Escuela Politécnica Superior

Trending Series

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Roberto Cánovas Carratalá

Tutor/es:
Domingo Gallardo López

Universitat d'Alacant
Universidad de Alicante

Febrero 2018

Contexto

La idea de este proyecto surgió de la observación del hecho de no poder consultar en cualquier momento de qué series se está hablando más en forma de ranking. Se puede consultar listados que ofrecen medios especializados que publican cada cierto tiempo, y que además obtienen de fuentes de costoso acceso como compañías de medición de consumo global de contenido, por ejemplo, Amobee Inc[1].

Así que, en un principio la idea era centrarse en la búsqueda de una fuente de libre acceso e inmediato de la que se pudiera, al menos, hacer un cálculo estimado reducido de las series de las que más se está hablando, haciendo una pequeña medición cada un determinado número de horas.

Se encontró una, Twitter. Si se obtenía una determinada cantidad de tweets y se calculaba cuántos de cada serie se estaban publicando por unidad de tiempo, se podría hacer un ranking de series que fuera actualizándose constantemente.

Solo faltaba decidir de dónde obtener los datos de las series para mostrarlos en el ranking (nombre, imágenes, etc.). En la búsqueda del servicio perfecto y más completo apareció otro problema, había tantas opciones que ofrecían distintos datos y que se complementaban entre ellas que lo mejor sería acceder a varias de ellas.

Lo que llevó a la siguiente decisión: crear una API propia de libre acceso la cual se vaya nutriendo de las API que se pensaba utilizar, a la que todo el mundo pueda acceder para obtener información sobre las series, temporadas, capítulos, actores, etc., sin restricciones y con opciones de votación y seguimiento.

El ranking de series parecía quedar a un lado y fue entonces cuando, debido a querer incluir esta y más características, se tomó la decisión de implementar un cliente que consumiera los datos de nuestra propia API para plataformas móviles (iOS y Android) en la que se pudiera incorporar la idea del ranking de series entre otras.

Por lo que, en resumidas cuentas, se buscaba crear un sistema que, a petición de los usuarios, fuera creciendo y creciendo, que expusiera una API libre en la que ofrecer todos estos recursos y un cliente para móvil en el cual consumirlos.

Para saber cómo se construyó esta idea y qué más se fue añadiendo por el camino, espero que disfrute leyendo esta memoria y que sea de su agrado.

[1] Amobee Inc, global marketing technology, sitio oficial

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, Domingo Gallardo López, por haberme guiado en todo momento en la dirección correcta, por haber aceptado ser mi tutor en este proyecto y, sobre todo, por haber tenido la suerte de poder asistir a sus clases las cuales fueron una fuente de motivación y entusiasmo sin igual.

“Hay que creer en la recursión”

También a mi familia y a mi pareja, por su incondicional apoyo, por creer en todo momento en mí y por encima de todo, por aguantar mis explicaciones tan aburridas durante todos estos largos años, con los que siempre estaré en deuda.

Por último, quería agradecer a la Universidad de Alicante, a la Escuela Politécnica Superior, a los docentes que me han impartido y a los compañeros que he conocido, donde y con los que he tenido la oportunidad de vivir unos años en los que hemos luchado por conseguir uno de nuestros sueños.

Gracias.

“Those who can imagine anything, can create the imposible.”

– Alan Turing

Índice

Índice de ilustraciones

1. Introducción

Como se ha explicado en el contexto, se propone desarrollar una API de la cual poder consumir datos de series de televisión y, además, desarrollar un doble cliente móvil para iOS y Android que sirva de cliente de los recursos que ofrece la API.

La API se nutrirá de otras API de libre acceso, obteniendo de ellas la información necesaria de las series que los usuarios van solicitando, creciendo en principio sin límite. Los usuarios podrán acceder tanto a la API como a los clientes donde ver y pedir series, marcar series como favoritas, puntuar series, ver información sobre las temporadas y capítulos de las series, ver información sobre los actores de las series… Sin olvidarnos de la inspiración inicial de un ranking de series basado en la popularidad, que se vería implementado de igual manera.

La idea principal es la "unificación" de otras API que se consideran incompletas para crear una completa en español y a la que todo el mundo pueda acceder, ya sea a través de la API, a través de los clientes móviles, o a través de cualquier cliente que cualquiera se atreva a implementar.

En resumen, se propone forjar la API *Única* que coja lo mejor de las demás, o como a mí me gusta llamarla: una API para gobernarlas a todas[2].

Para ver cómo se va a realizar, de qué APIs se va a nutrir y con qué tecnologías se va a implementar todo, se necesita un *estado del arte* que ya se ha ido pincelando entre el contexto y esta introducción.

[2] Referencia al «Anillo Único». Tolkien, J. R. R. *El Señor de los Anillos.*

2. Estado del arte

Después de pensar en lo que se quería desarrollar más concretamente, es el momento de la investigación y la búsqueda de soluciones semejantes o iguales directamente.

2.1. API de series de televisión

En cuanto a la idea de una API que ofrece datos de series de televisión, tenemos varias opciones.

2.1.1. The TVDB API[3]

Esta es la API que más llama la atención y que después de toda la investigación se ha concluido como la más completa e interesante. Se ha encontrado todas las series que se han buscado, incluyendo *Farmacia de Guardia*[4].

Ilustración 1 - TheTVDB

Contiene datos de las series, episodios, imágenes con votaciones, temporadas y episodios, actores con imagen, etc., la cual la convierte en una interesante fuente principal de la nueva API. Interesante el hecho de que ofrezca las imágenes en distintos idiomas aparte de la información de las series, incluyendo el español.

2.1.2. OMDb API[5]

Esta API comenzó siendo muy prometedora al inicio de este proyecto, ya que ofrece mucha información sobre las series con una simple búsqueda o petición, pero durante el transcurso del proyecto se ha hecho de pago parcialmente y no acaban por incluir el idioma español entre los datos que ofrece.

2.1.3. The Movie Database API[6]

En este caso, pese a llamarse *La Base de datos de Películas*, ofrece también datos de series de televisión, así como de sus temporadas, episodios, actores, directores, imágenes, etc.

THE MOVIE DB

Buena organización y datos en español, candidata para extraer los datos referentes a las temporadas, episodios y gente para complementar los que se obtendría de The TVDB API la cual, en estos recursos, resulta un poco engorroso obtenerlos. Imágenes por idioma, además de integrar el `id` de The TVDB API en la búsqueda de series, lo cual viene bien para relacionarlas.

[3] API de series de televisión, The TVDB API
[4] Serie de televisión, *Farmacia de Guardia*, 1991, en thetvdb.com
[5] API de series de televisión, OMDb API
[6] API de películas y series de televisión, The Movie Database API

2.1.4. Trakt.tv API[7]

A pesar de tener una presentación impresionante y responder muy bien, la información que ofrece de cada recurso resulta escasa, solo en inglés y no ofrece imágenes directamente desde su API, si no desde las API que se han mencionado.

2.1.5. Episodate API[8]

Sencilla API que además de ofrecer información sobre las series (no podemos especificar temporada, capítulo, etc.) ofrece también un `endpoint` con las series más populares del momento, lo cual podría interesar para nuestro ranking de series más populares.

2.1.6. SICK beard API[9]

Esta API parece haber cesado su funcionamiento recientemente. Su organización no es demasiado buena ni amigable para el usuario o incluso el desarrollador. Sus ejemplos de respuesta y el uso del `id` de The TVDB API delata que era una simple pasarela a esta última API.

Como podemos apreciar, hay una gran cantidad de proyectos que ofrecen datos sobre series de televisión, mejores y peores, con más datos o menos, de la cuales se puede seleccionar unas pocas elegidas para diseñar y nutrir la nueva API de forma que se convierta en una API completa que sirva para cumplir nuestro objetivo.

2.2. Cliente móvil de series de televisión

Como clientes móviles de series de televisión se puede encontrar varios proyectos muy conocidos e indiscutiblemente buenos ejemplos.

2.2.1. IMDb Cine y TV[10]

Aplicación móvil de la red de películas y series de televisión más importante de todo Internet. Cuenta con una base de datos inmensa y es la referencia número uno. En cuanto a lo que interesa para este proyecto, permite ver la información de las series, temporadas, episodios, actores, etc., puntuar series, seguirlas, noticias y mucho más.

Diseño oscuro, aunque peca de demasiada información y demasiados elementos en una sola vista, llegando a saturar un poco la pantalla.

Colores de resalto amarillos, hace buen contraste con el fondo oscuro.

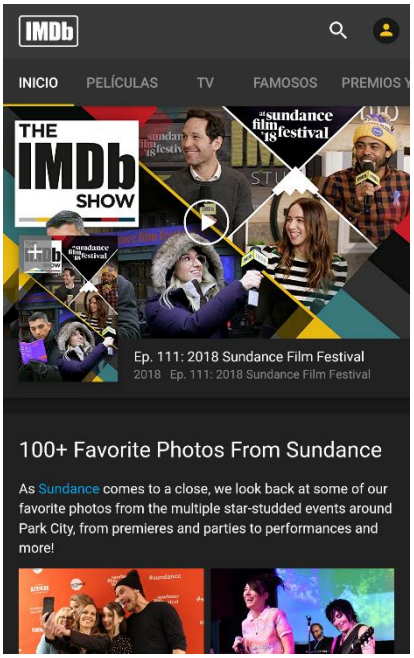


Ilustración 2 - IMDb: aplicación móvil

[7] API de películas y series de televisión, Trakt.tv API
[8] API de series de televisión, Episodate API
[9] API de series de televisión, SICK beard API
[10] Aplicación móvil de la famosa IMDb

2.2.2. SensaCine – Cine y Series[11]

Aplicación más sencilla que la anterior, pero con buena implementación. Ofrece información sobre las series, temporadas y episodios, pero de forma muy reducida. Permite el seguimiento de series, comentarios en las series e incluye vídeos de trailers de cada serie, temporada y capítulo si están disponibles.

2.2.3. Plex[12]

Aunque es una aplicación que organiza tu biblioteca personal, hace un buen uso de los datos de las series para organizarlas, sirviéndose de APIs como The TVDB API.

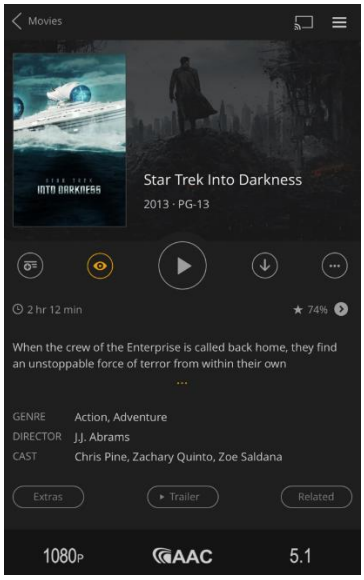


Ilustración 3 - Plex: aplicación móvil

De esta aplicación se puede obtener referencias claras de un buen diseño para aplicaciones de este tipo, ya que distribuye muy bien el contenido y lo divide en secciones claras.

Diseño de nuevo oscuro que favorece a las imágenes y da sensación de estar en una sala oscura como la de un cine.

Colores de resalto amarillos, como en IMDb, haciendo buen contraste con los fondos oscuros.

2.2.4. Netflix

Parecido al caso anterior, aplicación de streaming de series.

Diseño oscuro una vez más, resalto claro de imágenes y sensación de estar en un cine para una mejor inmersión.

Colores de resalto blancos y rojos.

En este punto del estado del arte, se va dando la repetición de diseños oscuros y colores de resalto blancos, rojos y amarillos.

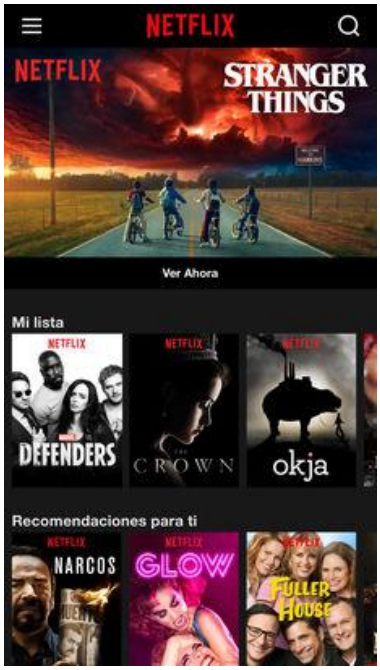


Ilustración 4 - Netflix: aplicación móvil

[11] Aplicación móvil del portal SensaCine
[12] Aplicación móvil de Plex

2.2.5. TV Time[13]

Con esta aplicación se puede llevar un seguimiento de las series que a uno le interesan, avisando al usuario de cuándo van a emitir un nuevo capítulo de las series que sigue, además comentar series, pero ofrece poca información sobre los recursos.

2.3. Reflexiones

Los patrones encontrados generalmente y las pautas a seguir son:

- Diseño oscuro
- Resaltos de contraste fuerte como blanco, rojo o amarillo
- Listados horizontales
- Imprescindible el uso de imágenes para destacar
- No sobresaturar las vistas de componentes e información
- Interfaz amigable, interactiva, reactiva e intuitiva
- Mínimo de pulsaciones en pantallas para encontrar lo que se busca

Hay diversas aplicaciones para móviles según las necesidades, aunque la referencia principal es el caso de IMDb. En base a esto, se debe decidir qué características tendrá la nueva aplicación y con qué tecnologías se implementará. Así que es hora ya de ir fijando unos objetivos claros.

[13] Aplicación móvil de TV TIME

3. Objetivos

Llegados a este punto y conociendo el problema expuesto, habiendo estudiado cómo está el mercado de APIs y clientes móviles y a lo que nos enfrentamos, es el momento de establecer unos objetivos claros en los que se basará el proyecto a partir de aquí en adelante.

3.1. API REST

En primer lugar, crear una aplicación API REST que se encargue de ofrecer los recursos consumibles ya comentados.

Gracias a la realización de la API REST, se separa el `back-end` de cualquier `front-end`, y de este modo, cualquiera puede consumir nuestra API sin necesidad de un cliente, o que cualquiera pueda implementar su propio cliente, ofreciendo una escalabilidad y flexibilidad muy interesante. Cuantos más usuarios usen la nueva API, más grande será y más recursos podrá ofrecer.

Además de ofrecer los recursos más que mencionados, un usuario podrá registrarse e identificarse, para poder hacer uso de la API, solicitar series, hacer seguimiento de las series que quiera, de los capítulos, votar series, incrementar su popularidad…

3.2. Cliente para Smartphones

En segundo lugar, crear un doble cliente para Smartphones (iOS y Android) que consuma dicha API. Desde este cliente, aparte de consumir los recursos que ofrecerá la API, un usuario podrá solicitar series que no estén en nuestro sistema.

Esta aplicación implementará todos los `endpoints` de la API de los que pueda hacerse uso, con el objetivo de poder ver información de series, temporadas y episodios, puntuar series, seguir series, marcar episodios como vistos, solicitar series que no estén en el sistema…

Para solicitar una serie que no esté en el sistema, bastará con que el usuario busque por nombre la serie que el sistema no tiene, y la API buscará en las APIs externas series que coincidan con la búsqueda, y entonces el usuario podrá elegir qué serie es la que desea solicitar de entre las encontradas, quedando a la espera de revisión por un administrador, todo de manera automática sin introducir datos de ningún tipo aparte del nombre.

3.3. Administración

En tercer lugar, crear un `front-end` para la parte de administración `responsive`[14], donde la característica más importante será la de revisar las peticiones que los usuarios hayan realizado y aceptarlas o denegarlas, además de poder consultar todos los recursos y realizar acciones sobre los mismos.

Es importante la implementación de esta, pues cuando el sistema se lance por primera vez, no contendrá serie alguna, y serán los usuarios los encargados de solicitar las series que deseen. El administrador podrá ver las peticiones de series nuevas, de modo que, si acepta una petición, el sistema descargará todos los datos de serie y la incorporará a la API de manera automática. Así, el

[14] Diseño web adaptable

sistema irá creciendo a medida que los usuarios vayan solicitando sus series favoritas y cuantos más usuarios haya, mayor será la cobertura de series ofrecidas en menor tiempo.

Desde esta parte de administración, un administrador también podrá redescargar los distintos tipos de información que ofrece una serie, por si hay cualquier error, o se debe actualizar la información, de modo que se podrá volver a descargar tanto la información general de una serie, sus imágenes principales, sus temporadas, sus episodios, etc.

4. Tecnologías

En este apartado se hablará de las tecnologías más importantes que van a sustentar las distintas partes del proyecto, las cuales son: la API, la base de datos y gestión de la misma, la parte de administración, y por último el cliente para smartphones.

4.1. Play Framework

Play[15] es un framework de aplicaciones web de Java y Scala de alta productividad que integra los componentes y las APIs que se necesita para el desarrollo moderno de aplicaciones web, open-source[16] y con bastante apoyo de la comunidad. Está basado en una arquitectura web-friendly, sin estado (stateless), y muy ligera que se caracteriza por tener un consumo predecible y mínimo de los recursos, como de la CPU, memoria, hilos…, ideal para aplicaciones muy escalables gracias a su modelo Reactive[17] basado en Akka Streams[18].

Ilustración 5 - Play Framework

Tiene soporte nativo para Java y Scala, de manera que podemos implementar todo lo que necesitemos en uno u otro, o en ambos. Es RESTful de serie, por lo que es un buen punto de partida para cualquier API REST.

Por ello, es ideal para la **API** que se quiere desarrollar aquí, ya que en un principio se comenzará con las pocas series de los pocos usuarios iniciales que se tenga, pero que con el tiempo irá creciendo exponencialmente lo cual generará una cantidad de datos de miles de series muy grande.

Este framework sigue la arquitectura MVC (Modelo, Vista, Controlador), pero es flexible en cuanto a esto, de tal modo que podemos no utilizar vistas para **la API, devolviendo directamente JSON,** y para **la parte de administración utilizar las vistas de Play.**

Play ofrece semillas de inicio de proyecto, teniendo desde el primer momento un proyecto inicial y limpio del que partir el cual es muy sencillo de utilizar, puesto que para implementar una acción tan solo hay que implementarla en un controlador (donde se puede obtener toda la información de la petición, y generar todo tipo de respuestas), y llamarla desde la ruta que escribamos en su fichero routes.

Más adelante, en el apartado Arquitectura, se verá más en profundidad.

[15] Play Framework – Sitio oficial
[16] playframework/playframework – Repositorio en GitHub de Play Framework
[17] Reactive – Manifiesto Reactive
[18] Akka Streams Introduction – Introducción a Streams de Akka

4.2. MySQL

Más que conocidísimo sistema de gestión de bases de datos relacionales de Oracle, open-source[19], con veintidós años de trayectoria y muchísimo soporte de la comunidad.

Ilustración 6 - MySQL Logo

Como su nombre bien indica, basado en SQL, usado y diseñado para la gestión de los datos que mueven los sistemas de bases de datos relacionales. Durante el transcurso de todo el grado, es el lenguaje referente a bases de datos que más se ha visto y practicado.

Además, MySQL es el sistema más ofrecido por los servicios de bases de datos en internet y de hosting, por no hablar de las empresas y corporaciones que lo utilizan como Google, PayPal, Facebook, GitHub, etc.

Por lo anterior ha sido elegido para formar parte de este proyecto.

4.3. JPA

Esta API llamada JPA[20], desarrollada por Sun Microsystems (ahora Oracle), es una API de persistencia y un framework que maneja datos relacionales en aplicaciones Java SE y Java EE.

El objetivo de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos, haciendo un mapeo objeto-relacional para permitir usar objetos regulares conocidos como POJOs[21].

Gracias a esta API, se podrá realizar peticiones de objetos directamente a la base de datos y beneficiarnos de otras características como la de obtener un objeto y modificar sus atributos y que automáticamente JPA se encargue de hacer las acciones pertinentes sobre la base de datos para actualizarla.

Para implementar las consultas en Java, esta API utiliza JPQL[22] como lenguaje de consulta de persistencia en Java, muy parecido a la sintaxis de SQL que actúa sobre los objetos de JPA.

[19] mysql/mysql-server – Repositorio de MySQL Server
[20] JPA – Java Persistence API
[21] POJO – Plain Old Java Object
[22] JPQL – Java Persistence Query Language

4.4. React Native

React Native[23] es una librería basada en React[24] centrada en construir interfaces de usuario. Parte de la base de React en cuanto a ser declarativo y estar basado en los componentes que mantienen su propio estado.

Ilustración 7 - React Native logo

Usando tan solo Javascript, se puede diseñar interfaces interactivas con poco esfuerzo, ya que la librería se encarga de actualizar las vistas y los componentes de manera muy eficiente cuando los datos cambian.

Lo más interesante de una aplicación React Native es que es realmente una aplicación móvil, no una simple aplicación web móvil, ni una HTML5 app ni una aplicación híbrida, sino que se construye una aplicación móvil indistinguible de una aplicación construida en Objective-C o en Java. React Native usa la misma norma fundamental de construcción de interfaces que una aplicación iOS y Android normal, simplemente usando Javascript y React juntamos los componentes y listo.

Así pues, usando simplemente un componente de React Native, se hace uso del componente nativo tanto de iOS como de Android fácilmente.

También dispone de ciertas características y ventajas que lo hacen ideal para el desarrollo como el Hot Reloading, que permite implementar nuevo código o modificar el existente y recargar la aplicación sin tener que recompilarla ni perder el estado actual tanto de la aplicación como de cada uno de los componentes, viendo la modificación instantáneamente en la aplicación en ejecución.

Es posible utilizar código nativo de cada una de las plataformas si fuera necesario, combinando apaciblemente componentes escritos en Objective-C, Java o Swift. Gracias a esto, no solo nos limitamos a los componentes que vienen incluidos en React Native, sino que podremos crear componentes propios para lo que se necesite usando JavaScript, React Native, React y/o código nativo y usarlos donde se requiera. De hecho, la comunidad es una fuente enorme de implementaciones de componentes muy útiles y cada vez tiene más soporte.

Por lo que la idea es que usando este framework, se cree un fichero por cada vista que incluya la lógica necesaria (si es que se necesita) que pida y reciba los datos necesarios de la API que se implementa en Play Framework.

Al igual que con Play, en el apartado Arquitectura, se verá esto en más en profundidad.

[23] React Native – Un framework para construir aplicaciones nativas usando React
[24] React – Una librería JavaScript para construir interfaces de usuario

5. Metodología

Debido a que el equipo de desarrollo cuenta solo con un miembro se ha optado por una metodología ágil que combina características de metodologías ágiles, como lo son Scrum[25] y Kanban[26].

5.1. Scrum y Kanban

Se ha hecho una combinación de las metodologías Scrum y Kanban. Se ha realizado una serie de sprints para el desarrollo del proyecto y unas reuniones semanales, que en este caso han sido cada dos o tres semanas con el tutor. Cada vez que se hacía una reunión, se pensaba en cómo solucionar cualquier problema que quedara pendiente y en cómo y con qué seguir.

Cada reunión se aprovechaba para realizar varias a la vez, las cuales eran una pequeña demo de lo implementado hasta el momento y las mejoras, una retrospectiva hablando de los problemas y dificultades durante esas dos semanas de trabajo y finalmente, la planificación del sprint de las siguientes dos o tres semanas.

