo que es importante adaptar la disponibilidad del contenido según la ubicación del usuario.
- *Eliminar cuenta:* ya sea por cumplimiento de normativas, respeto a la privacidad de los usuarios o por promover la transparencia, esta funcionalidad es fundamental de incorporar en un futuro.
- *Métrica de la calidad de recomendaciones:* la incorporación de métricas para evaluar la calidad de las recomendaciones quedó pendiente debido a los problemas expuestos

en la sección de métricas.

7. Conclusiones

Llegando al final del proyecto, la principal conclusión es que se logró cumplir con el objetivo planteado inicialmente: desarrollar una aplicación competitiva y superadora en muchos aspectos dentro del mercado actual. Sin embargo, se puede notar una gran diferencia entre aquello realizado para un trabajo profesional y lo que sería necesario para un entorno productivo.

Entre los puntos más destacados se encuentra la lentitud de los servicios desplegados. Los tiempos de respuesta hacen que la aplicación se vuelva difícil de utilizar, ya que es normal estar esperando más de 2 segundos para que cargue alguna pantalla, tiempo demasiado elevado para los estándares actuales. El uso de la API gratuita de TMDB permitió el desarrollo del proyecto, pero existen alternativas superiores en el mercado que mejorarían significativamente la calidad de la aplicación.

Otro aspecto relevante fue la alta motivación del equipo con lo desarrollado. Esto por más que fuera esencial para el desarrollo, se volvía un problema ya que se pensaban funcionalidades en exceso que luego deberían ser acotadas. El alcance del proyecto inicialmente era demasiado optimista, y tuvo que ser ajustado durante el desarrollo. Hubo un gran trabajo conjunto con el tutor para definir qué funcionalidades eran esenciales y cuáles no, lo que generó múltiples debates, por momentos largos.

Por otro lado, el proyecto fue llevado a cabo por un equipo de tres miembros. Si bien esta cantidad de integrantes permitió el desarrollo de una aplicación completa, también representó un gran esfuerzo en términos de gestión del proyecto. La organización y la comunicación constante dentro del equipo fue esencial para asegurar que las decisiones estuvieran alineadas y contaran con el respaldo de todos los integrantes.

En conclusión, el equipo está satisfecho con lo logrado. En el proceso se enfrentaron a

diversos desafíos que pudieron ser superados, algunos con más sobresaltos que otros. La experiencia de haber desarrollado este proyecto deja valiosas lecciones y muchos conocimientos para el futuro.

8. Referencias

1. "What is Streaming - Definition, Meaning & Explanation". verizon.com. <https://www.verizon.com/articles/internet-essentials/streaming-definition/> (accedido el 26 octubre de 2024).
2. "About - MyAnimeList.net". MyAnimeList.net. <https://myanimelist.net/about.php> (accedido el 26 octubre de 2024).
3. "Frequent questions". Letterboxd. Social film discovery. <https://letterboxd.com/about/faq/> (accedido el 26 octubre de 2024).
4. "JustWatch - About us". JustWatch. <https://www.justwatch.com/ar/quienes-somos> (accedido el 26 octubre de 2024).
5. L. Argelich, N. Golmar, D. Martinelli, M. Ramos Mejía y J. A. Laura, "14. Sistema de Recomendaciones", en Organización de Datos Apunte de la Materia. 2020.
6. L. Argelich, N. Golmar, D. Martinelli, M. Ramos Mejía y J. A. Laura, "17. Redes Sociales", en Organización de Datos Apunte de la Materia. 2020.
7. what-is-aws. (s.f.). [Vídeo]. Amazon Web Services, Inc. <https://aws.amazon.com/es/what-is-aws/> (accedido el 26 octubre de 2024).
8. Kaggle: your machine learning and data science community. <https://www.kaggle.com/> (accedido el 26 octubre de 2024).
9. Introduction to Expo Router. Expo Documentation. <https://docs.expo.dev/router/introduction/> (accedido el 26 octubre de 2024).
10. Read Me · Redux en Español. <https://es.redux.js.org/> (accedido: 26 de octubre 2024).