De Kanban se ha optado por organizar el proyecto en tarjetas y una serie de pequeñas reglas, como por ejemplo la de que una parte defectuosa no debe pasar al proceso siguiente o no iniciar una tarea hasta que la anterior haya sido finalizada.

5.2. Flujo de trabajo: GitFlow

Como modelo de flujo se ha elegido GitFlow[27] el cual es un modelo de Git basando en ramas, ideal para la forma de trabajo y ritmo de implementación que se quería para este proyecto y que encaja muy bien con la combinación de Scrum y Kanban que se ha descrito en el punto anterior.

Sí es cierto que, en este caso, determinados beneficios de este flujo de trabajo no se pueden aprovechar, como el desarrollo en paralelo o la colaboración, pero otros beneficios como la rápida incorporación de características terminadas al proyecto que pueden ser lanzadas en cualquier momento o el soporte para la rápida implementación de corrección de errores, sí.

Explicado esto, se ha elegido el modelo de dos ramas principales llamadas **master** y **develop**:

- master: contiene el código en producción y la release actual
- develop: contiene el código actual del desarrollo, con la última característica/ticket implementada

Además, se hace uso de unas determinadas subramas llamadas **ticket**, **fix**/**hotfix** y **release**:

- ticketN.M: contiene el código de una parte de una nueva característica que se está implementando. Salen y se unen a develop
- fix o hotfix: contiene el código de correcciones o correcciones de emergencia. Los fix salen y se unen a develop y los hotfix salen y se unen a master
- releaseN: contiene el código a publicar en producción. Salen de develop y se unen a master

[25] Scrum – Metodología ágil de desarrollo de software, libro gratuito *Scrum Manager*
[26] Kanban – Marco de trabajo para desarrollo basado en tareas visuales, artículo de Atlassian
[27] GitFlow – Flujo de trabajo de estrategia de ramas y releases, *A successful Git branching model* por Vicent Driessen

De esta manera, cuando se vaya a desarrollar un nuevo ticket de una característica, se abre una rama desde develop y se implementa lo necesario. Una vez implementado y testeado, se integra en develop y si pasan los tests, el ticket queda finalizado.

Una vez todos los tickets, incluyendo un ticket final de refactorización, revisión de código y actualización de la documentación terminados, se da la característica como completada y se pasaría a la siguiente.

5.3. Tablero: Trello

Como tablero y organización principal del proyecto se ha utilizado Trello, donde se han impuesto una serie de listas en las que se ha ido colocando las distintas características, historias de usuario y tickets o tareas de las que se ha hablado antes, a implementar. Gracias a las características de Trello y haciendo buen uso de ellas, se puede hacer un seguimiento muy detallado del estado actual del proyecto.

5.3.1. Listas

Así pues, las listas principales en el tablero son:

- Features: características del proyecto aún no desarrolladas
- To Do: características del proyecto elegidas para el sprint actual a realizar
- In progress: tickets de las características del proyecto que se están implementando
- Integration: tickets que se están testeando en el sistema de integración continua
- Testing: tickets que se están testeando en local
- Done: tickets que se han terminado de implementar y testear
- Features done: características terminadas del sprint actual

La planificación del trabajo es muy sencilla, una vez se ha seleccionado las características que se deben implementar, se dividen en tickets pequeños en forma de tarjeta que será lo que se tenga que implementar uno a uno.

Al final del proyecto se ha generado un total de **155 tarjetas de Trello**.

Una vez implementado un ticket, pasa a integración y una vez pasa la integración, pasa a ser testeados localmente y finalmente se aprueban como terminados.

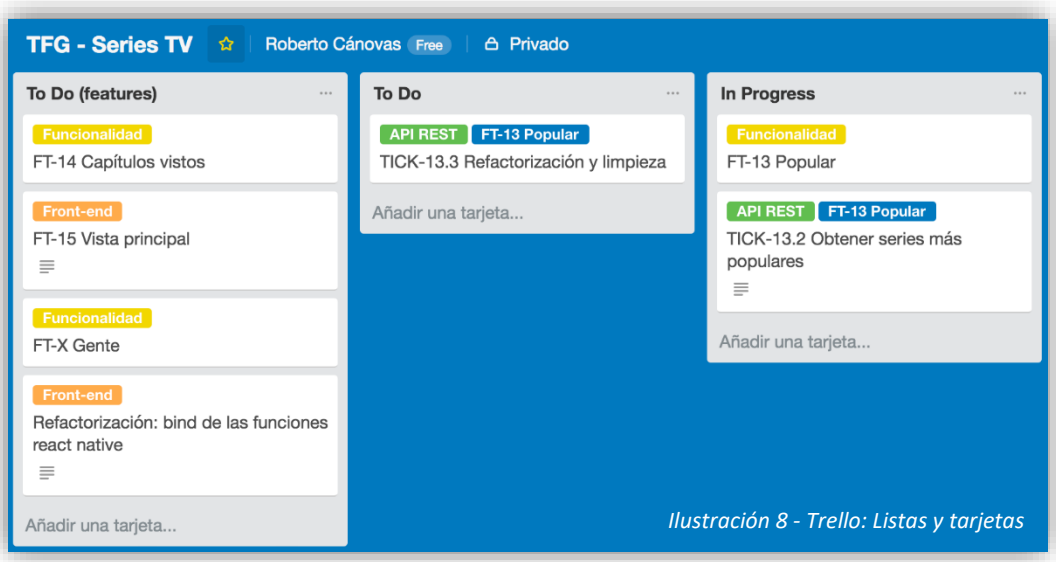


Ilustración 8 - Trello: Listas y tarjetas

Como se puede apreciar en la anterior imagen, en Trello podemos etiquetar cada tarjeta con colores y títulos; de esta forma, es muy fácil identificar las funcionalidades, a qué funcionalidad pertenece cada ticket, y qué parte del proyecto es afectada por cada ticket (API, Front-end, Administración, etc.).

5.3.2. Tarjetas

Estas tarjetas contendrán las features y tickets que se irán implementando a lo largo de todo el proyecto. Contienen información relevante que nos servirá para identificar el estado actual en cualquier momento.

En esta imagen podemos ver la lista de tickets terminados.

Vemos cómo de un pequeño vistazo, en una lista podemos ver fácilmente de qué trata cada ticket.

Por ejemplo, se observa un ticket del front-end para smartphones, donde en la vista de una temporada se ha implementado que se muestren también los episodios, mostrando incluso su boceto en el ticket, y que es perteneciente a la Feature 12 Episodios.

Ilustración 9 - Trello: Tickets finalizados

También se observa al pie de cada ticket ciertos iconos y números.

Ilustración 10 - Trello: pie tarjeta

- El primer icono indica que hay una descripción dentro del ticket, se ha utilizado esta descripción para, en caso necesario, describir lo que se va a implementar o ciertos objetivos del ticket.
- El segundo icono indica el número de archivos adjuntos al ticket, en este caso son dos imágenes de los bocetos.
- El tercer icono indica cuántas asignaciones con GitHub tiene y en este caso es su Pull request.
- El cuarto icono, indica que se trata de un Pull request, y que además ha sido combinado.

Si abrimos dicha tarjeta, podemos ver más información junto a la descripción, imágenes, enlaces de GitHub, información del Pull request, etc., así como el historial de cambios que ha tenido la tarjeta (modificaciones, de qué lista a qué lista ha cambiado, etc.).

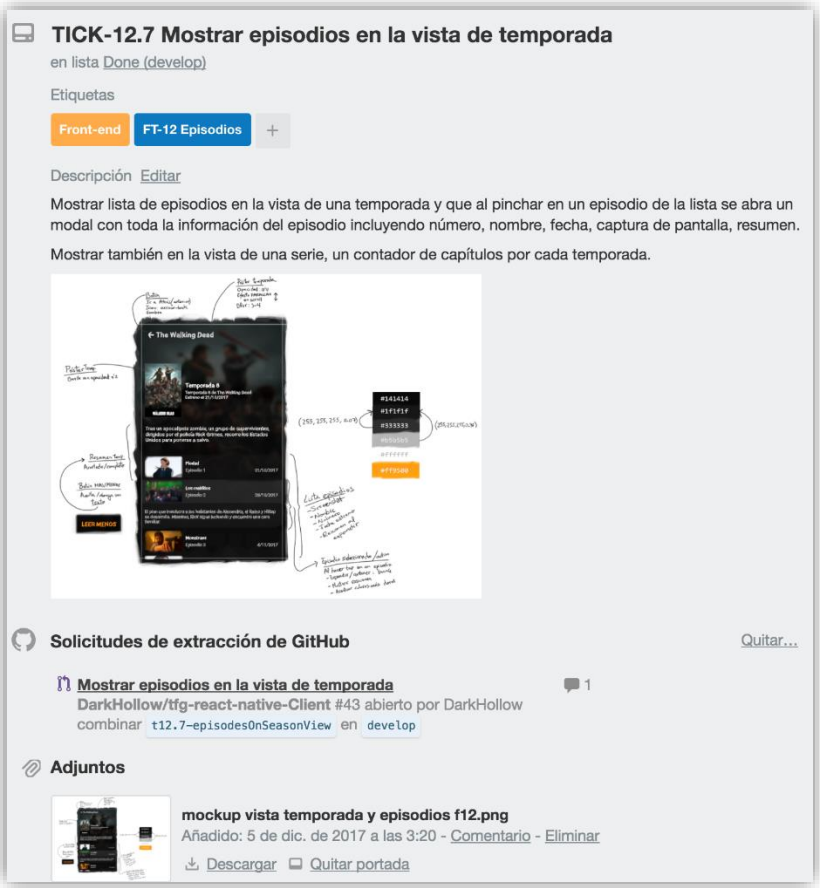


Ilustración 11 - Trello: detalle de tarjeta

Las tareas o tarjetas, también pueden contener listas de checkbox, asignarles prioridad, e incluso asignarles un usuario y fecha de vencimiento, pero en la mayoría de los casos no se ha establecido tales características al tratarse de un equipo de desarrollo de un solo miembro.

5.3.3. Power-Ups

Trello dispone de power-ups los cuales son una manera de conectar tus tableros con los servicios que utilizas. En este caso, se ha hecho uso del de GitHub.

Con este power-up se puede adjuntar Pull requests, ramas, vínculos o issues de GitHub a cualquier tarjeta. No solo eso, sino que en las tarjetas también se mostrarán un check verde del servicio de integración continua si ha pasado los tests o una cruz roja si no en las tarjetas:

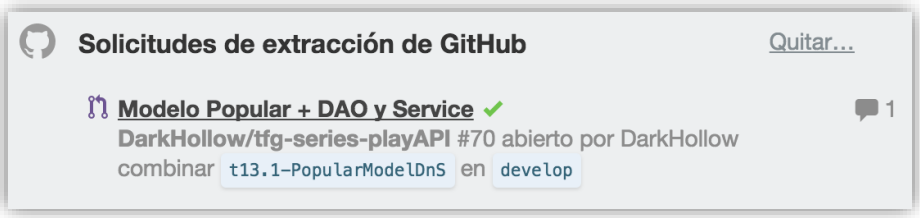


Ilustración 12 - Trello: GitHub Power-Up

5.4. Control de versiones e integración continua: GitHub

Ya solo queda hablar del control de versiones donde se ha seguido esta metodología, haciendo uso de GitFlow y asignando los Pull requests a las tarjetas del tablero de Trello.

GitHub es el control de versiones remoto más conocido y es compatible con el flujo de trabajo que se ha descrito anteriormente en cuanto a la manera de llevar las ramas y pull requests. Al ser solo un miembro en el equipo, no hay confusiones ni problemas de ningún tipo, ni de merges ni de pull requests.

5.4.1. Repositorios

Gracias a GitHub podemos llevar un control bastante completo sobre los repositorios: podemos gestionar y ver las ramas, los commits, los pull request, etc., y ver estadísticas y gráficos bastante interesantes, como el mapa de calor de nuestra actividad, el tráfico, la frecuencia de código o la gráfica de network, en la que podemos ver de manera gráfica las ramas:

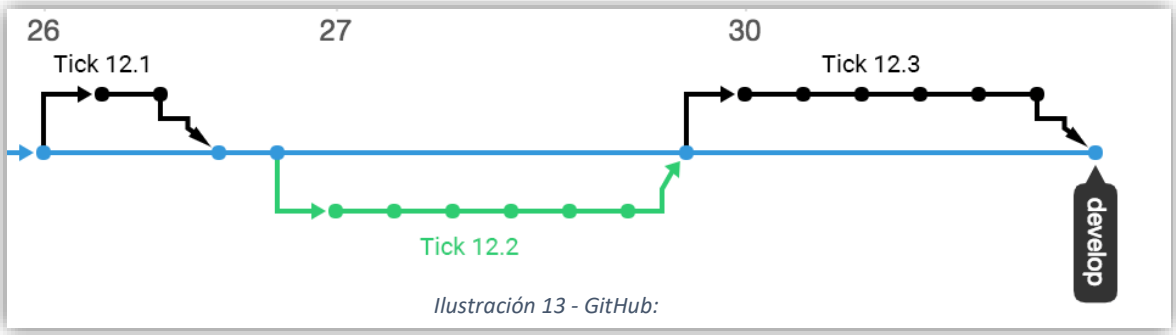


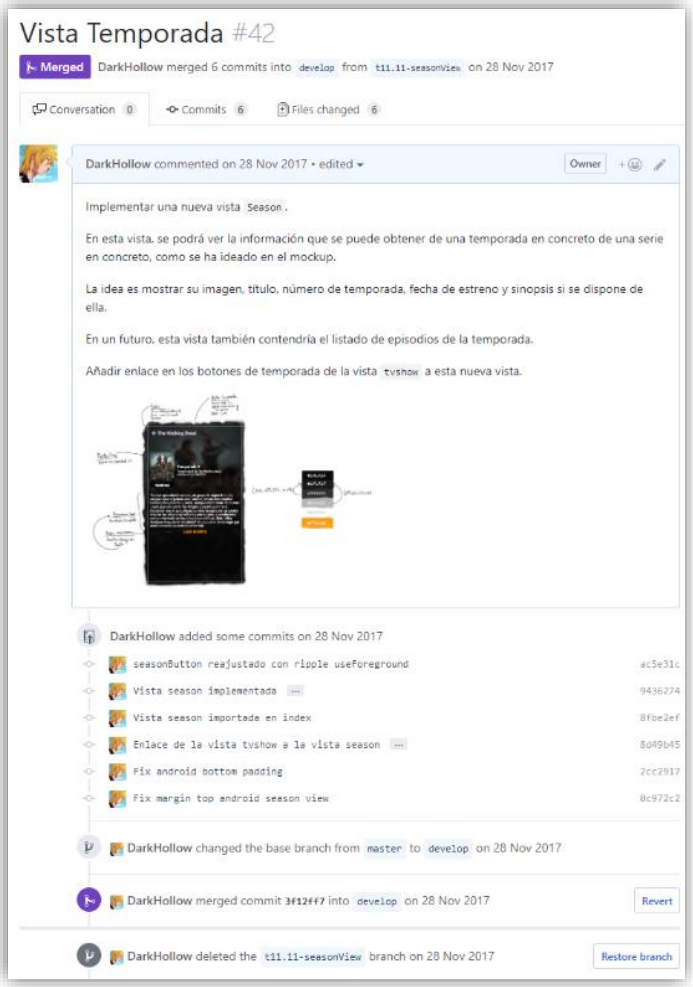
Ilustración 13 - GitHub:

Se puede observar claramente la rama develop en azul, y subramas en negro y verde que representan los tickets de la feature 12.

5.4.2. Pull request

Cada vez que se comienza una tarea o ticket, se realiza alguna refactorización inicial y se genera directamente un Pull request en GitHub. Una vez generado, en Trello se asocia su tarjeta con el Pull request gracias al power-up anteriormente nombrado. Al hacer esto, se puede revisar todo el historial de cambios, commits, tags, comentar partes de código, etc.

Se ha ido indicando en el mensaje principal del Pull request la descripción que se pone en la tarjeta del tablero de Trello, incluyendo listas de tareas e imágenes de mockups.



Como se puede observar en la ilustración, se detalla bien en el Pull request la tarea como en la tarjeta de Trello, y se ve a continuación, un listado de modificaciones para llevar a cabo la tarea.

Finalmente, se observa que el Pull request se aceptó y que la rama se eliminó, pero el Pull request quedará en Pull requests cerrados, de manera que siempre se podrá consultar dichos cambios, a qué ficheros afectó, etc.

Ilustración 14 - GitHub: Pull request

En la recta final del proyecto, se lleva las siguientes **estadísticas** de cada repositorio:

- API REST en Play
    - 9 pull requests
    - 657 commits
    - 5 releases

- Cliente móvil en React Native
    - 58 pull requests
    - 299 commits
    - 4 releases

5.4.3. Integración continua: Travis CI

GitHub es compatible con un montón de servicios externos lo cual amplía la experiencia mucho más, y aquí se ha hecho uso del servicio Travis CI[28] como integración continua. Con este servicio nos aseguramos de que, en un entorno totalmente nuevo y limpio, los tests pasan al 100%, y no solo eso, a cada commit que subamos, aunque sea en una subrama, Travis pasará los tests de igual manera para comprobar que no hay ningún error.

Así que una vez terminados los commits necesarios para implementar un ticket, Travis nos indicará en GitHub, en el propio Pull request, que está pasando los tests en dos checks distintos, uno de la subrama del ticket y otro haciendo merge con la rama develop para comprobar si podemos hacer merge sin problemas.

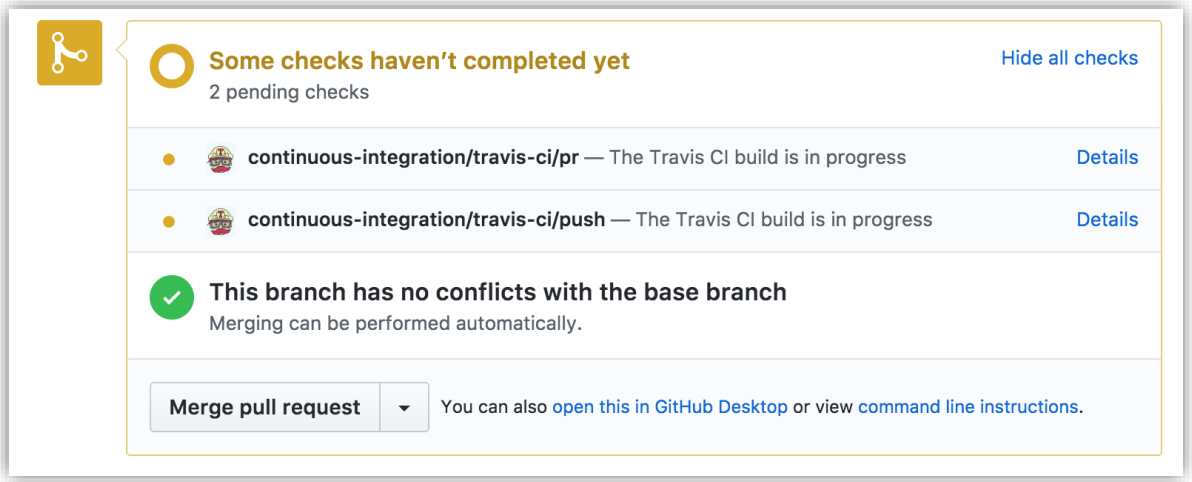


Ilustración 15 - GitHub: integración con Travis CI (en progreso)

Una vez pasados los tests en Travis, nos indicará automáticamente si han pasado o no y por lo tanto si podemos hacer merge sin miedo alguno.

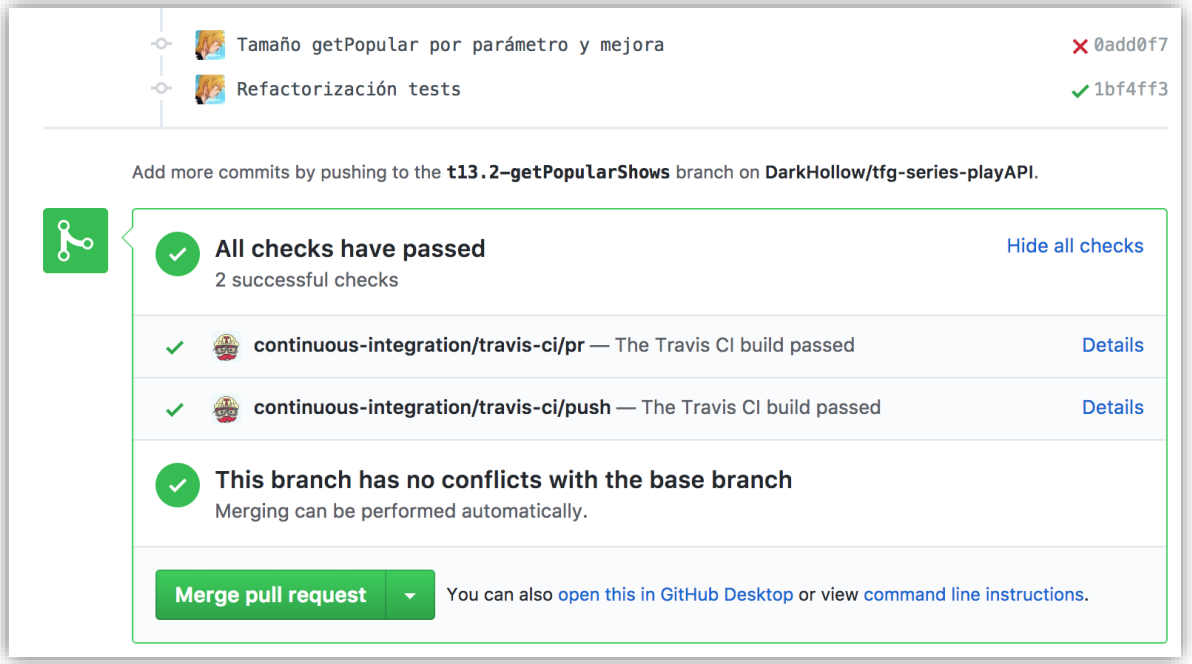


Ilustración 16 - GitHub: integración con Travis CI (finalizada)

[28] Travis CI – Servicio de integración continua y despliegue alojado en la web

Si los tests no pasaran o por cualquier razón queremos ver más información sobre las ejecuciones de Travis, tan solo tenemos que entrar en su web con nuestra cuenta de GitHub o pinchar en los enlaces que se generan (Details). De esta manera podemos ver el log completo de cada build y ejecución que Travis haga, en busca del error de los tests o cualquier cosa que queramos consultar.

Al final del proyecto se ha generado **un total de 652 construcciones en Travis CI**.

5.5. Mensajería y avisos: Slack

Para este tipo de metodología ágil, sería interesante tener una plataforma de mensajería en la que, si bien es cierto que el equipo es de un solo miembro y esta plataforma potenciaría mucho la comunicación necesaria entre distintos miembros de un equipo de más personas, también podemos beneficiarnos de ciertas características que nos brinda un servicio como lo es Slack[29].

La comunicación entre miembros que no se usará en este proyecto, pero sí se hará uso de avisos de servicios como GitHub y Travis CI. Se integrarán en canales específicos para ellos de modo que, cuando se haga un push a GitHub y Travis intente pasar los tests, llegue un mensaje a su canal, indicando si los tests han pasado o no, de manera que no se tiene que estar constantemente comprobando si han pasado o no, se recibirá una notificación que se ha configurado tanto en el navegador web como en el smartphone para estar al tanto de estas notificaciones en tiempo real, tanto si se está en el ordenador como si se está solo con el smartphone donde sea.

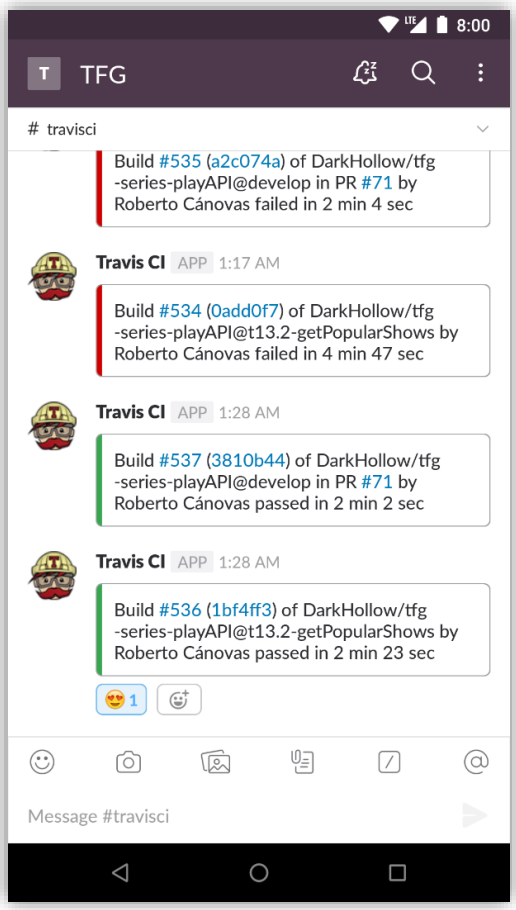


Ilustración 17 - Slack: integración con Travis CI

Con esto se cierra el apartado de metodología, donde se ha establecido la manera de trabajar e ir implementando el proyecto paso a paso, de una manera ágil que muestra muy frecuentemente el estado actual del proyecto y pudiendo probar sus avances aproximadamente cada dos semanas.

[29] Slack – Unificador de comunicaciones y optimizador del proceso de trabajo con integración de distintos servicios como GitHub, Travis CI, Trello, etc.

5.6. Releases

Mediante esta metodología se ha hecho una serie de lo que hemos llamado major releases, las cuales son un conjunto de sprints juntos que concentran cierto número de características implementadas.

5.6.1. API y Administración

Las releases de la API serían las siguientes (cada versión es un enlace a su release en GitHub). Cada versión es un enlace a la release.

- **Versión inicial v0.0.0**
    - Proyecto Play iniciado
    - Semilla Java
    - Integración continua con Travis CI
- **Versión base v0.0.1**
    - Persistencia MySQL
    - Conexión con TVDb automatizada al inicio y cada 12 horas
    - Documentación Swagger
    - Documentación Swagger vía GitHub Pages
- **Versión v0.1.0**
    - Features nuevas: FT-1
    - Serie implementada
    - Búsqueda de series
    - JSON Views implementadas
    - Control más exhaustivo parámetros de entrada
    - Fixes generales
    - Documentación Swagger actualizadas
- **Versión v0.10.0**
    - Features nuevas: FT-2, 3, 4, 5, 6, 7, 8, 9 y 10
    - Registro e identificación
    - Peticiones de series
        - Búsqueda y obtención de datos en API externa
        - Contador de petición de misma serie
    - Votaciones de series
        - Puntación global: ver
        - Puntuación personal: ver, crear, actualizar, borrar
    - Control de versiones de API y base de datos con `evolutions`
    - Administración
        - Resumen estadístico general
        - Administración de peticiones: ver, aceptar, rechazar, borrar
        - Administración de series: ver, actualizar, borrar, preview smartphone
        - Control de versiones de base de datos: ver, aceptar actualización
    - Heroku: versión de producción desplegada

    - Cambios técnicos
        - Control de roles para usuarios normales y administradores
        - Herencias implementadas para reducción y optimización de código
        - Refactorizaciones y mejora de rutas para ser REST
        - Eliminados todos los ficheros, clases y código no necesarios
        - Todas las nuevas implementaciones y refactorizaciones pensando en las futuras features
        - Tests unitarios y de integración optimizados
        - Play Framework actualizado a v2.5.12
        - Swagger UI actualizado a v3.1.7
        - Demás optimizaciones y mejoras de código
        - Demás optimizaciones en las respuestas de la API (trabajo en proceso)
- **Versión v1.0.0**
    - Features nuevas: FT-11, 12, 13, 14, 15, 16 y 17
    - Temporadas
        - Obtener externamente todas las temporadas de una serie
        - Ver temporadas incluyendo: poster, nombre, número, sinopsis, etc.
    - Episodios
        - Obtener externamente todos los episodios de las temporadas
        - Ver episodios incluyendo: screenshot, nombre, número, sinopsis, etc.
    - Popular
        - Control de todas las visitas de todas las series
        - Obtener listados con las series más visitadas
        - Obtener listados con las series mejor valoradas
    - Following
        - Seguir/dejar de seguir series
        - Lista de Mis series
    - Tendencia en Twitter
        - Obtener un ratio de tweet/hora de las series más populares
        - Obtener un listado de las series con mejor ratio de Twitter
    - Episodios vistos
        - Marcar episodios como vistos para un seguimiento de los mismos
        - Suficientes datos para desarrollar un sistema de seguimiento completo
    - Cambios técnicos
        - Herencias implementadas para reducción y optimización de código
        - Demás optimizaciones y mejoras de código
        - Demás optimizaciones en las respuestas de la API (trabajo en proceso)

5.6.2. Cliente para Smartphones

Las releases para el cliente para iOS y Android ha ido más o menos a la par que la API (cada versión es un enlace a su release en GitHub). Cada versión es un enlace a la release.

- **Versión inicial v0.0.0**
  - Proyecto React Native iniciado
  - Primeras pruebas y vista inicial
- **Versión v0.1.0**
  - Features nuevas: FT-1
  - Vista de una serie básica
  - Búsqueda de serie básica
  - Interfaz y experiencia de usuario pobre (actualmente trabajando en un diseño)
- **Versión v0.10.0**
  - Features nuevas: FT-2, 3, 4, 5, 6, 7, 8, 9 y 10
  - Registro e identificación: incluye encriptación y JWT
  - Búsqueda mejorada: nombre, banner, votación media
  - Búsqueda externa mejorada: nombre, banner, en local o no
  - Vista de una serie mejorada
    - Incluyendo: nombre, fanart con efecto parallax, poster, votación media, votación de usuario, año de estreno, géneros, duración episodios, cadena oficial, sinopsis, estado de emisión (finalizada, renovada, cancelada).
    - Votar una serie, modificar voto o borrarlo
  - Diseño 2.0
    - Nueva paleta de colores
    - Gradientes
    - Placeholders de imágenes
    - Modales para avisos, alertas, acciones
    - Animaciones: por defecto y nuevas implementaciones
    - Transiciones
    - Fuentes, iconos e indicadores de actividad según plataforma (iOS o Android)
- **Versión v1.0.0**
  - Features nuevas: FT-11, 12, 13, 14, 15, 16 y 17
  - Series
    - Marcar series como seguidas
  - Ver temporadas incluyendo: nombre, poster, número, fecha, sinopsis, etc.
  - Ver episodios incluyendo: nombre, screenshot, número, fecha, sinopsis, etc.
    - Marcar episodios como vistos o no vistos
  - Listas de series
    - Series populares en el sistema
    - Series mejor valoradas
    - Series con mejor ratio en Twitter
    - Mis series
  - Sistema de seguimiento de series/temporadas/episodios
    - Se muestra en todos los posters un indicador de episodios no vistos

- Vista principal implementada
- Muchos componentes nuevos desarrollados
- Nuevos componentes externos instalados
- Muchas mejoras de implementación y unificación de vistas

6. Arquitectura

Se ha tenido que elegir la arquitectura de cada una de las distintas partes del proyecto como son la API, los clientes móviles para iOS y Android y finalmente, la parte de administración.

6.1. API REST

En cuanto a la API, se ha optado por la arquitectura de capas siguiente:

- Modelo: capa que se encarga del acceso a los datos realizando la conexión con la base de datos. Se emplea el patrón DAO (Data Access Object) el cual se centra en dicho acceso, aislando este al resto de la aplicación.

- Servicio: capa que contiene toda la lógica de negocio para la manipulación de los datos que se obtienen en la capa modelo, de esta manera aislamos la lógica de negocio tanto del controlador como del modelo.

- Controlador: capa que se encarga de recibir las peticiones y de generar las respuestas. Para ello hace uso de la capa servicio para obtener o alterar los datos que se requieren.

La API no necesita la capa de Vista, ya que todos los datos se devuelven en JSON.

6.2. Administración

Para la parte de administración, la cual va incluida en el proyecto Play de la API, se ha utilizado el patrón MVCS (Modelo, Vista, Controlador, Servicio), el cual es el descrito en el caso de la API, pero añadiendo la capa vista:

- Vista: capa que se encarga de mostrar la interfaz al usuario, y mediante acciones de la propia vista, realizar peticiones al sistema para pedir datos o alterarlos.

6.3. Cliente móvil

El cliente móvil, desarrollado en React-Native, se hará únicamente de una capa:

- Vista: capa que se encargará de mostrar las vistas o interfaces de la aplicación móvil

Al ser React-Native una librería basada en Javascript y React, se puede aplicar la lógica que se necesite, obtener datos para mostrarlos, realizar peticiones, mantener en memoria su state, etc.

En el siguiente diagrama queda más claro en conjunto toda la arquitectura juntando las 3 partes del proyecto:

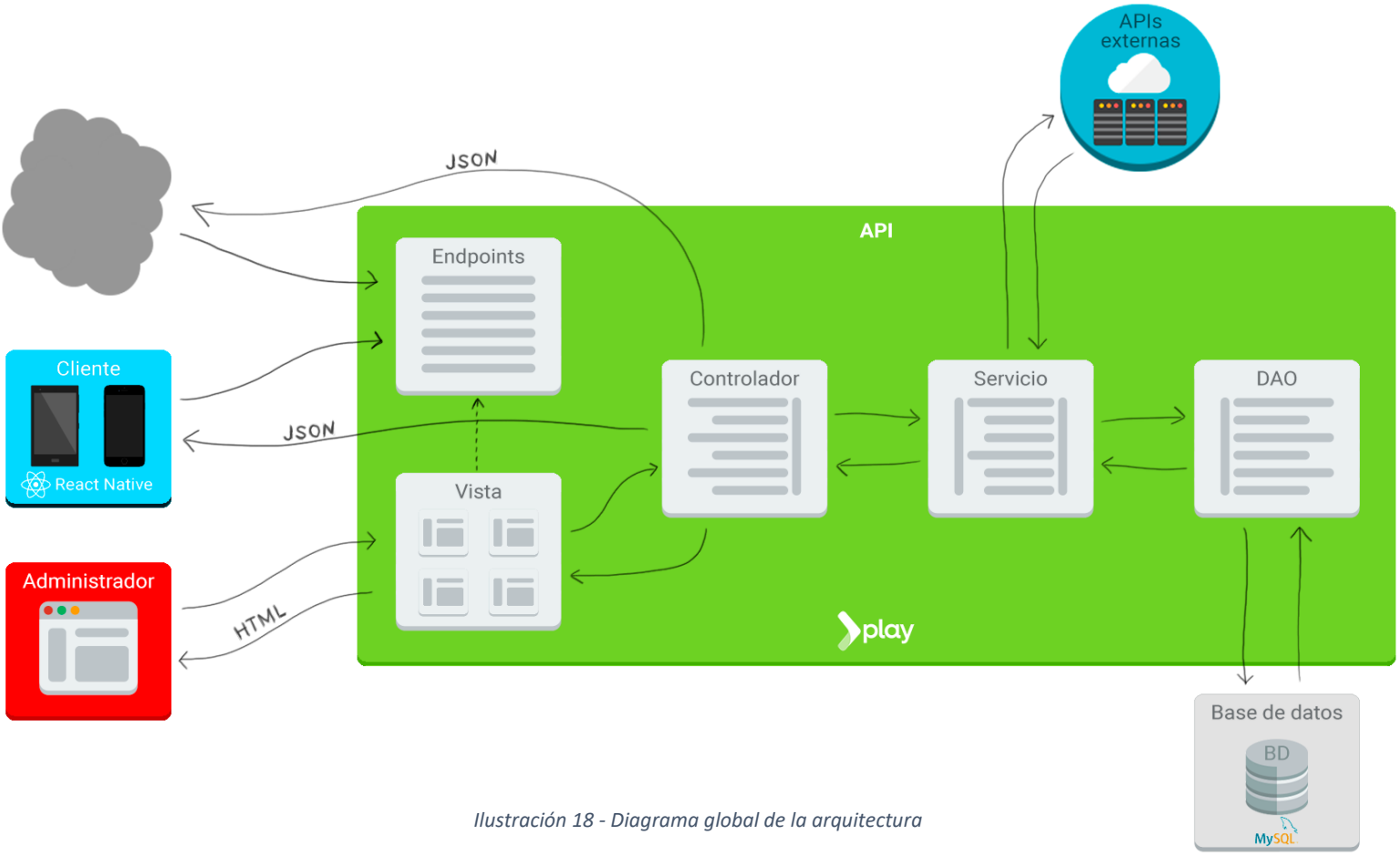


Ilustración 18 - Diagrama global de la arquitectura

Gracias a esta arquitectura, separando y aislando cada una de sus partes, se puede obtener diversos beneficios, como por ejemplo, el de separar la capa de servicio de la del modelo, aislando por completo el acceso a datos de la lógica de esos mismos datos, o de separar la capa de controlador de la de servicio, haciendo lo mismo que en las capas anteriores y además pudiendo controlar libremente en el controlador cómo se devuelven esos datos, con qué tipo de respuesta, control de errores y excepciones, etc., dejando todo muy limpio y de fácil mantenimiento y escalabilidad.

6.4. Evolución del modelo de datos

Para el desarrollo del modelo de datos se ha utilizado una metodología iterativa, en la que partimos de un diseño inicial muy básico al que se le va añadiendo nuevas entidades conforme se necesitan para la implementación de nuevas funcionalidades y relaciones entre ellas.

- API v0.0.0: sin modelo de datos
- API v0.0.1: modelo de datos base vacío (persistencia implementada)
- API v0.1.0: nuevas entidades Serie y Test

Ilustración 19 - Diagrama base de datos v0.1.0

- API v0.10.0: nuevas entidades Usuario, Petición, Voto, Evolución

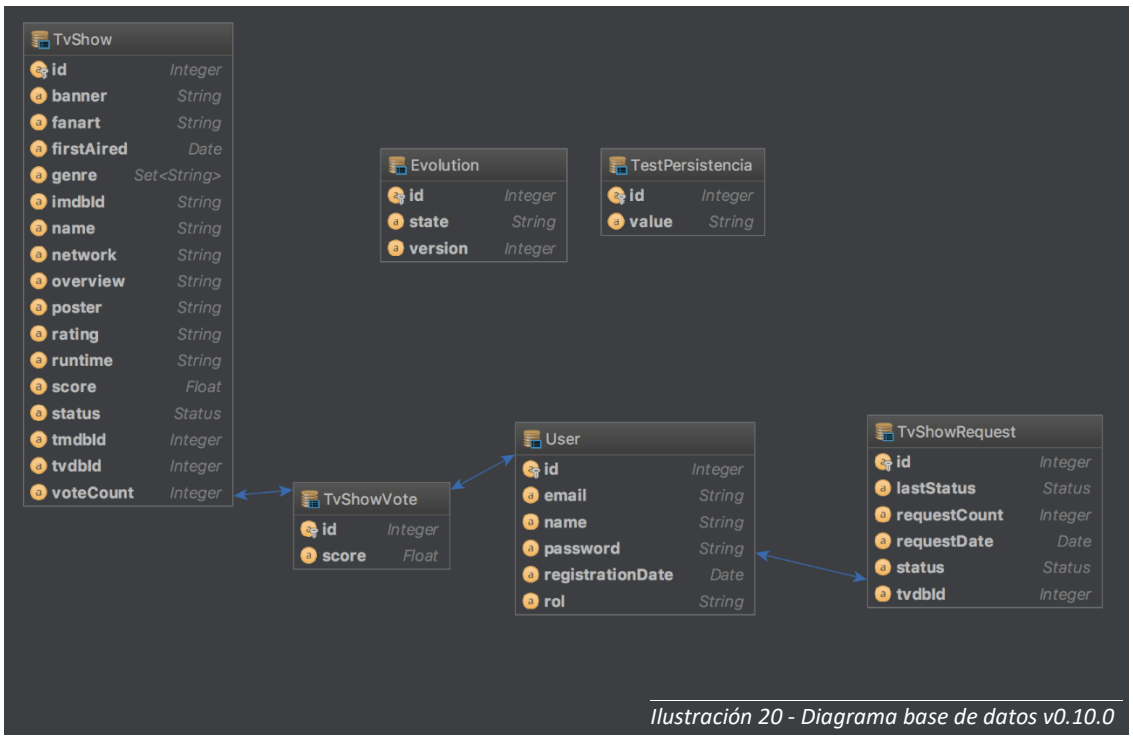


Ilustración 20 - Diagrama base de datos v0.10.0

- API v1.0.0: nuevas entidades EpisodeSeen, Popular, relaciones: following

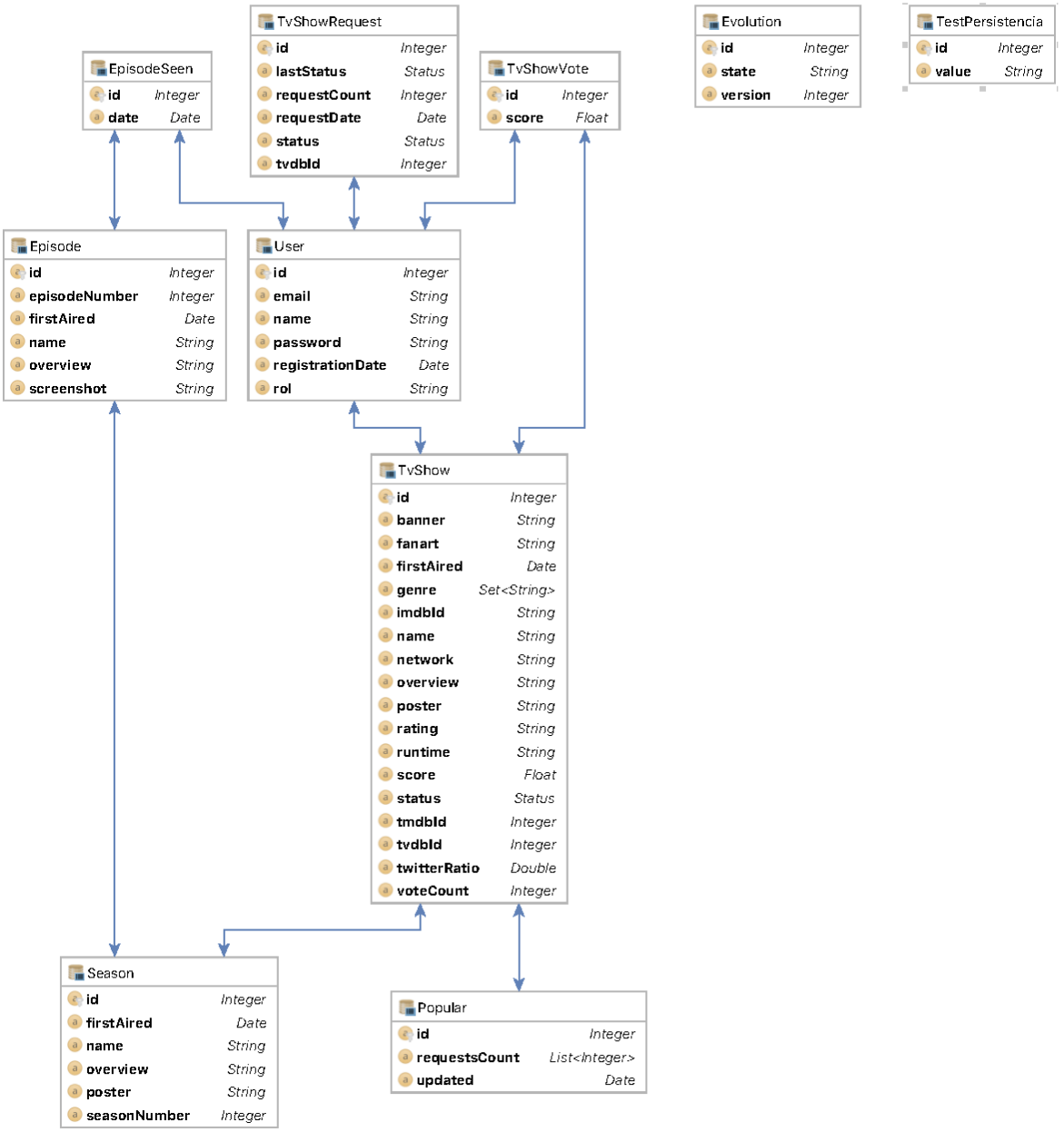


Ilustración 21 - Diagrama base de datos v1.0.0

Esta ha sido la versión final del modelo de datos el cual todavía no es lo suficientemente complejo como para no hacerlo seguir creciendo con nuevas funcionalidades futuras.

6.5. Implementación de la arquitectura con Play

Para entender un poco mejor cómo se implementa una acción en Play siguiendo la arquitectura descrita, lo mejor es ver un simple ejemplo real.

Lo primero es conocer su estructura. La estructura de un proyecto típico en Play es la siguiente:

```
┌/proyecto-play
│
├──/app
│   │
│   ├──/controllers
│   │
│   ├──/models
│   │   │
│   │   ├──/dao
│   │   │
│   │   └──/service
│   │
│   └──/views
│
├──/conf
│   │
│   ├──/routes
│   │
│   └──/application.conf
│
├──/public
│
└──/test
```

Para ver el proceso de implementación de una acción, se necesita un ejemplo de acción y ruta:

- Acción: devolver una serie por id
- Ruta deseada: /api/tvshows/{id}

Para devolver una serie buscada por su id se necesita acceso a la base de datos en el que se filtren las series por id. Gracias a JPA, tenemos una operación en su API que nos permite buscar directamente por id el objeto en cuestión.

`/app/models/dao/TvShowDAO.java`

```java
// buscar por id
public TvShow find(Integer id) {
  return jpa.em().find(TvShow.class, id);
}
```

Ahora, se necesita que en el Servicio pertinente existe una función que llame a la implementada en el DAO, y en ella se añadirá la lógica necesaria (en caso necesario) para procesar los datos.