11. Simões, C. (2020, July 14). MoSCoW. ¿Qué es y cómo priorizar en el desarrollo de tu aplicación? Blog ITDO - Agencia De Desarrollo Web, APPs Y Marketing En Barcelona. <https://www.itdo.com/blog/moscow-que-es-y-como-priorizar-en-el-desarrollo-de-tu-aplicacion/> (accedido: 26 de octubre 2024).
12. ¿Cuáles son los desafíos y las oportunidades de usar el filtrado colaborativo para los usuarios de arranque en frío? (2023, August 25). <https://es.linkedin.com/advice/0/what-challenges-opportunities-using-collaborative?lang=es> (accedido: 26 de octubre 2024).
13. Atlassian. Jira | Software de seguimiento de proyectos e incidencias | Atlassian. <https://www.atlassian.com/es/software/jira> (accedido: 26 de octubre 2024).
14. About GitHub and Git - GitHub Docs. GitHub Docs. <https://docs.github.com/en/get-started/start-your-journey/about-github-and-git> (accedido: 26 de octubre 2024).
15. Free design tool for websites, product design & more | Figma. Figma. <https://www.figma.com/design/> (accedido: 26 de octubre 2024).
16. What is Postman? Postman API Platform. Postman API Platform. <https://www.postman.com/product/what-is-postman/> (accedido: 26 de octubre 2024).
17. Get Started with React Native · React Native. (2024, October 23). <https://reactnative.dev/docs/environment-setup> (accedido: 26 de octubre 2024).
18. Google Colab. Colab.google. colab.google. <https://colab.google/> (accedido: 26 de octubre 2024).
19. Neon Serverless Postgres — Ship faster. Neon. <https://neon.tech/> (accedido: 26 de octubre 2024).

20. Cloud Application Platform | Render. <https://render.com/> (accedido: 26 de octubre 2024).
21. “Home.” (2024, September 19). Docker Documentation. <https://docs.docker.com/> (accedido: 26 de octubre 2024).
22. MongoDB: the Developer Data platform. MongoDB. <https://www.mongodb.com/> (accedido: 26 de octubre 2024).
23. Expo documentation. Expo Documentation. <https://docs.expo.dev/> (accedido: 26 de octubre 2024).
24. Getting Started. The Movie Database. <https://developer.themoviedb.org/reference/intro/getting-started> (accedido: 26 de octubre 2024).
- 25.Codecov. (2024, September 10). CodeCOV - the leading code coverage solution. <https://about.codecov.io/> (accedido: 26 de octubre 2024).
26. Amazon VPC features. Amazon Web Services, Inc. <https://aws.amazon.com/vpc/features/> (accedido: 26 de octubre 2024).
27. Kinsta. (2022, December 19). ¿Qué Es el Web Scraping? Cómo Extraer Legalmente el Contenido de la Web. Kinsta®. <https://kinsta.com/es/base-de-conocimiento/que-es-web-scraping/> (accedido: 26 de octubre 2024).
28. One hot encoding | Interactive Chaos. <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/one-hot-encoding> (accedido: 26 de octubre 2024).
29. scikit-learn: machine learning in Python — scikit-learn 1.5.2 documentation. (n.d.). <https://scikit-learn.org/stable/> (accedido: 26 de octubre 2024).
30. What is k-Nearest Neighbor (kNN)? | A Comprehensive k-Nearest Neighbor Guide. (n.d.). Elastic. <https://www.elastic.co/what-is/knn> (accedido: 26 de octubre 2024).

31. Home | Scrum Guides. (n.d.). <https://scrumguides.org/> (accedido: 26 de octubre 2024).
32. “¿Qué es orm - object relational mapping? - el blog De Ensalza (2023) el blog donde las ideas toman forma“. <https://www.ensalza.com/blog/diccionario/que-es-orm-object-relational-mapping/> (accedido: 26 de octubre 2024).

9. Anexos

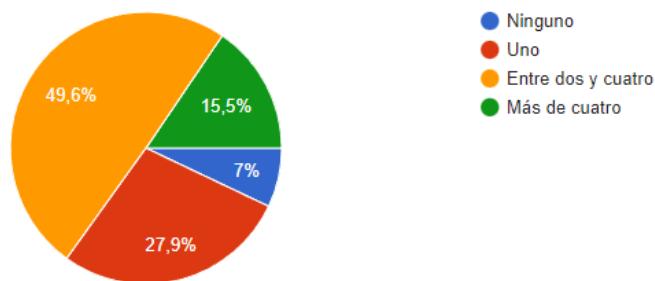
9.1. Encuesta

Para fundamentar la necesidad de desarrollar Stream Club, se llevó a cabo la siguiente encuesta, compuesta de seis preguntas:

■ Pregunta 1

¿Con cuántos servicios de streaming para ver películas y series contás? (estilo: Netflix, HBO Max, Disney +, etc.)