`/app/models/service/TvShowService.java`

```java
// buscar por id
public TvShow find(Integer id) {
  return tvShowDAO.find(id);
}
```

En este caso, no hace falta añadir ninguna lógica por lo que queda muy limpio. Si fuera necesario, se llamaría a otros servicios o servicios externos desde aquí para poder cumplimentar las condiciones de los datos que han sido solicitados.

Lo siguiente será definir su acción en el Controlador en cuestión para poder devolver los datos solicitados.

`/app/controllers/TvShowController.java`

```
// devolver un TvShow por id
@Transactional(readOnly = true)
@Security.Authenticated(Roles.class)
public Result tvShowById(Integer id) {
  TvShow tvShow = tvShowService.find(id);

  if(tvShow == null) {
    ObjectNode result = Json.newObject();
    result.put("error", "Not found");
    return notFound(result);
  }

  try {
    JsonNode jsonNode = jsonUtils.jsonParseObject(tvShow,
JsonViews.FullTvShow.class);
    return ok(jsonNode);
  } catch (Exception ex) {
    ObjectNode result = Json.newObject();
    result.put("error", "It can't be processed");
    return internalServerError(result);
  }
}
```

Como se puede observar en un código que habla por sí mismo, primero se anota que es una transacción de solo lectura, porque solo vamos a obtener los datos de una serie (y no crear, ni modificar, ni borrar ningún dato).

Lo siguiente que se hace es llamar a la función del servicio que antes se ha implementado para obtener el objeto de la serie con esa id. Se comprueba si la serie ha sido obtenida (existe) o si no (no existe serie con esa id). En caso de no encontrarse, se devuelve un objeto JSON que contiene los datos de que no ha sido encontrado con una respuesta del tipo *Not Found* (Status 404).

En caso de que haya sido encontrado, se parsea el objeto serie a JSON y se devuelve con una respuesta del tipo *Ok* (Status 200).

Por último, el método de parseo a JSON puede lanzar excepciones, así que se capturan y se devuelve un objeto JSON de error con una respuesta del tipo *Internal Server Error* (Status 500).

Finalmente, lo único que queda por hacer, es anotar la ruta deseada en el fichero de rutas que invoca a la acción implementada anteriormente en el controlador.

`/app/conf/routes`

```
GET  /api/tvshows/:id    controllers.TvShowController.tvShowById(id: Integer)
```

Ahora, accediendo a la ruta implementada, se puede comprobar el resultado.

Accediendo una serie que existe.

Petición

```
GET /api/tvshows/10

No content
```

Respuesta

```
// Status: 200 OK

{
  "id": 10,
  "tvdbId": 305288,
  "tmdbId": 66732,
  "name": "Stranger Things",
  "firstAired": "2016-07-14",
  "banner": "./assets/images/series/10/banner.jpg",
  ...
}
```

Accediendo una serie que no existe.

Petición

```
GET /api/tvshows/99999999

No content
```

Respuesta

```
// Status: 404 Not Found

{
  "error": "Not found”
}
```

Como ya se ha explicado, esta arquitectura y jerarquía permite una fácil detección de errores al tenerlo todo aislado y localizado, un fácil y claro sistema de respuestas y control de errores y excepciones en el controlador para saber qué respuesta dar, un fácil mantenimiento del código y una muy buena escalabilidad.

7. Diseño

Al comienzo del proyecto, se pensaba realizar un diseño con fondo claro y texto oscuro tanto para la administración como para los clientes, pero más adelante se decidió diseñar una interfaz con una paleta oscura para los clientes, dado que lo importante era resaltar las imágenes, tanto de las portadas de las series, como de las temporadas o imágenes de los episodios.

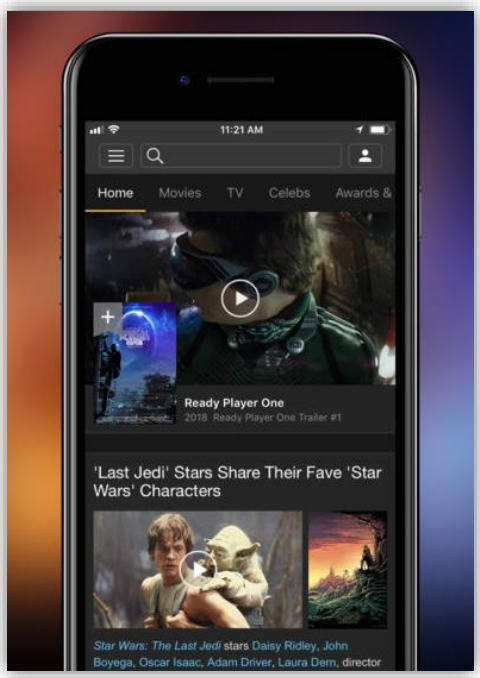


Ilustración 22 - IMDb en iTunes

Como se comprobó en el estado del arte, la mayoría de aplicaciones para smartphones que tratan sobre series o películas, tienen un diseño oscuro por esto mismo.

Además, gracias a la paleta oscura, da una sensación de estar en un cine dando una inmersión adecuada a la aplicación que encaja con el tipo de contenido que se muestra.

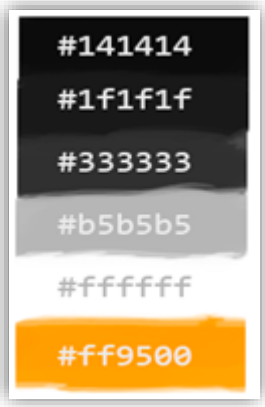
Los colores principales y de resalto por lo general son blancos, amarillos o rojos.

Así pues, se realizó un diseño principal y una paleta de colores de la que heredarían todas las demás vistas de la parte del front-end.

También, conformé se aprendía más y se estaba más cómodo con React Native, se fue desarrollando e imponiendo unas pautas en cuanto a las pequeñas animaciones que se pueden encontrar en el flujo de acciones de los clientes y en las transiciones.

7.1. Paleta de colores

Como ya se ha dicho, se decidió que la paleta fuera oscura, pero faltaba encontrar un color principal para los botones o enlaces y resalte de elementos.



Se ha elegido la paleta de tal manera que de un color a otro se puede llegar mediante transparencias salvo el color principal.

El color principal se ha elegido de una tonalidad entre el amarillo y el naranja que resulta ser agradable para la vista en fondos oscuros y que ha sido aceptado por todos los usuarios de tests de usabilidad.

Gracias a esta paleta, las imágenes resultarán más vistosas.

Ilustración 23 - Paleta de colores definitiva

Una vez elegida la paleta, la duda estaba en si diseñar una interfaz distinta, aunque parecida, para iOS y para Android ya que en ambas se suelen seguir unas pautas ligeramente distintas en cuanto a la organización de ciertos elementos de las vistas.

Finalmente, se decidió combinar lo que mejor se consideraba de ambas para crear un diseño único, sin diferenciar plataformas y teniendo desarrolladas las vistas en un solo fichero con misma implementación. De esta manera, mantener el código será mucho más fácil que tener dos implementaciones de vistas distintas.

7.2. Mockups

Para el diseño de los mockups coloreados se ha hecho uso de una tableta gráfica Wacom Intuos Art y del software Adobe Photoshop CC 2017. En estos mockups, se ha hecho uso directamente de la paleta de colores anteriormente descrita.

7.2.1. Mockup identificación y registro

Estas dos vistas son exactamente iguales, solo cambia los distintos campos de los formularios y los textos de los botones.

El fondo oscuro, se hecho semitransparente para colocar debajo una malla de portadas de series difuminada, de forma que se vea ligeramente debajo de toda la vista, lo cual queda visualmente bien y acorde con el tipo de contenido que se encontrará el usuario al entrar en la aplicación.

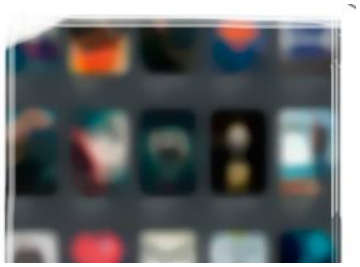


Ilustración 24 – Mockup malla series

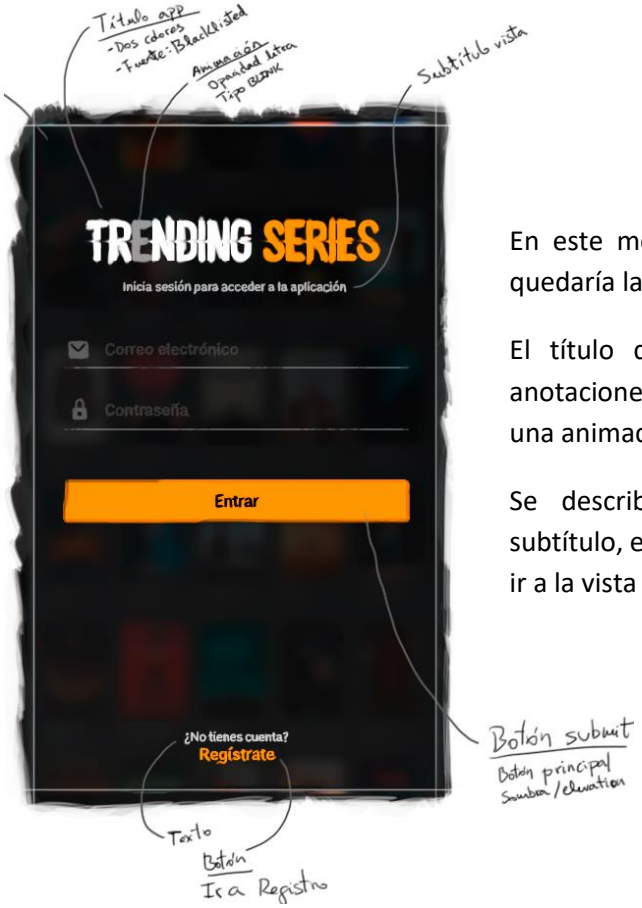


Ilustración 25 – Mockup identificación completo

En este mockup, se puede ver perfectamente cómo quedaría la malla de series de fondo.

El título de la aplicación, como se indica en las anotaciones hechas a mano, contiene en la primera "E" una animación de *blink* o parpadeo.

Se describe los demás elementos incluyendo el subtítulo, el botón de enviar el formulario y el botón de ir a la vista de registro.

Y, como se puede comprobar en la Ilustración de la derecha, se ha descrito también los iconos, efectos y animaciones que ocurren si se activa un cuadro de texto del formulario o se sale de él, haciéndose el icono más pequeño y cambiando de color la línea del campo.

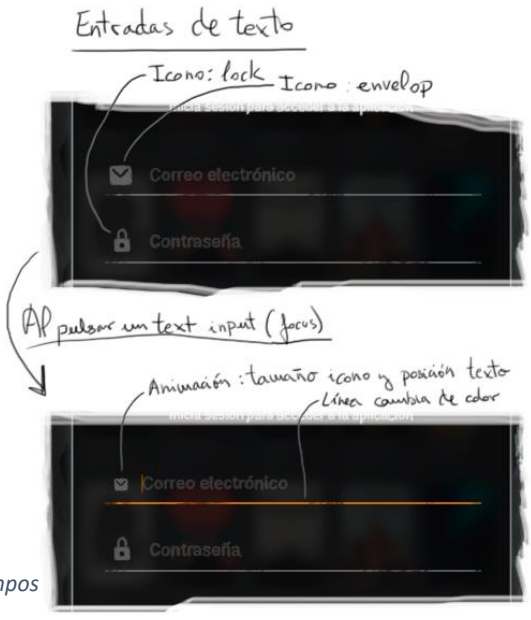


Ilustración 26 - Mockup identificación: animación campos

En esta Ilustración, se puede observar cómo sería el formulario de identificación con los campos rellenos, en los que el texto que los rellena, como se indica al pie, debe ser más opaco que los placeholder.

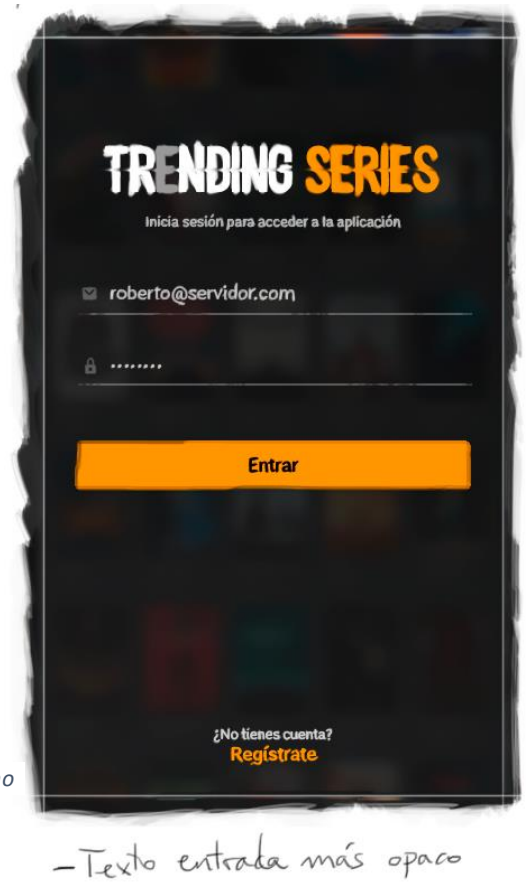


Ilustración 27 - Mockup identificación: formulario relleno

El mockup de la vista de **registro**, como se ha dicho, es exactamente igual, lo único que cambia son los campos del formulario y los botones.

Al pie de ambos mockups se ven botones para ir de una vista a la otra, en función de la necesidad del usuario.

Ilustración 28 - Mockup registro: completo

7.2.2. Mockup de generalidades

El diseño de toda la aplicación tiene que tener una consistencia y unas reglas generales que se deben aplicar a todas las vistas, si no, no sería un diseño conciso, sería algo aleatorio. Habiendo realizado ya los mockups diseñados para la vista de identificación y de registro y habiéndoles dado ya el visto bueno, se continuó esa línea de diseño para el resto de la aplicación.

Se empezó diseñando un mockup de vista vacía con barra de título de vista en la parte superior como esquema y diseño base de toda la aplicación; cada mockup de cualquier vista hereda de este mockup.

Como se puede observar, se utiliza los colores de la paleta mostrada anteriormente.

Ilustración 29 - Mockup padre

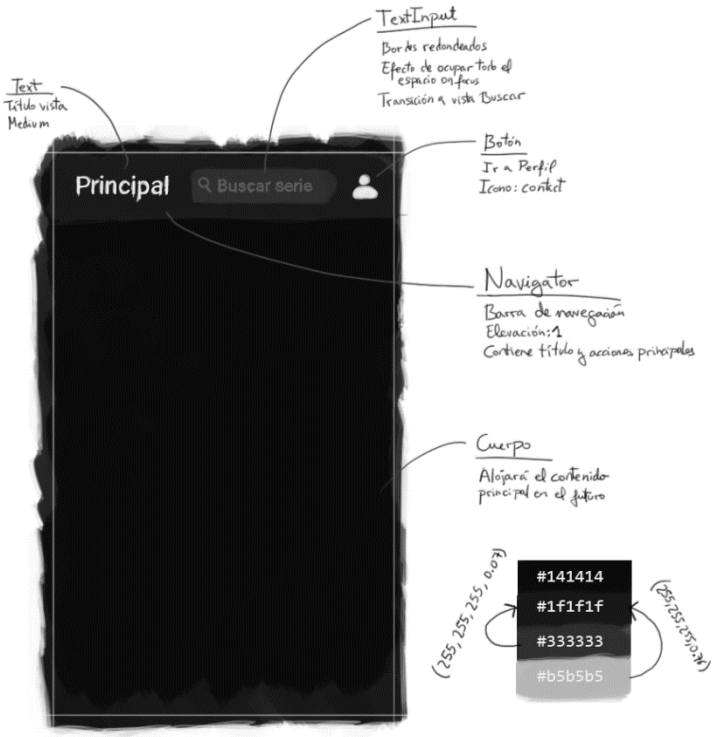


Ilustración 30 - Mockup principal vacío

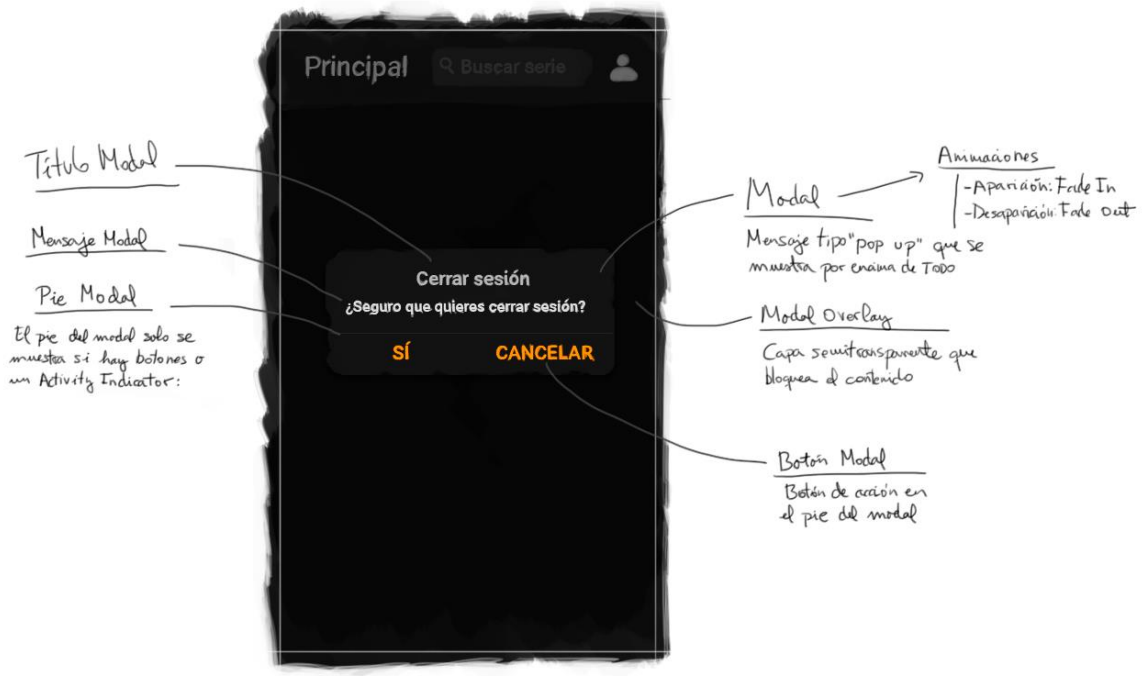
De este mockup, se hizo una versión básica de la vista principal de la aplicación, pero con el cuerpo vacío, ya que aún no estaba diseñada la vista principal y no se sabía del todo qué elementos iba a contener.

Este mockup principal vacío se realizó para diseñar elementos comunes que se verían por todas las vistas de la aplicación, como los títulos de vista, iconos, modales[30], ciertas animaciones, etc.

[30] Modal – elemento gráfico que simula una ventana superpuesta a una vista para informar de algún evento o confirmar/cancelar acciones

Modales

Para los modales, que sustituyen al típico Alert, se decidió que se diseñaría del mismo modo que las vistas en vez de coger el diseño original de iOS y Android, para que la aplicación no se diferencie tanto de una plataforma a otra y tuviera más personalidad.



Como se aprecia en la Ilustración anterior, se ha diseñado y definido muy bien cada parte del Modal, teniendo un título, un cuerpo con mensaje y un pie. Este pie puede no aparecer en caso de no necesitarlo o contener distintos elementos:

- Botones: sí, no, cancelar, etc.
- Indicador de actividad: icono y animación según plataforma

También se ha definido bien la opacidad de la capa superpuesta a toda la vista llamada **overlay** y la animación con la que aparecerá y desaparecerá el modal: animación de tipo fade.

Por último, en la Ilustración de la derecha se puede observar cómo sería el modal con indicador de actividad en el pie en vez de botones.

Se ha detallado el tipo de animación que tendrá (spin) y plataforma disfrutará de su indicador de actividad de sistema, pero con el color de acentuado de la paleta de colores que se confeccionó para el diseño de esta aplicación.

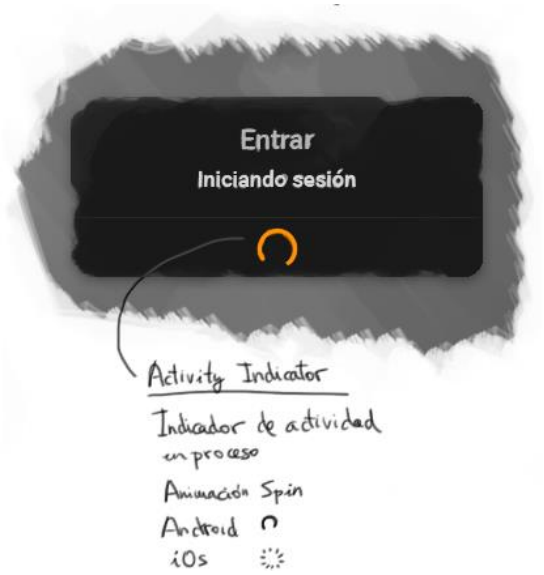


Ilustración 31 - Mockup Modal: indicador actividad

7.2.3. Mockup vista principal

Esta es la vista a la que navega la aplicación una vez que el usuario se ha identificado correctamente.

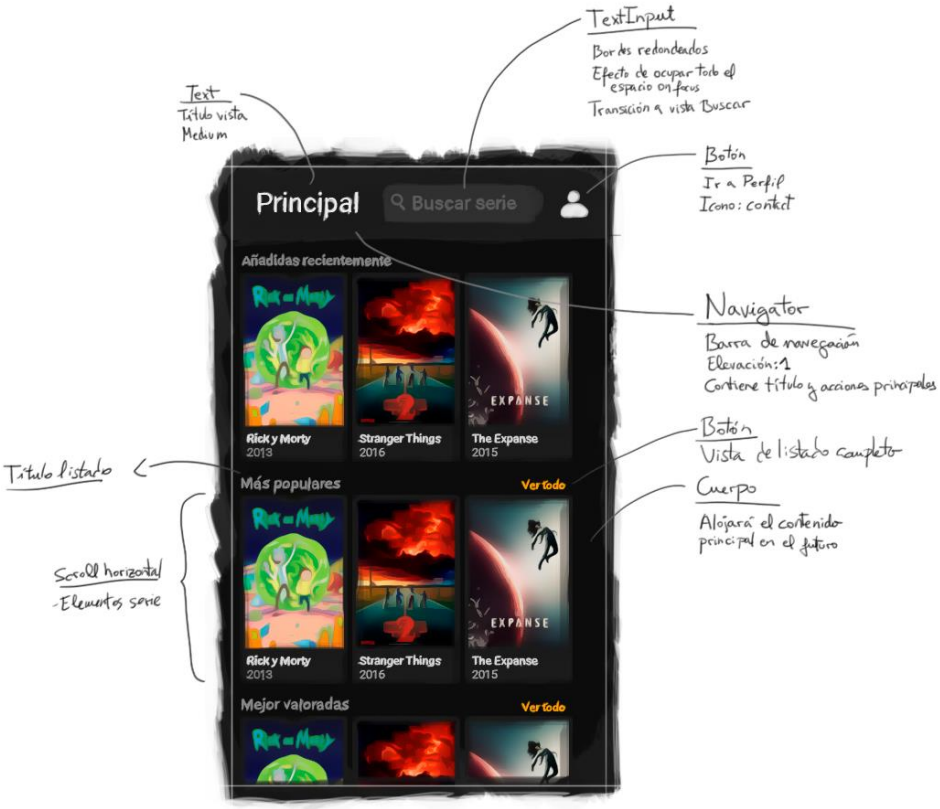


Ilustración 32 - Cliente: principal

Se puede observar en la barra superior el título de la vista, el campo de búsqueda con el que se accederá a la vista de búsqueda, y también un botón para ir a la vista de usuario o cerrar sesión.

En el cuerpo de la vista se encuentra el contenido principal. En este caso, se ha planificado que contenga listas de distintas categorías de series como las recientemente añadidas, las más populares o las más valoradas.

Como se observa en la ilustración de la derecha, una lista contiene un título de lista, un botón de **Ver todo** que nos llevaría a una vista con el listado completo, y también contiene una lista horizontal de series.

Se ha diseñado para que el scroll se haga horizontalmente y que, al pulsar en cualquiera de ellas, se cargue la vista de una serie con la serie seleccionada.

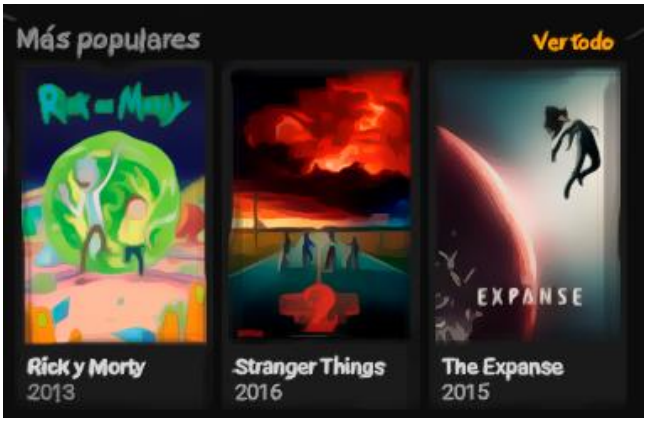


Ilustración 33 - Cliente: principal - listado horizontal

Se ha diseñado cómo debe ser la animación de cambio de vista se si pulsa en el cuadro de texto de búsqueda de la barra de título de la vista.

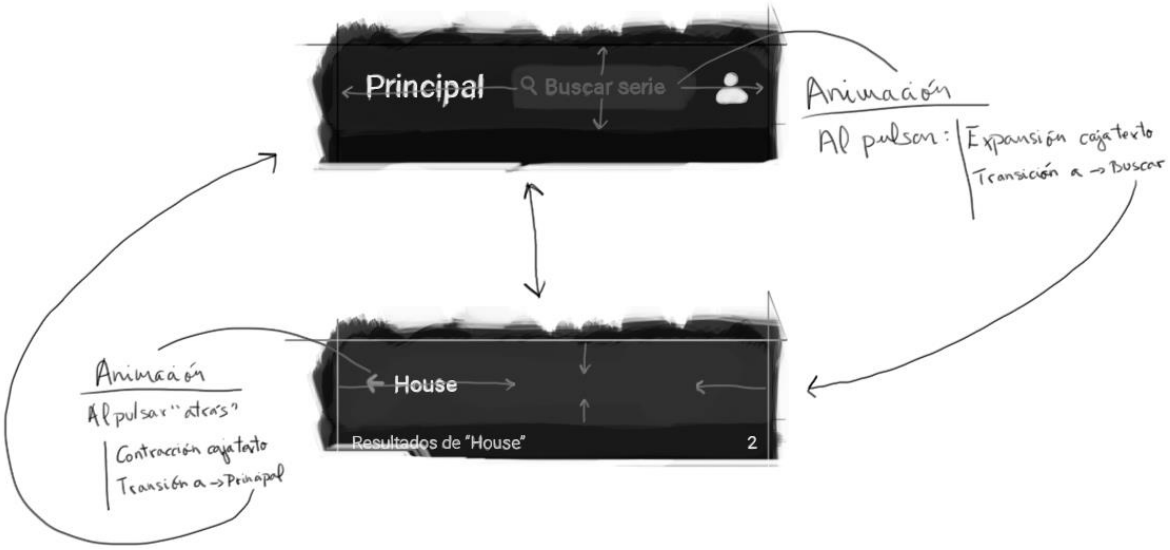


Ilustración 34 - Mockup principal/búsqueda: animación

La idea es que, al pulsar sobre el cuadro de texto, este se expanda por toda la barra de título ocupándola al completo y con animación de fade aparezca la vista de búsqueda. De igual manera, si se volviera de la vista de búsqueda, la animación se ejecutaría al revés.

7.2.4. Mockup vista búsqueda

Esta en la vista donde se podrá realizar búsquedas de series en el sistema. Se viene a esta vista desde la vista principal.

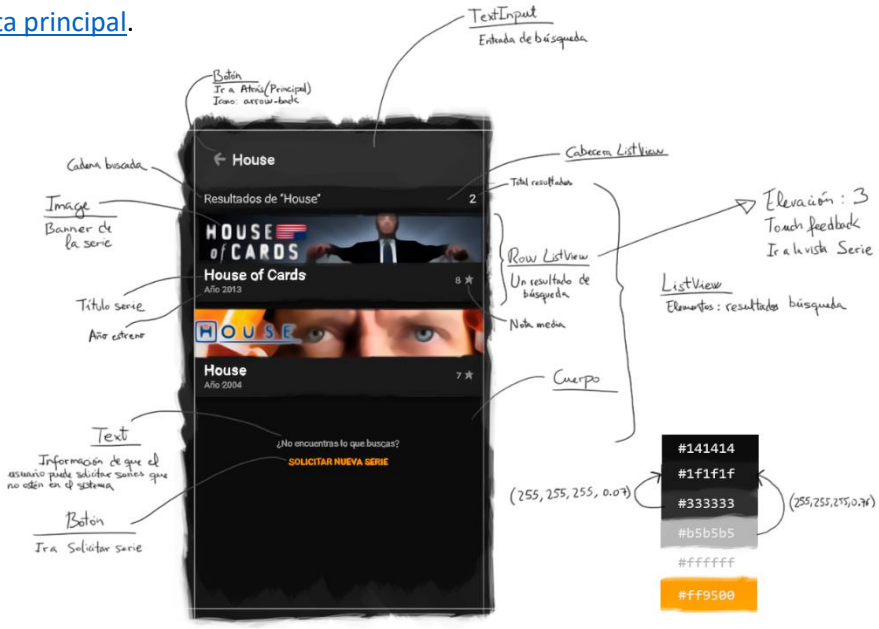


Ilustración 35 - Mockup búsqueda: completo

En la parte superior podemos ver cómo la caja de búsqueda mediante la animación ha pasado a ocupar la totalidad de la barra de título de vista, y también aparece el botón de ir hacia atrás. Este cuadro de texto es el de búsqueda.

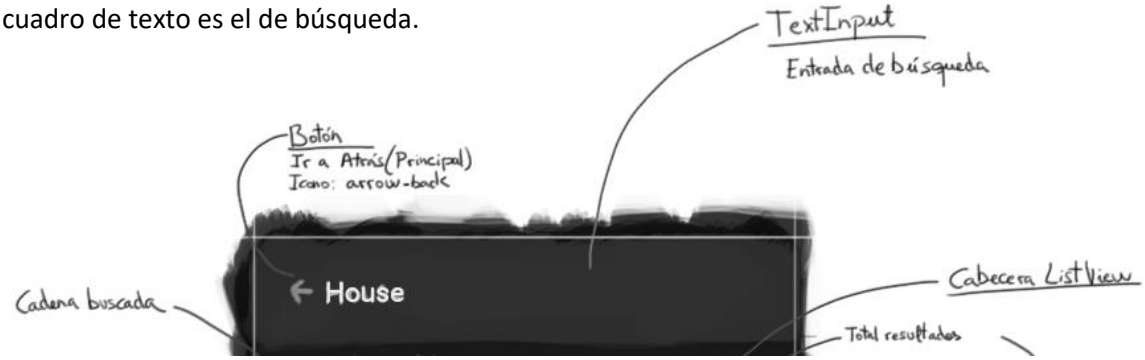


Ilustración 36 - Mockup búsqueda: cuadro de búsqueda

La siguiente parte, es la lista de scroll vertical de los resultados encontrados. Se muestra el número de resultados y luego cada una de las series mostrando su imagen, nombre, año y valoración.

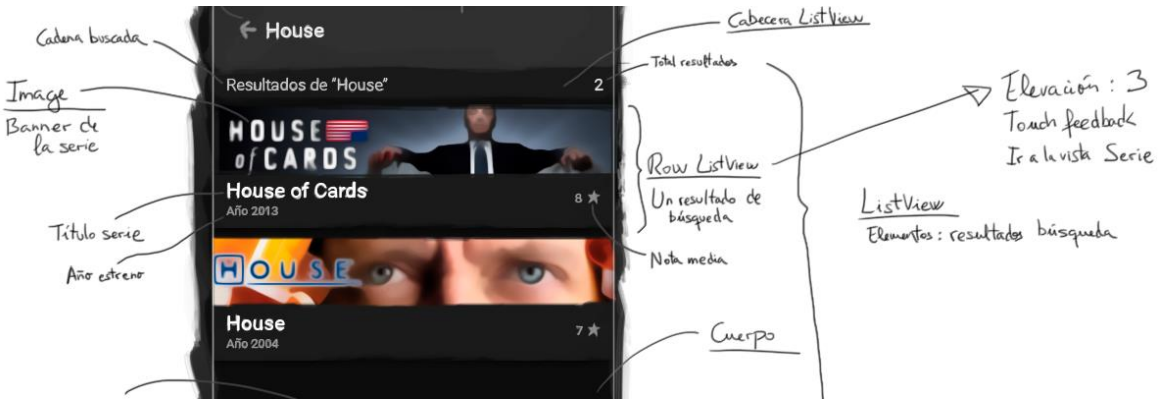


Ilustración 37 - Mockup búsqueda: lista de resultados

Si se echa un vistazo más de cerca a un resultado se puede ver lo siguiente.

La mitad superior será ocupada por el banner de la serie, y en la mitad inferior se colocará la información que se ha nombrado: nombre, año de estreno y valoración media.

Ilustración 38 - Mockup búsqueda: detalle resultado

Si se pulsa en cualquier parte del resultado, llevará a la vista de una serie en concreto y mostrará toda su información.

Por último, al pie de la lista de resultados encontrados, se encuentra un texto y un botón. Son los correspondientes a si no se encuentra la serie que se está buscando.

Ilustración 39 - Mockup búsqueda: botón solicitar nueva serie

Este enlace llevaría a la vista de solicitar nueva serie, con algún efecto de transición de abajo a arriba para la nueva vista que entra, como si se pusiera encima, ya que trata de lo mismo, de una búsqueda, aunque en servicio externo.

7.2.5. Mockup vista solicitar nueva serie

En esta vista, de la que se viene a partir de la vista de búsqueda de series al no encontrar la que se buscaba, se podrá realizar búsquedas en un servicio externo para poder solicitar series nuevas.

Ilustración 40 - Mockup solicitar serie: completo

Lo primero en lo que difiere este diseño del anterior, es que ha salido una barra de título en la parte superior, y el cuadro de texto de búsqueda ha pasado a estar justo debajo.

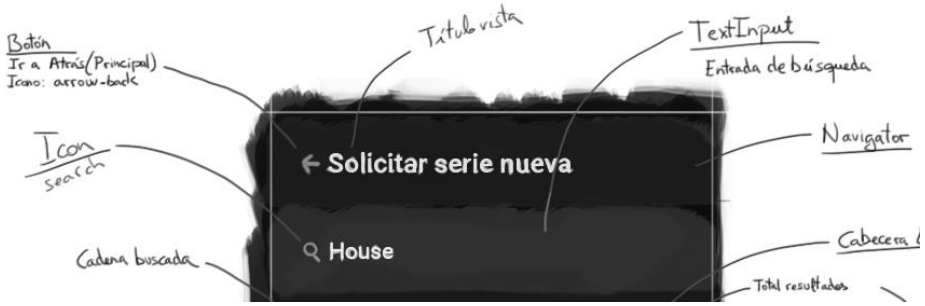


Ilustración 41 - Mockup vista solicitar serie: título y cuadro de búsqueda

Lo siguiente es la lista de scroll vertical de los resultados. Se muestra el número de resultados y para las series muestra su imagen, nombre y año.

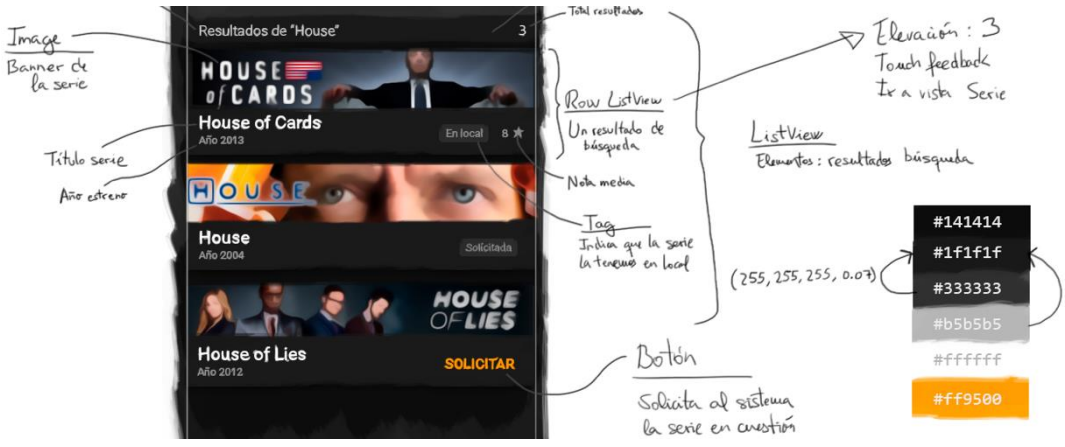


Ilustración 42 - Mockup solicitar serie: lista deresultados

Si se echa un vistazo más de cerca a un resultado de la búsqueda podemos ver más detalles.

Series encontradas externamente que no están en el sistema

Para las series encontradas que no estén en el sistema, muestra un texto de **SOLICITAR**, para que, pulsando en la serie, se solicite.

Ilustración 43 - Mockup solicitar serie: no solicitada

Series encontradas externamente que han sido solicitadas, pero no están en el sistema

Las series que se encuentren externamente que ya hayan sido solicitadas, pero no estén en el sistema se marcan con una etiqueta **Solicitada**.

Ilustración 44 - Mockup solicitar serie: solicitada

Series encontradas externamente que ya están en el sistema

En el caso de las series encontradas que ya están en el sistema, aparece una etiqueta diciendo que ya está **En local**, además de mostrar su valoración media.

Ilustración 45 - Mockup solicitar serie: en local

7.2.6. Mockup vista de una serie

En esta vista es donde se podrá encontrar toda la información sobre las series. Este es el diseño que más ha ido cambiando con el tiempo y que más componentes muestra en pantalla. También es la vista para la que se empezaron a diseñar y desarrollar animaciones que se extendieron al resto de vistas. Además, hay elementos y estructuras que se extienden a otras vistas como la de una temporada.

Ilustración 46 - Mockup serie: completo

Conociendo un poco más React Native, se conocía mejor las posibilidades y limitaciones que se podrían encontrar a la hora de desarrollar lo que se diseñara en los mockups, así que, sabiéndolas de antemano, fue fácil diseñar esta vista lo más *compleja* posible.

En el primer mockup de esta combinación de mockups, se puede ver cómo se vería la vista nada más entrar en una serie.

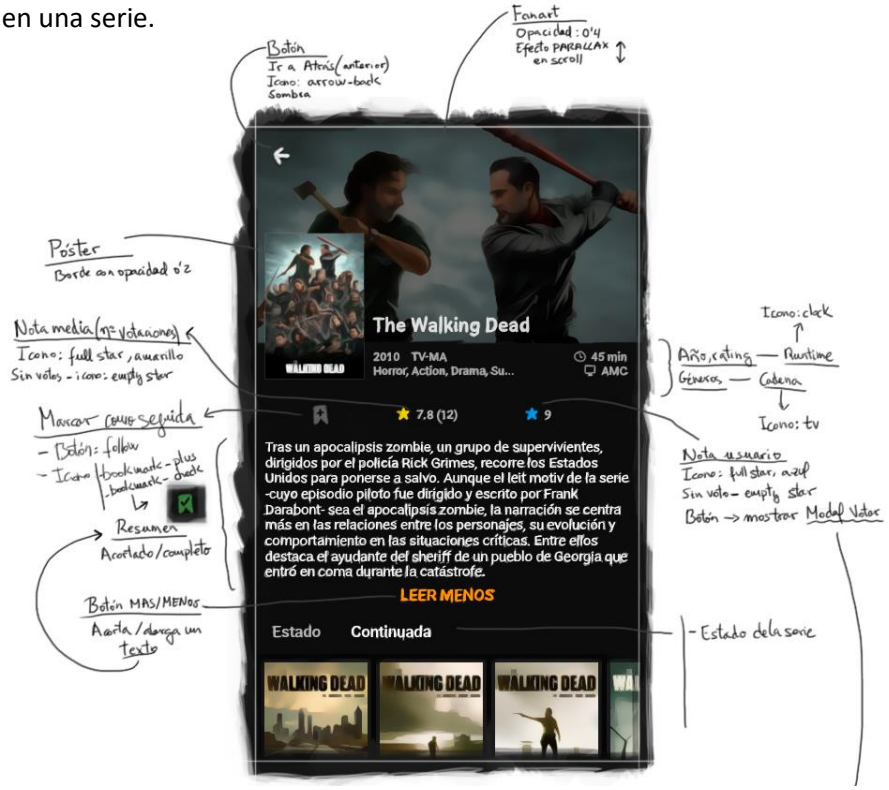
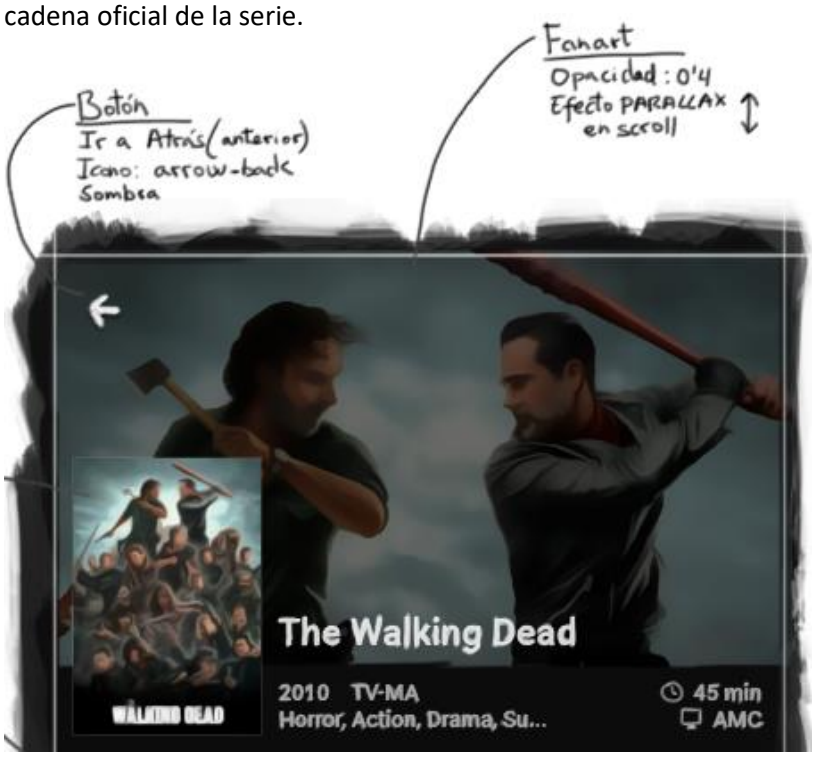


Ilustración 47 - Mockup serie: vista completa

Lo primero que se encuentra en este diseño es la cabecera, en la que se encuentra el botón atrás, imagen fanart de fondo, el póster, nombre, año de estreno, rating, género, tiempo de episodio y cadena oficial de la serie.



Como se puede observar en la ilustración de la izquierda, la imagen fanart de fondo tiene diseñada una opacidad del 40% y un efecto *parallax* de scroll vertical. El efecto hará que cuando se haga scroll-down, el contenido suba, pero la imagen suba a un ritmo más lento, para que de la sensación de que tiene una profundidad mayor al resto del contenido.

Ilustración 48 - Mockup serie: cabecera

En la siguiente sección de este mockup, se ha diseñado los botones de acciones para la serie y la media de las valoraciones de la serie por parte de los usuarios.

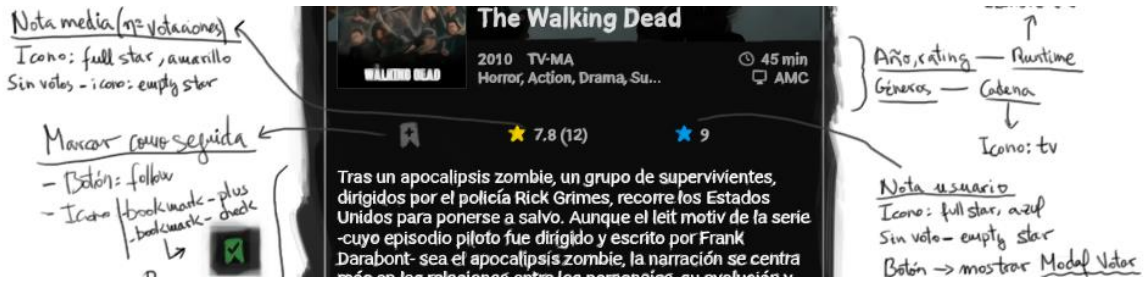


Ilustración 49 - Mockup serie: acciones y nota

Las acciones son dos:

- Marcar serie como vista: con el botón del marcador
- Votar serie: con el botón de la estrella azul – este botón abre el [modal de votar]

Modal votar serie

Al pulsar sobre el botón de la estrella azul, se debe de abrir el modal de votación como ya se ha dicho:

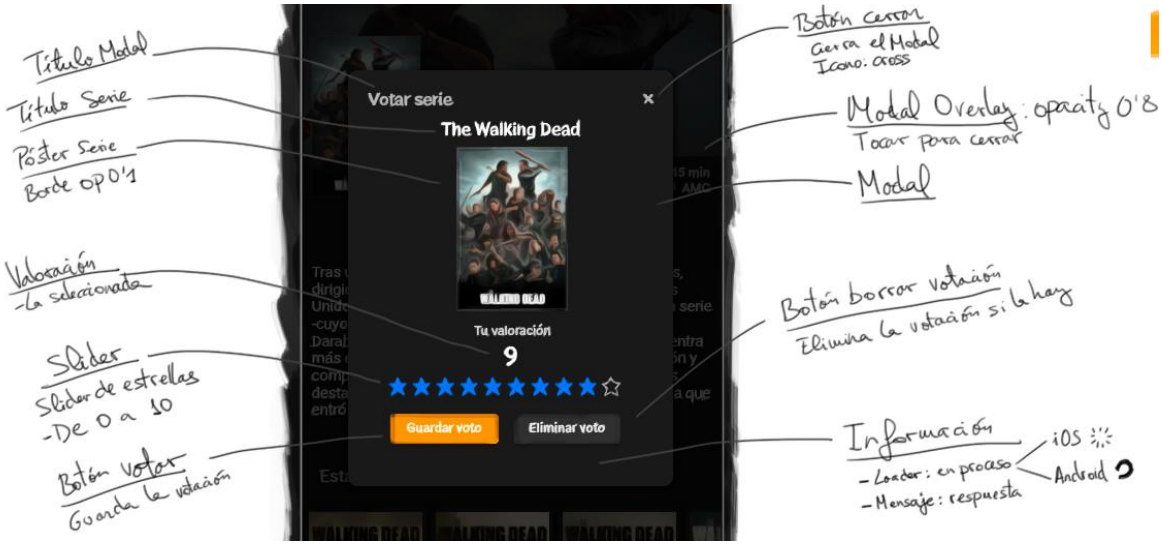


Ilustración 50 - Mokcup serie: modal votar

Se puede observar detalladamente cada elemento de este modal, como el numerador de la votación que va deberá ir cambiando el número según se arrastre el dedo sobre el slider de estrellas azules.