355 respuestas



■ Pregunta 2

¿Con qué frecuencia utilizas cada una de las plataformas de streaming que tenés?

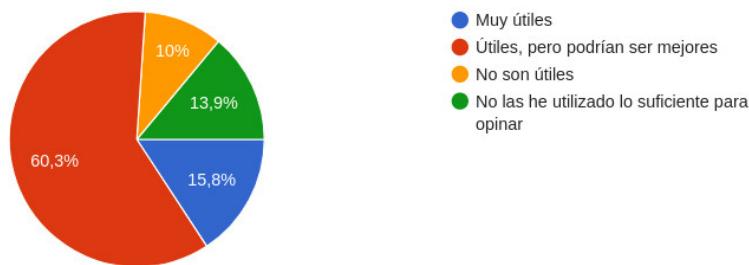
231 respuestas



■ Pregunta 3

¿Cuál es tu opinión general sobre las recomendaciones que te ofrecen las plataformas de streaming?

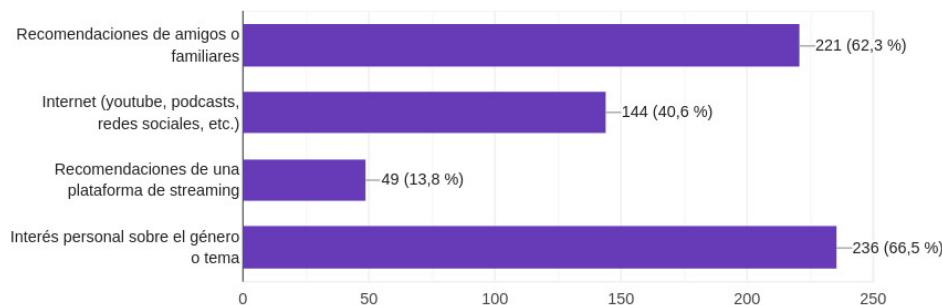
330 respuestas



■ Pregunta 4

¿Cuáles son tus principales motivos para ver una serie o película? (marque todas las que correspondan)

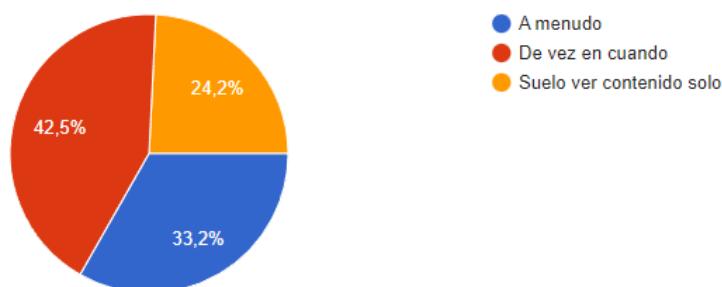
355 respuestas



■ Pregunta 5

¿Solés ver contenido con amigos o familiares?

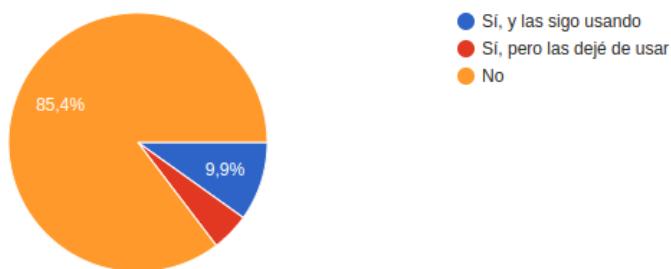
355 respuestas



- Pregunta 6

¿Has utilizado aplicaciones de seguimiento de películas y series? (como Letterboxd, JustWatch, MyAnimeList, etc.)

355 respuestas



9.2. Repositorios

- *mobile-app*: <https://github.com/StreamClub/mobile-app>
- *user-api*: <https://github.com/StreamClub/user-api>
- *content-api*: <https://github.com/StreamClub/content-api>
- *recommendations-api*: <https://github.com/StreamClub/recommendations-api>
- *process-api*: <https://github.com/StreamClub/process-api>
- *content-script*: <https://github.com/StreamClub/content-script>