También contiene dos botones, uno para guardar la votación en caso de modificación de la misma, y otro para eliminar la votación en caso de que se quiera borrar la votación guardada.

Es modal, deberá aparecer encima de la vista de la serie, oscureciéndola para destacar el modal. Se ha indicado también en el diseño que al pulsar fuera del modal o en el botón de cerrar o incluso al realizar cualquiera de las dos acciones de guardar o eliminar voto, se cierre automáticamente.

En la siguiente sección se encuentra diseñada la sinopsis, la cual ocupa todo el ancho de la vista. Se quiere que, si esta descripción es demasiado larga, se contraiga a unas tres líneas y que con un botón de **LEER MÁS/LEER MENOS** se extienda/contraiga.

Justo después de la sinopsis, se colocado en el diseño el **estado** de la serie, que podría ser continuada, cancelada, finalizada, etc.

Ilustración 51 - Mockup serie: sinopsis y estado

Por último, en el diseño de esta vista se encuentra el listado de temporadas de la serie.

Ilustración 52 - Mockup sserie: temporadas

Se ha diseñado como un listado horizontal que arrastrando de izquierda a derecha se recorran las temporadas para poder acceder a la que se quiera. Por lo tanto, cada elemento de la lista será un botón que lleve a la vista de la temporada pulsada.

Cada elemento botón de la lista contiene el póster, número y cantidad de episodios de cada temporada.

Conociendo mejor React Native, este será uno de esos componentes a diseñar reutilizables en cualquier parte de la aplicación.

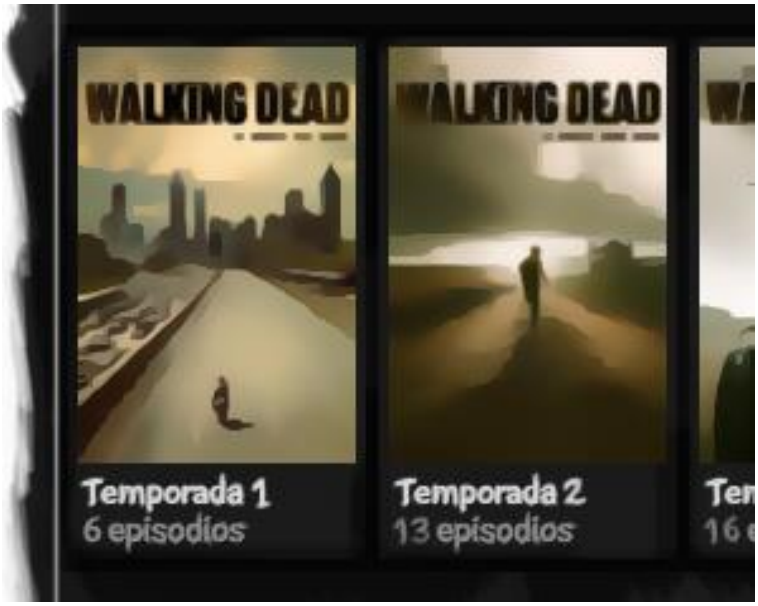


Ilustración 53 - Mokcup serie: detalle temporada

7.2.7. Mockup vista de una temporada

Esta será la vista donde se mostrará la información de una temporada junto a un listado con todos sus episodios. Se ha diseñado de manera que de fondo completo de la vista sea la imagen con un efecto blur y degradado a negro hacia abajo, y que tenga también un efecto parallax.

Ilustración 54 - Mockup: temporada y espisodios

El diseño es muy similar al de la vista de una serie y tiene sentido que así lo sea. En la cabecera se encuentra el poster de la temporada e información relativa a la misma y la sinopsis debajo.

Ilustración 55 - Mockup: episodios

Lo siguiente que se encuentra es el listado vertical de episodios desplegables.

Se espera mostrar el título, número y fecha en la cabecera, junto a un botón de más opciones.

Pulsando en esa cabecera, se desplegaría hacia abajo la sinopsis.

8. Funcionalidades

Ya se ha hablado de las funcionalidades principales que se quería implementar, pero vamos a recapitularlas para ver han quedado después de la implementación.

Funcionalidades API y clientes móviles

- Registro y autenticación de usuarios. La seguridad empleada es mediante `JWT` y habrá dos roles: usuario y administrador, para el acceso a distintos `endpoints` públicos y/o privados. Podremos profundizar más en la seguridad empleada en el apartado de Seguridad.
- Buscar series por nombre, tanto de series que haya en el sistema, como en APIs externas, filtrando por lo que se quiere buscar y dónde.
- Ver series en concreto, donde se podrá ver toda su información, incluyendo temporadas y episodios, marcar series como seguidas y puntuar series. Gracias a la puntuación, se podrá ver su puntuación media.
- Ver cada temporada de una serie en concreto, incluyendo póster, información y un listado de todos los capítulos que contiene con cierta información.
- Ver cada episodio de una temporada en concreto de una serie en concreto, incluyendo imagen y toda su información. Marcar cada episodio como visto o deshacer esta acción.
- De las series seguidas se puede marcar episodios como vistos, así como marcar una temporada completa como vista y marcar una serie completa como vista.
- Gracias al seguimiento de las series y marcación de episodios como vistos, se podrá llevar a cabo un seguimiento de cuántos capítulos hay sin ver de una serie que se sigue, o de las temporadas, etc.
- Solicitar series que no estén en el sistema haciendo una búsqueda por nombre de serie en APIs externas.
- Ver listados de series
    - Más populares en el sistema según visitas de usuarios
    - Mejor valoradas en el sistema según valoración de usuarios
    - Mejor ratio de tweet/hora en Twitter
- Cliente para Smartphones incluyendo iOS y Android, desde los que realizar todas las acciones anteriormente mencionadas.

Funcionalidades Administración Web

- Administración web responsive donde poder revisar las peticiones de las series que han hecho los usuarios para aceptarlas o denegarlas. También para volver a descargar datos corruptos de series, imágenes, actualizar datos e imágenes, etc., y desde donde poder actualizar el sistema en caso de ampliación o cambios en la implementación/base de datos.

8.1. API REST

En la API se encuentran todos los `endpoints` desde los que consumir, generar, actualizar o eliminar los distintos recursos. Como son muchos, se va a mostrar unos ejemplos de funciones principales. Se puede consultar el listado completo en Swagger o en el propio README del repositorio.

8.1.1. Funcionalidades

Autenticación

```
POST        /api/users/session
```

Petición – parámetros en el cuerpo

Nombre	Tipo	Descripción
email	String	**Requerido**. E-mail del usuario
password	String	**Requerido**. Primera mitad del resumen de la contraseña en SHA-512

```
POST /api/users/session

{
  "email": "user@email.com",
  "name": "hashedpassword"
}
```

Respuesta: 200 OK

```
// Status: 200 OK

{
  "ok": "user logged”, "type": "ok",
  "message": "Usuario identificado con éxito",
  "Authorization": "longaniza-de-JWT",
  "userId": 2,
  "userName": "User Name",
  "userRol": "u",
}
```

El token recibido en el campo `Authorization` es el JWT que se deberá utilizar como header con el mismo nombre añadiéndole la palabra clave `Bearer` para las peticiones que requieran autenticación: `Bearer <token>`

Respuesta: 401 Unauthorized

```
// Status: 401 Unauthorized

{
  "error": "incorrect email or password”,
  "type": "unauthorized",
  "message": "El correo electrónico o la contraseña no son correctos"
}
```

Listar todas las series

```
GET        /api/tvshows
```

Autenticación de nivel usuario necesaria. Se necesita proporcionar en esta petición el token de autenticación del usuario en la cabecera, en el header `Authorization` con el siguiente formato: `Bearer <token>`

Petición – parámetros en la query

Nombre	Tipo	Descripción
search	string ?= ""	Filtro de búsqueda.
tvdb	integer ?= 0	Si se debe buscar o no en la fuente externa The TVDB API

```
GET /api/tvshows

No content
```

Respuesta: 200 OK

```
// Status: 200 OK

[
  {
    "id": 1,
    "name": "series name 1",
    "firstAired": "2017-08-23",
    "banner": "./url/to/banner1.jpg",
    "score": 8,
    "voteCount": 10
  },
  {
    "id": 2,
    "name": "series name 2",
    "firstAired": "2017-08-23",
    "banner": "./url/to/banner2.jpg",
    "score": 4.3,
    "voteCount": 7
  },
  …
]
```

Observamos cómo se obtiene un listado de las series que hay en el sistema.

A continuación, un ejemplo usando esta misma ruta, pero con el parámetro de filtrado de búsqueda.

```
GET /api/tvshows?search=house

No content
```

Respuesta: 200 OK

```
// Status: 200 OK

[
  {
    "id": 3,
    "name": "House of Cards",
    "firstAired": "2013-01-31",
    "banner": "./assets/images/series/3/banner.jpg",
    "score": 7.1,
    "voteCount": 22
  },
  {
    "id": 4,
    "name": "House",
    "firstAired": "2004-11-15",
    "banner": "./assets/images/series/4/banner.jpg",
    "score": 8.2,
    "voteCount": 13
  }
]
```

En todos los casos del uso de este `endpoint`, si no se encuentra ninguna serie, se recibe la siguiente respuesta:

Respuesta: 404 Not Found

```
// Status: 404 Not Found

{
  "error": "Not found”
}
```

Estos son unos pocos ejemplos mostrados en profundidad, para verlos todos en profundidad se puede acceder a la Documentación Trending Series Swagger Offline implementada en GitHub Pages[31].

[31] GitHub Pages, servicio de hosting estático directamente desde el repositorio de un proyecto

Listado completo de endpoints

Aun así, se muestra a continuación un listado completo por de todos los endpoints de la API, **sin incluir los que requieren un administrador**, los cuales se verán en la sección de Administración.

Root		
GET	/	Estado de la API
TV Shows		
GET	/tvshows	Buscar series
POST	/tvshows	Crear una serie
GET	/tvshows/{id}	Obtener una serie
PUT	/tvshows/{id}	Modificar una serie
DELETE	/tvshows/{id}	Eliminar una serie
GET	/tvshows/tvdb/{tvdbid}	Obtener una serie de TVDb
Seasons		
GET	/tvshows/{id}/seasons	Obtener temporadas de una serie
GET	/tvshows/{id}/seasons/{number}	Obtener una temporada de una serie
Episodes		
GET	/tvshows/{id}/seasons/{sn}/episodes	Obtener episodios de una temporada
GET	/tvshows/{id}/seasons/{sn}/episodes/{en}	Obtener un episodio
Rating, Popularity and Trend		
GET	/tvshows/{id}/rating	Obtener voto de una serie
PUT	/tvshows/{id}/rating	Votar una serie
DELETE	/tvshows/{id}/rating	Eliminar voto de una serie
GET	/popular	Obtener series populares
GET	/toprated	Obtener series mejor valoradas
GET	/toptwitter	Obtener series mayor ratio en Twitter
Following		
GET	/tvshows/following	Obtener series seguidas por el usuario
GET	/tvshows/{id}/following	Obtiene si el usuario sigue una serie
PUT	/tvshows/{id}/following	Sigue una serie
DELETE	/tvshows/{id}/following	Deja de seguir una serie

Seen		
PUT	/tvshows/{id}/seen	Marcar serie como vista
DELETE	/tvshows/{id}/seen	Marcar serie como no vista
PUT	/tvshows/{id}/seasons/{sn}/seen	Marcar temporada como vista
DELETE	/tvshows/{id}/seasons/{sn}/seen	Marcar temporada como no vista
GET	/tvshows/{id}/seasons/{sn}/episodes/{en}/seen	Obtener si un episodio está visto
PUT	/tvshows/{id}/seasons/{sn}/episodes/{en}/seen	Marcar episodio como visto
DELETE	/tvshows/{id}/seasons/{sn}/episodes/{en}/seen	Marcar episodio como no visto

Requests		
POST	/requests	Solicitar una serie

Users		
POST	/users	Registrarse
GET	/users/session	Comprobar sesión
PUT	/users/session	Identificarse

Rutas totales usuario corriente: 32 rutas

Este listado pertenece a la documentación que se ha generado para el proyecto de la API en Play. En el apartado de documentación se puede encontrar más información al respecto.

8.2. Administración

Esta es la aplicación web que un administrador debe utilizar para revisar todos los datos de los recursos y, sobre todo, para aceptar o denegar las peticiones de las series que no estén en el sistema solicitadas por los usuarios.

Si cualquier recurso, ya sea una serie, una temporada, un capítulo, etc., contuviera errores de algún tipo, basta con pulsar un botón para volver descargar los datos en concreto que estén mal; lo mismo para actualizar datos que estuvieran desfasados.

Se trata de un diseño web totalmente responsive para que se adapta a cualquier dispositivo que pueda usar un navegador web como un ordenador, smartphone, tablet, etc. Basado en una plantilla llamada Limitless[32], se ha realizado todas las modificaciones necesarias para satisfacer las necesidades del proyecto y de los estilos deseados.

Se va a ver todos los endpoints para administradores y las vistas de esta aplicación web.

8.2.1. Endpoints administración

Además de que un administrador puede hacer uso de todos los endpoints vistos en API REST, se dispone de los siguientes:

TV Shows		
POST	/tvshows	Crear una serie
PUT	/tvshows/{id}	Modificar una serie
DELETE	/tvshows/{id}	Eliminar una serie
GET	/tvshows/tvdb/{tvdbid}	Obtener una serie de TVDb
Requests		
PATCH	/requests/{id}	Cambiar de estado una petición
PUT	/requests/{id}/newtvshow	Aceptar una petición y crear serie
Evolutions		
GET	/evolutions	Obtener todas las evolutions
PATCH	/evolutions	Aplicar una evolution determinada
Web Application views		
GET	/admin/login	Identificación
GET	/admin	Principal
GET	/admin/tvShows	Series y series eliminadas
GET	/admin/tvShows/:id	Serie, temporadas y episodios

[32] Limitless, Responsive Web Application Kit

GET	/admin/trending	Series populares, valoradas, tendencias
GET	/admin/tvShowRequests	Peticiones pendientes, aceptadas, rechazadas
Users		
GET	/users/session	Comprobar sesión
PUT	/users/session	Identificarse

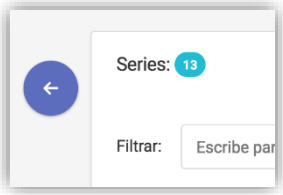
Además de este listado de endpoints a los que solo un administrador puede acceder, un administrador puede realizar cualquier otra acción, y también acciones que van incluidas en la propia aplicación web puesto que funcionan a través de las vistas de Play, las cuales trabajan directamente con los objetos JPA, como las funciones de redescargar datos de una serie o una temporada, episodios, etc., como se especifica en cada una de las vistas explicadas de aquí en adelante.

8.2.2. Características comunes

Toda la aplicación web contiene unas características que se repiten en todas las vistas.

- Diseño responsive
- Diseño claro, optimizado para la vista y minimalista
- Acciones tipo Ajax para no recargar la página y permitir la paralelización de las mismas
- Cabecera con información importante y avisos
- Miga de pan (breadcrumb) para saber en qué posición se está y cómo volver
- Tablas: responsive, filtrado, ordenación, paginación y número de elementos por página
- Botón atrás persistente en zona superior izquierda

En cuanto al filtrado de las tablas, todas ellas contienen una caja de texto donde poder introducir cualquier texto o número para filtrar o buscar. También disponen de ordenación por la columna que se desee, con tan solo pinchar en ella.



Y en cuanto al botón atrás persistente, se trata de un botón típico de volver atrás que conforme se hace scroll-down, el botón se adapta para permanecer arriba a la izquierda para no perderlo de vista.

Ilustración 56 - Administración: botón atrás persistente

8.2.3. Vista de Login

En esta vista, un usuario podrá identificarse, pero solo los que pertenezcan al rol de administradores tendrán la suerte de acceder a la aplicación web de administración.

Se ha guardado cierta similitud de estilo con los clientes móviles para mantener la línea de cara al exterior.

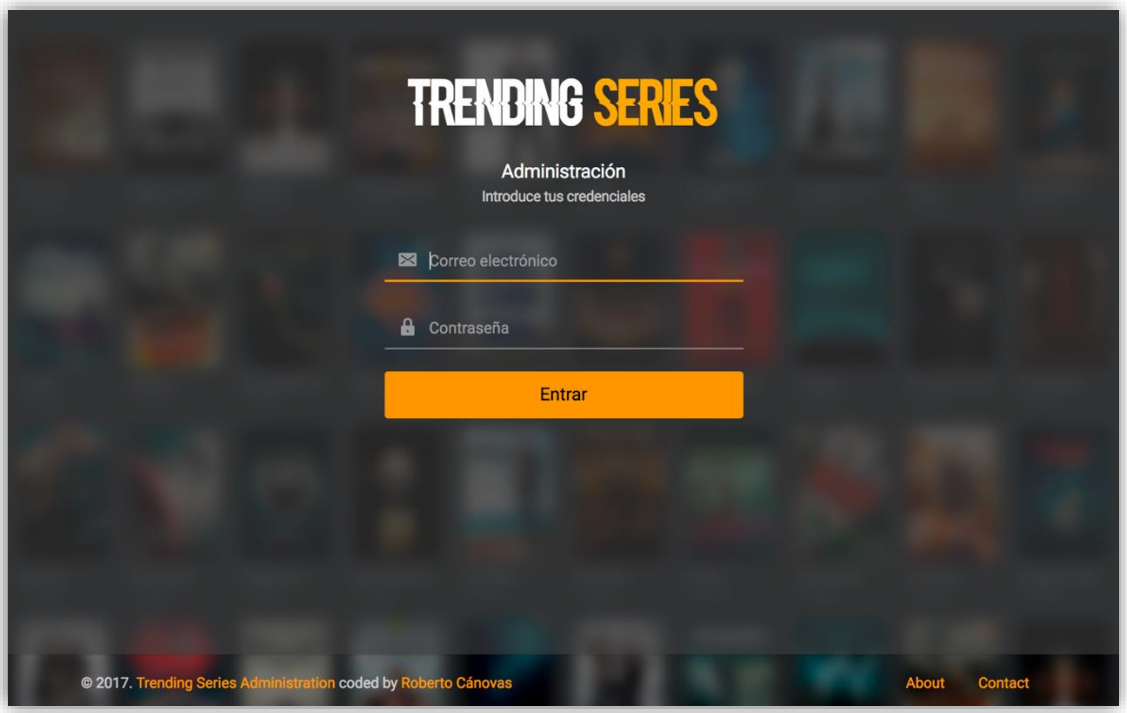


Ilustración 57 - Administración: vista login

Este formulario contiene validación de datos para el correo electrónico y la contraseña.

Una vez se hace login con un administrador válido, se accederá al panel de administración.

8.2.4. Cabecera

Primero se verá la cabecera de la aplicación web, pues será igual para todas las vistas internas de administración.

Ilustración 58 - Administración: cabecera

En esta cabecera se puede encontrar información relativa a la aplicación, al administrador identificado, a la posición actual en el panel en el que se encuentra el administrador y el **menú principal** que sirve para acceder a las distintas secciones principales.

Hay un panel de notificaciones donde aparecerán avisos de recursos pendientes, como, por ejemplo, si hubiese peticiones de series que todavía no han sido aprobadas o rechazadas.

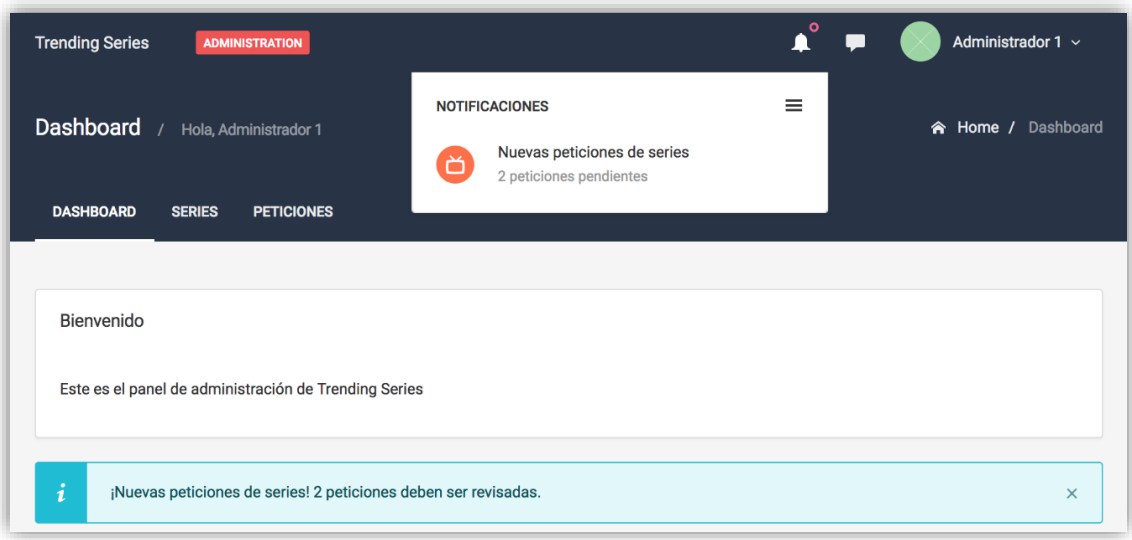


Ilustración 59 - Administración: notificaciones

También, aparecerá cuando esté disponible, una actualización del sistema o evolución de la base de datos.

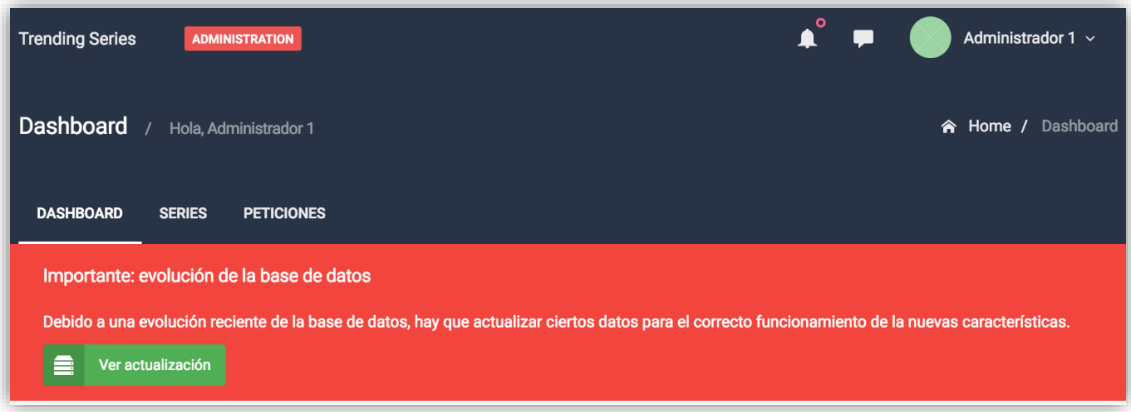


Ilustración 60 - Administración: aviso evolución del sistema

De esta manera, el administrador podrá ejecutar la tarea de actualización antes de continuar.

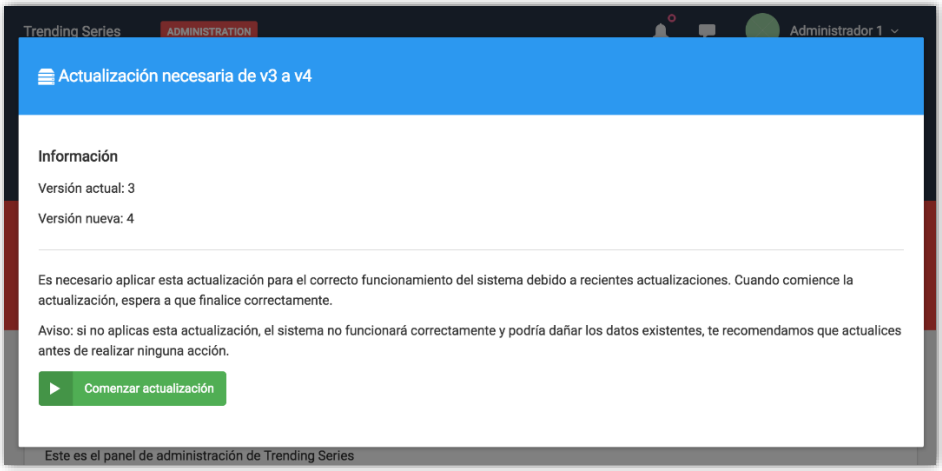


Ilustración 61 - Administración: detalles actualización

8.2.5. Vista principal

En esta vista, podemos encontrar información y datos generales de los recursos de los que dispone el sistema, así como avisos de recursos pendientes.

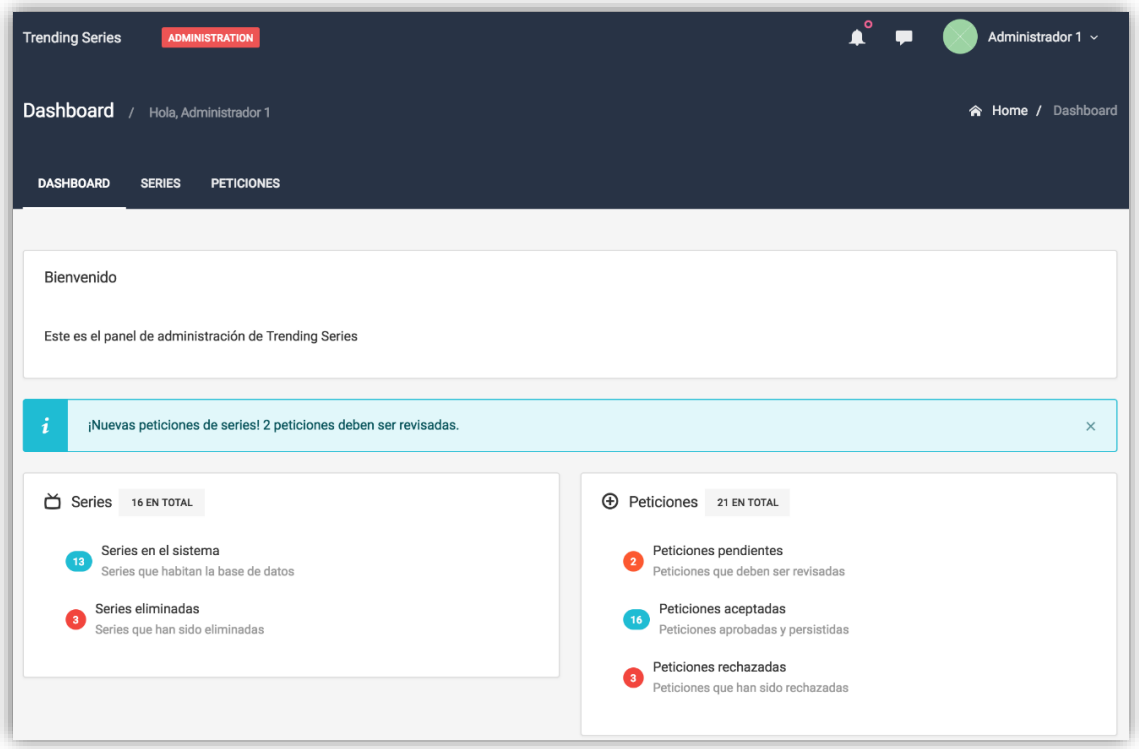
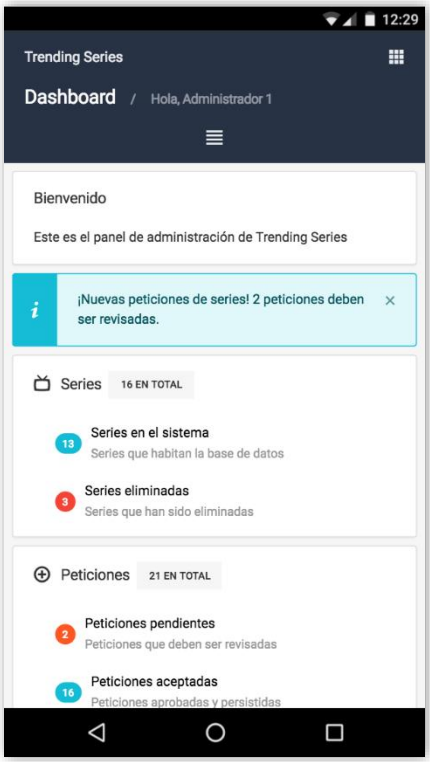


Ilustración 62 - Administración: vista principal

Se puede observar información relativa al número de recursos del que dispone el sistema en cuanto a series y peticiones, viendo de un vistazo rápido dónde haría falta que el administrador atendiera.



Así es la vista adaptada al navegador de un dispositivo con una pantalla reducida como la de un smartphone.

Gracias al haber implementado las vistas con un diseño responsive, las vistas del panel de administración se adaptan a cualquier tamaño de pantalla, para disfrutar de una buena experiencia de usuario se acceda desde donde se acceda.

De esta manera, un administrador podrá acceder al panel de administración desde un ordenador, desde una Tablet o un smartphone en cualquier momento y realizar las acciones pertinentes o controlar un poco la situación.

Ilustración 63 - Administración: vista en smartphone (responsive)

8.2.6. Vista series

Esta es la vista en la que se muestra un listado de todas las series que hay en el sistema, además de un listado de las series que han sido eliminadas.

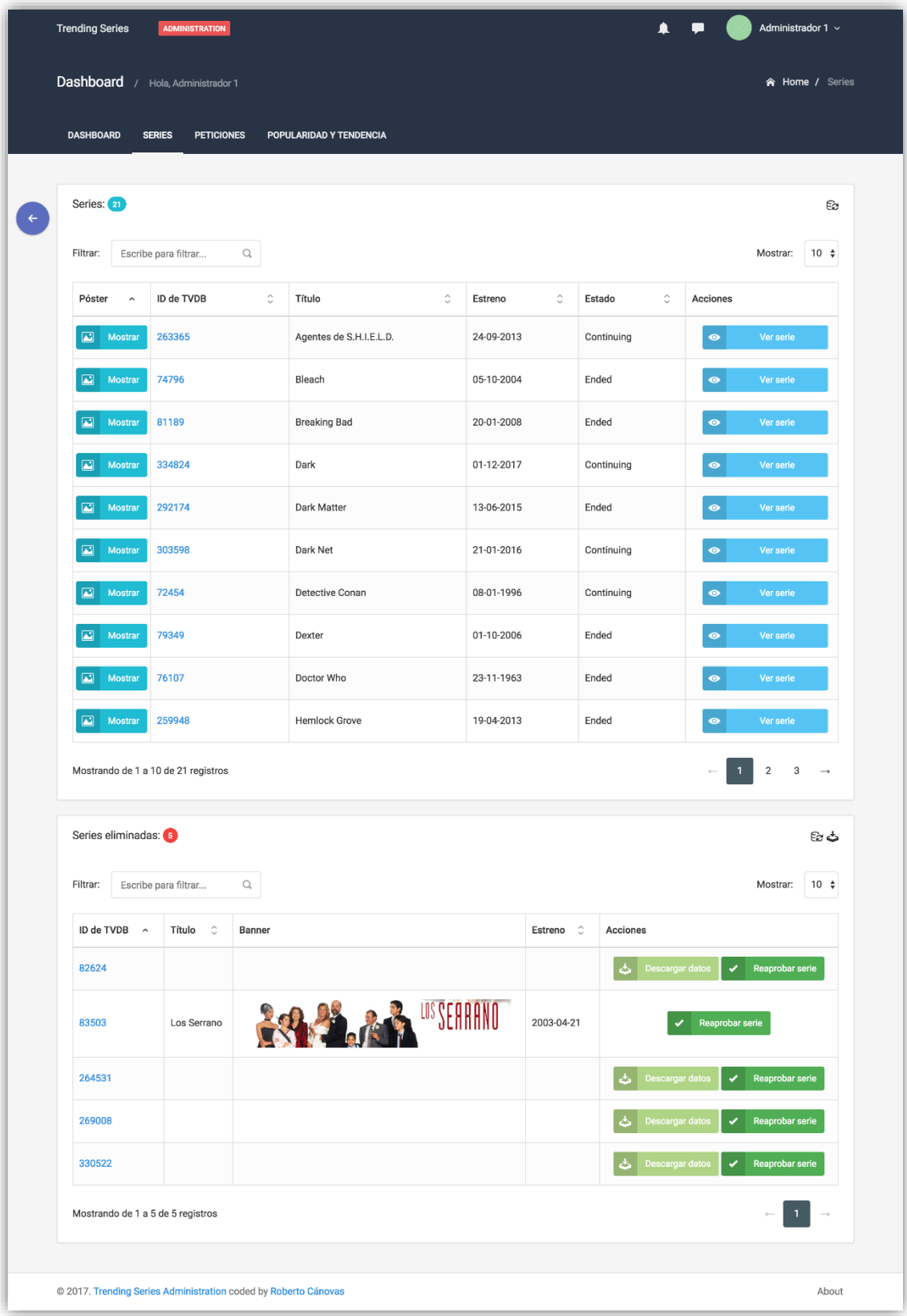


Ilustración 64 - Administración: vista series completa

Por un lado, se observa el listado de series en el sistema, donde se puede ver de cada una su información más general y su póster.

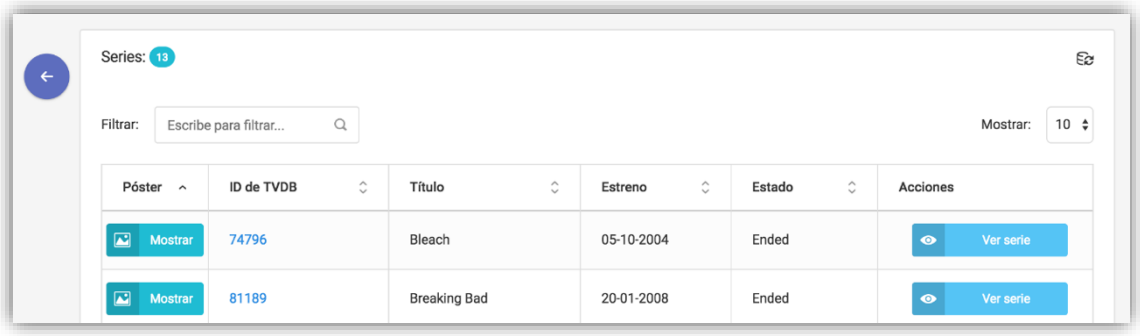


Ilustración 65 - Administración: lista de series

Además, la columna de póster, por defecto no carga todos los pósteres para evitar sobrecarga, y pulsando el botón **Mostrar** se revela.

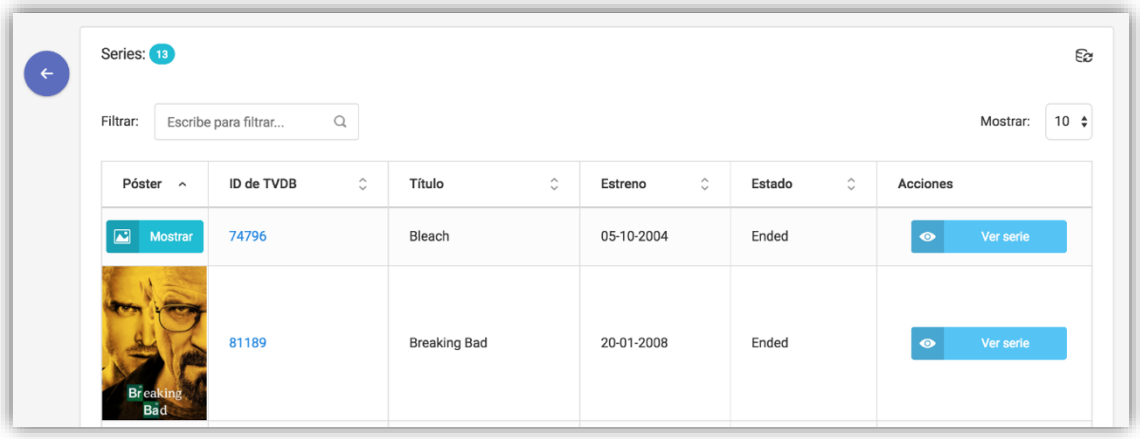


Ilustración 66 - Administración: lista de series (mostrar póster)

Como acciones se observa la de **Ver serie**, la cual nos llevaría a la vista de una serie en concreto, donde se encuentra toda la información sobre esa serie y donde se encuentran más acciones.

Por otro lado, está el listado de series eliminadas desde donde se pueden **reaprobar**. Al no tener su póster, con una petición a TheTVDB externamente, se obtiene su banner.

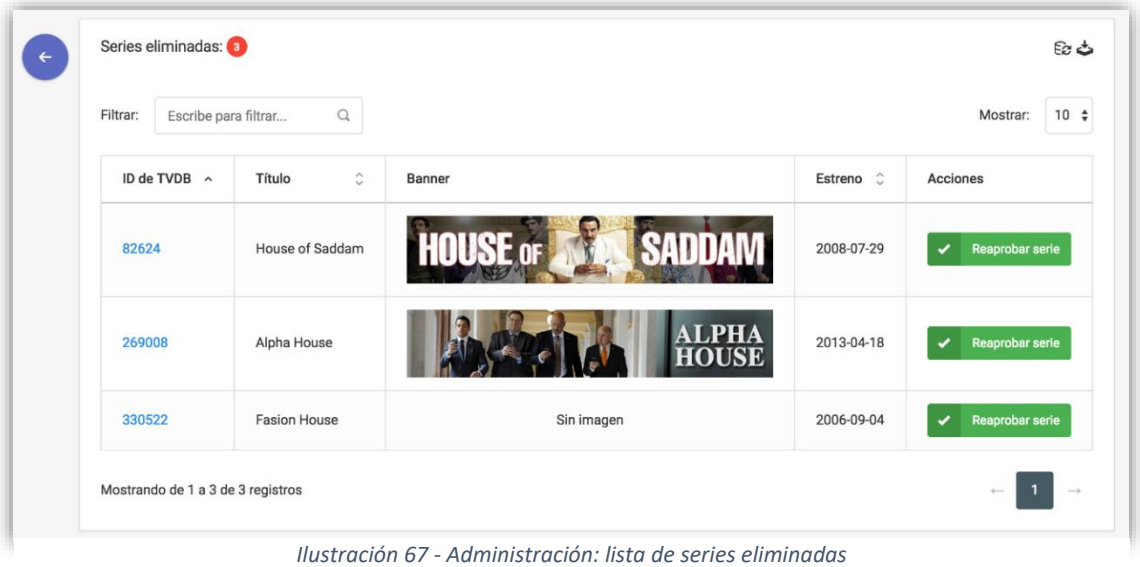


Ilustración 67 - Administración: lista de series eliminadas

8.2.7. Vista de una serie

En esta vista se encuentran todos los datos que el sistema alberga de una serie. No solo datos sobre la serie sino también sus imágenes, temporadas y episodios, y acciones para su mantenimiento.

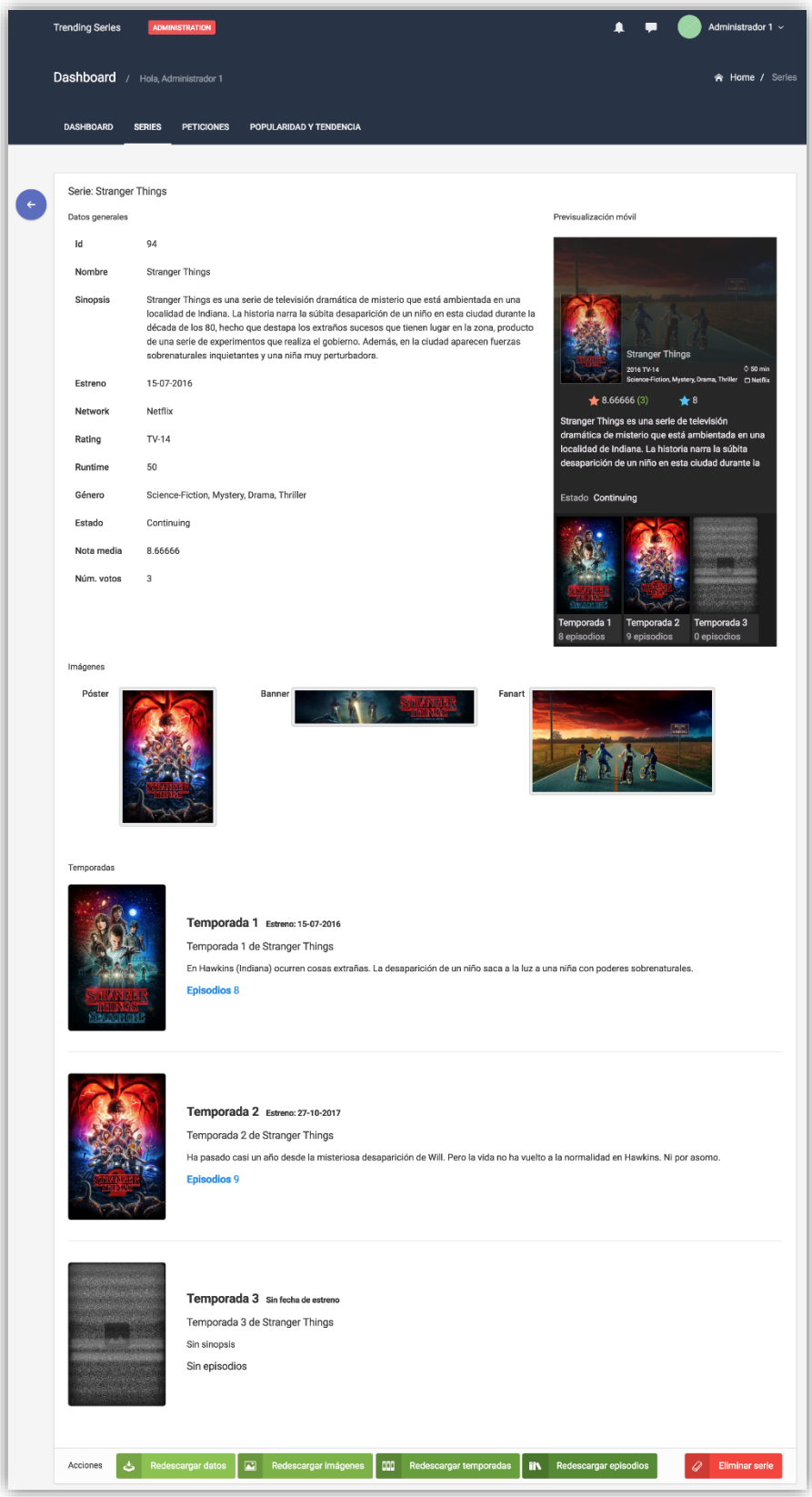


Ilustración 68 - Administración: vista serie completa

La parte superior, se divide en dos columnas; en la primera se puede observar la información principal sobre la serie como su id, nombre, sinopsis, etc., y en la segunda se observa una previsualización móvil, de cómo se vería todos estos datos en el cliente para iOS y Android.

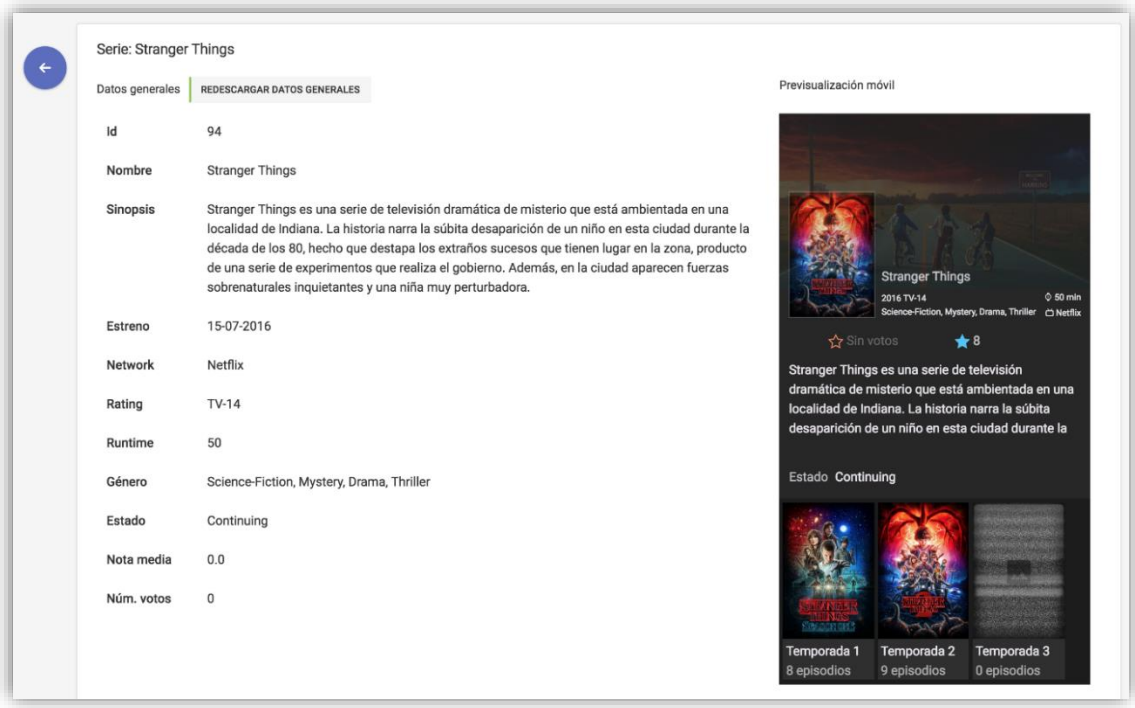


Ilustración 69 - Administración: datos generales de la serie

Además, encontramos un botón al lado del título de la sección **Datos generales** para poder redescargar estos datos principales de la serie.
En la siguiente sección, se encuentran las tres imágenes principales: banner, póster y fanart, las cuales son las imágenes más importantes de una serie y además son esenciales para el cliente móvil que se ha desarrollado en paralelo.

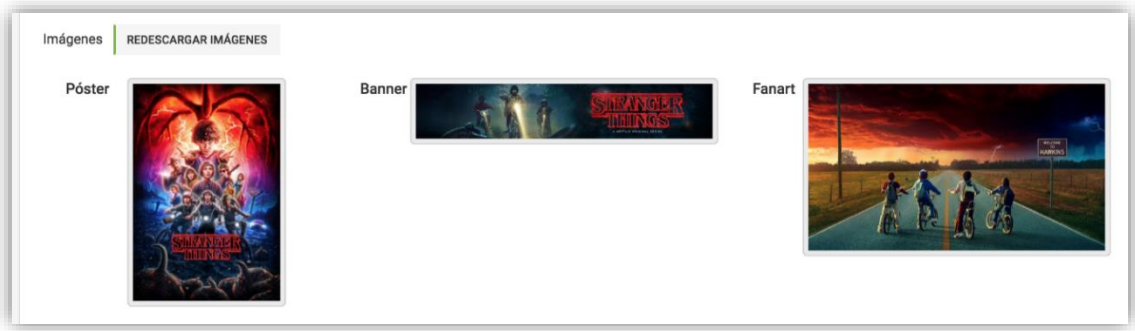


Ilustración 70 - Administración: imágenes principales de la serie

Esta sección también incluye un botón al lado del título de la sección **Imágenes** para poder volver a descargar esta información, en este caso, estas imágenes.

En la siguiente sección, titulada **Temporadas**, encontramos un listado de todas las temporadas de la serie que el sistema alberga. Contiene de cada temporada su información principal como su póster, número de temporada, nombre, fecha de estreno, sinopsis y cantidad de episodios.

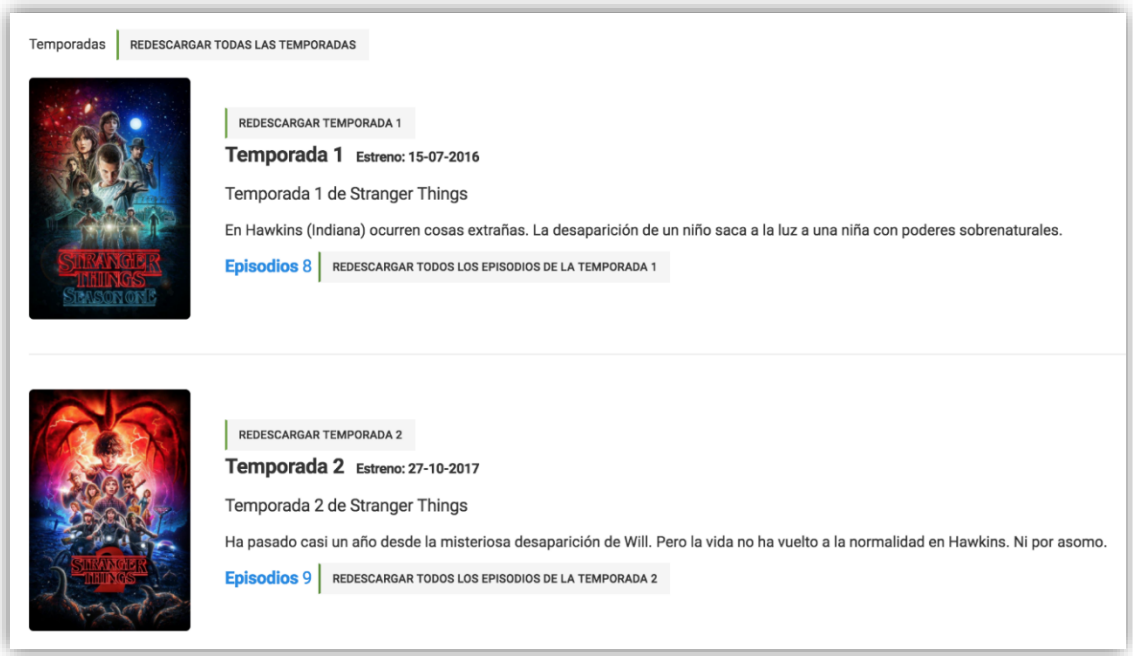


Ilustración 71 - Administración: temporadas

También contiene un botón para redescargar todas las temporadas, y además otro botón por cada temporada para redescargar solo una temporada en concreto.

Por último, pinchando en el contador de episodios de una temporada que aparece como un enlace (en este caso, **Episodios 9**), se mostrará un listado con todos los episodios de esa temporada con su imagen, nombre, fecha de estreno y sinopsis.

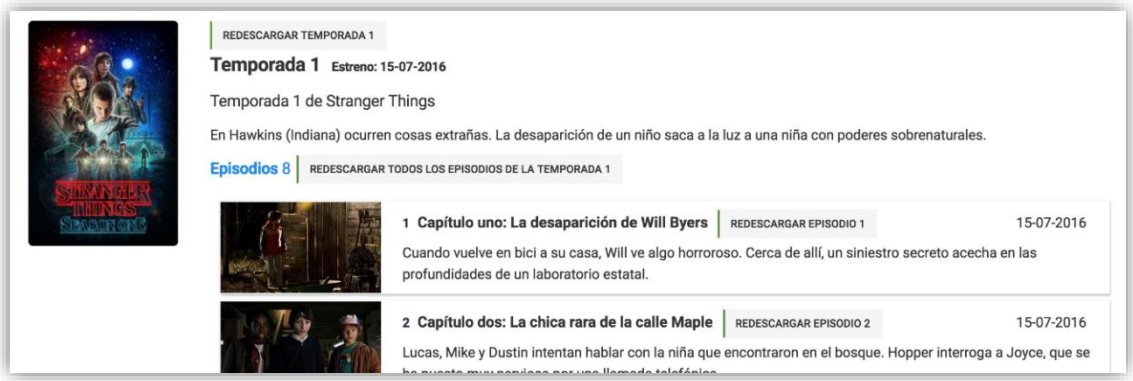


Ilustración 72 - Administración: episodios

Evidentemente, también existe un botón por cada episodio al lado de su título para volver a descargar solo los deseados.

Y finalmente, al pie, se puede encontrar unas acciones globales para la serie como la de redescargar los datos, las imágenes, las temporadas, los episodios y también una acción para **eliminar la serie**.

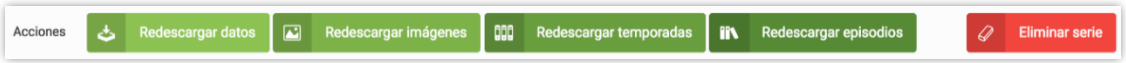


Ilustración 73 - Administración: acciones de serie

8.2.8. Vista de peticiones de serie

En esta vista contiene tres listados, uno de peticiones de series pendientes, otro de peticiones de series aceptadas, y otro de peticiones de series rechazadas.

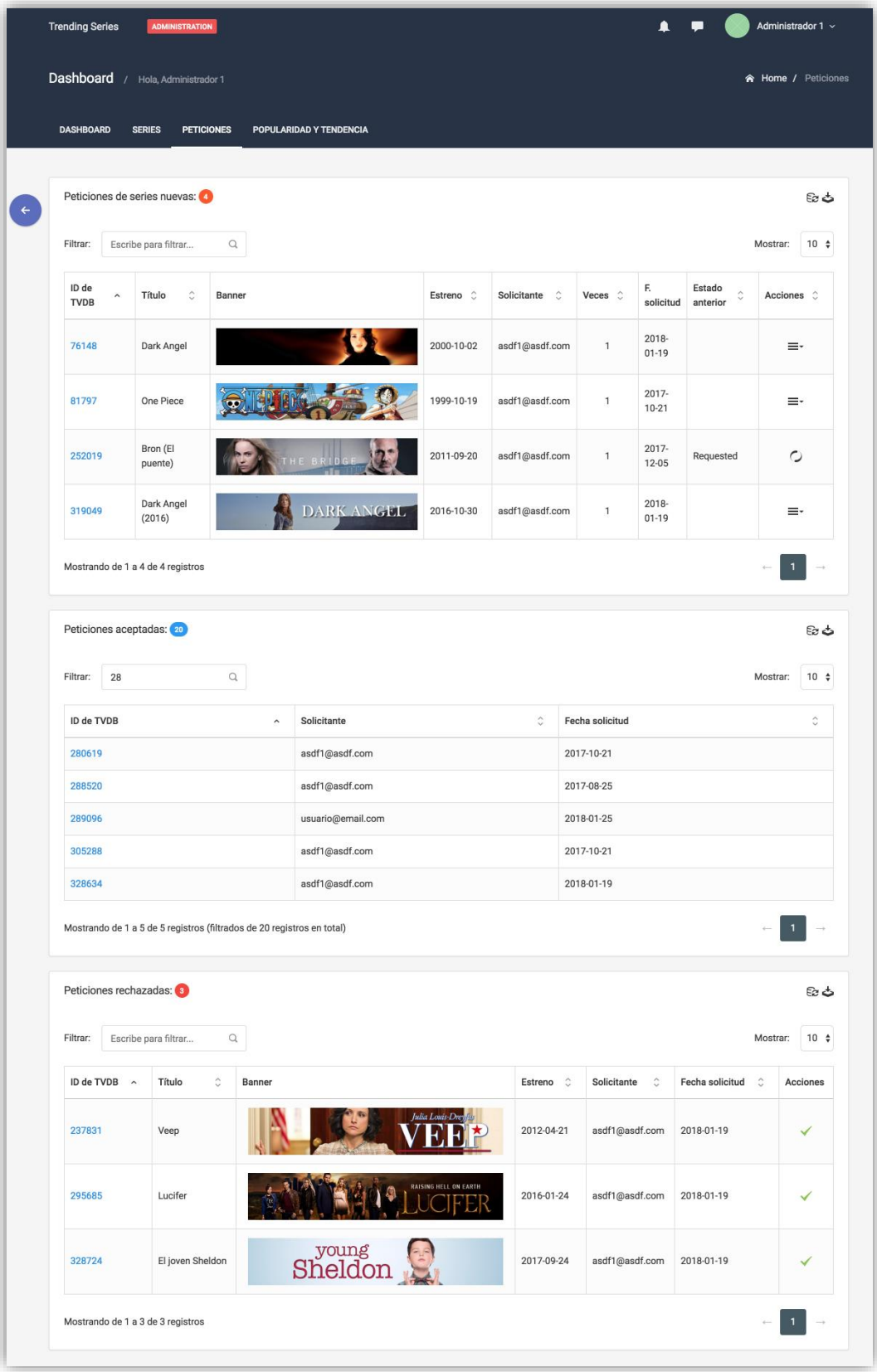


Ilustración 74 - Administración: vista peticiones completa

Como ya se ha dicho, en el primer listado se encuentran las peticiones de series pendientes, es decir, que no han sido ni aprobadas ni rechazadas, o que fueron rechazadas y los usuarios las han vuelto a solicitar. Para obtener la información de las series solicitadas pendientes se hacen peticiones a TheTVDB para obtener externamente su información principal.

Con ello se muestra un enlace a TheTVDB de cada serie, nombre de la serie, banner, fecha de estreno, qué usuario la ha solicitado, número de veces que a lo largo del tiempo ha sido solicitada, fecha de la solicitud actual, el estado anterior de la petición y una columna con acciones.

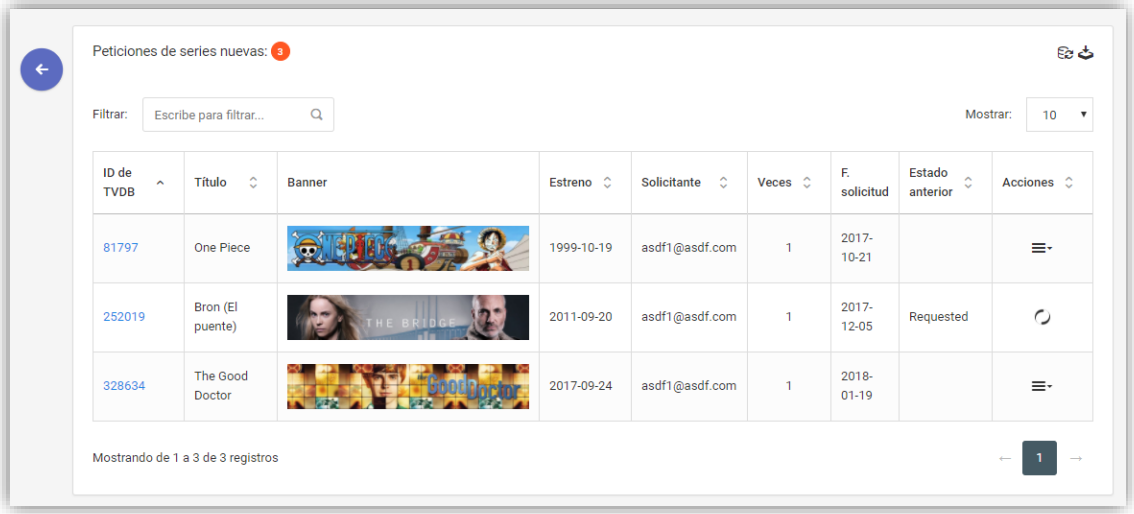


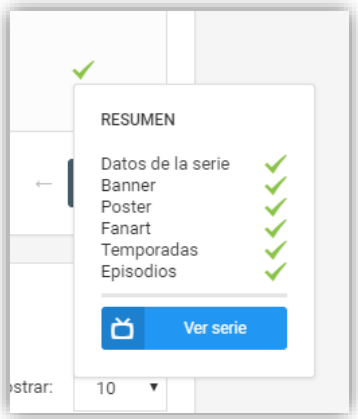
Ilustración 75 - Administración: lista peticiones pendientes

Ilustración 76 - procesando...

Si se pulsa en el botón de acciones se muestran dos: una para aceptar la serie y otra para rechazarla. En caso de que se aceptara una serie, el estado de la petición cambiaría a **Procesando** y el icono de acciones cambiaría a un indicador de actividad con una animación de giro constante.

Una vez que el sistema ha terminado el proceso de esa petición, el indicador de actividad terminaría su animación y cambiaría a un icono de resumen, indicando así el éxito o el fracaso de la operación.

Ilustración 77 - éxito



Si se pulsa en el icono de resumen, se muestra un listado con las operaciones que han funcionado y/o las que no, además de un botón para llevarnos a la vista de esa serie nueva.

Ilustración 78 - Administración: resumen acción

En el siguiente subapartado, se detalla en Obtención de datos externos, el proceso que lleva internamente la aplicación para llevar a cabo esta acción.

Después se encuentra el listado de peticiones aceptadas, en el que se muestra tan solo un enlace a TheTVDB de cada serie, quién la solicitó, la fecha de última solicitud y un botón para ir a la vista de serie de dicha serie aceptada.

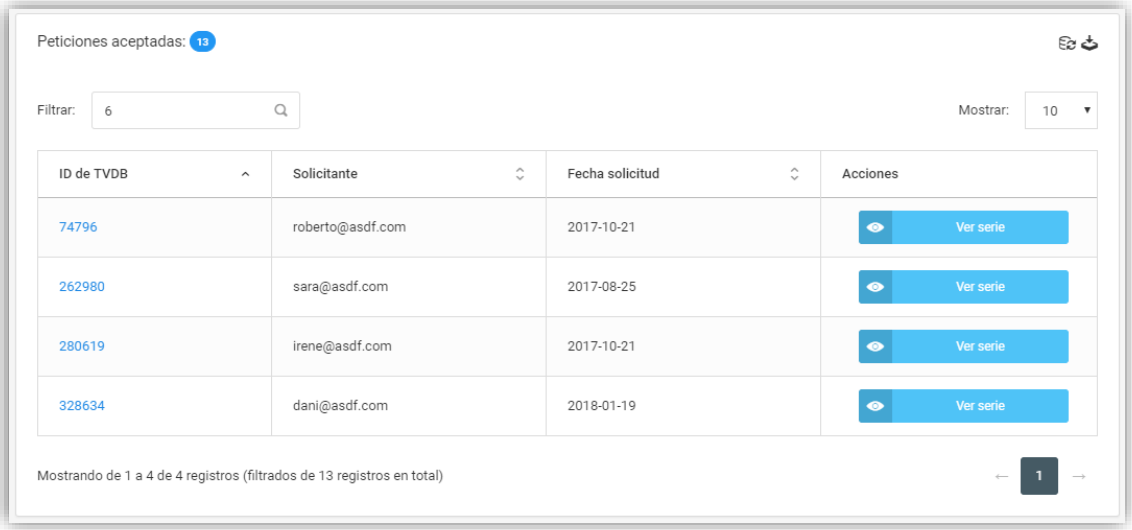


Ilustración 79 - Administración: listado peticiones aceptadas

Este listado, en un principio, está vacío. Solo cuando se pulsa el botón de descargar los datos de este listado, es cuando se cargan las peticiones de series aceptadas, ya que lo normal no es entrar a esta vista para ver el listado de series aceptadas, teniendo ya en la sección series todas las series aceptadas. Por lo que, para ahorrar recursos y cargas innecesarias, al entrar a esta vista, permanecerá este listado vacío hasta que se pulse el botón nombrado.

Por último, en el tercer listado, se muestran las peticiones de series que han sido rechazadas. También aparece vacío este listado, ya que, para cargar su información, hay que realizar peticiones externas a TheTVDB y no se suele entrar a esta vista para verlo. Una vez se pulsa el botón de obtención de datos, se muestra el listado completo con la información de las series de peticiones rechazadas.

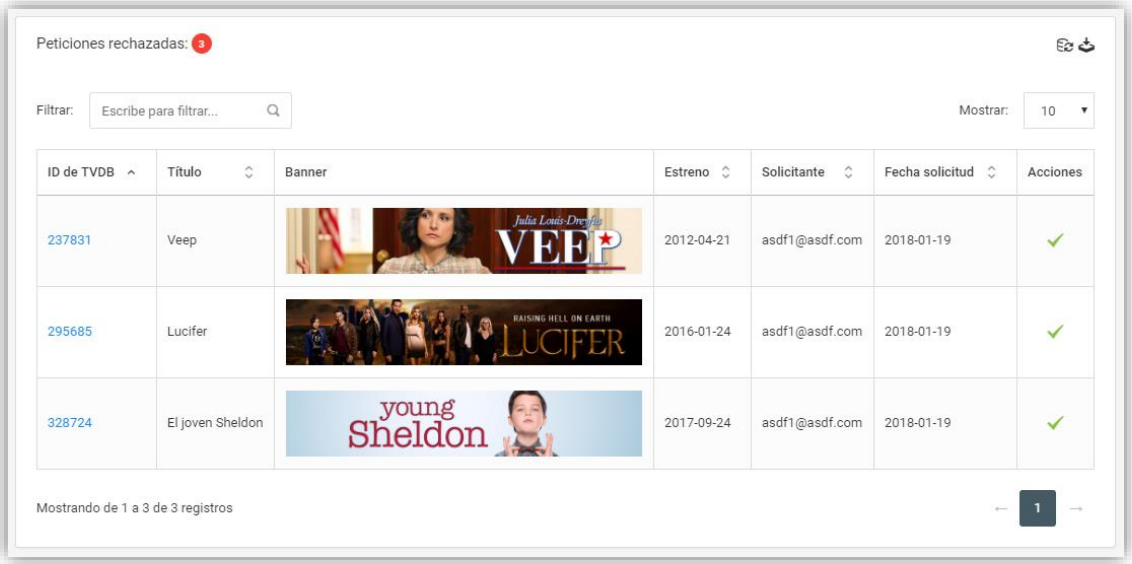


Ilustración 80 - Administración: listado peticiones rechazadas

Obtención de datos externos

Aquí se va a hacer hincapié en el proceso de obtención de los datos de una serie, explicando de qué forma la API se encarga de obtener externamente todos los datos necesarios de las series, así como de sus temporadas y episodios.

Todo este proceso se puede comprobar visualmente accediendo al log del sistema, en el que se indica paso a paso la obtención de cada dato, éxitos y fracasos de cada llamada, y qué servicio es el que está dando la información, para saber en cualquier momento qué se está ejecutando y qué funciona o qué falla.

Suponiendo que un usuario ha solicitado una serie que no está en el sistema, sólo queda que un administrador acepte la petición para que el sistema se encargue de todo, como hemos visto en esta vista de peticiones de serie.

Una vez el administrador acepta la petición, lo primero es comprobar que dicha serie existe realmente haciendo una petición inicial al servicio externo TheTVDB. De esta petición, si existe, se obtienen los siguientes datos de la serie:

- Id en TVDB
- Id en IMDb
- Nombre de la serie
- Cadena/productora oficial
- Sinopsis de la serie
- Rating de edad
- Duración aproximada de un episodio
- Fecha de estreno
- Géneros

Con estos datos la serie ya se persiste. El siguiente paso automático que hace el sistema es el de obtener las tres imágenes principales restantes también de TheTVDB:

- Banner – imagen principal de la serie horizontal
- Póster – imagen principal de la serie en vertical
- Fanart – imagen principal de la serie en 16:9

Lo siguiente es obtener las temporadas. Esta información se obtiene de TheMovieDb, por lo que se hace una petición a este servicio para obtener la serie usando el Id de IMDb que antes hemos obtenido. Lo que generamos con este paso es un objeto de tipo lista de las temporadas con su número de temporada en cada una, con una sola petición a TheMovieDb:

- Lista con todas las temporadas
    - Número de temporada

Una vez el sistema tiene las temporadas tan sólo con sus números de temporada y asignadas a su serie, se persisten.

En el siguiente paso, se recorren todas esas temporadas, y se solicita la información de cada una de ellas a TheMovieDb, además de obtener los pósteres de cada una de ellas, y lo persiste todo.

De cada temporada:

- Nombre de la temporada
- Sinopsis de la temporada
- Póster de la temporada
- Fecha de estreno de la temporada

Una vez el sistema tiene persistidas tanto la serie como sus temporadas, es hora de obtener los episodios de cada temporada. Para ello se recorre de nuevo las temporadas y se realiza una petición por cada una a TheMovieDb que nos devuelve toda la información de todos los episodios de una temporada en concreto.

De cada episodio:

- Número del episodio
- Nombre del episodio
- Sinopsis del episodio
- Imagen principal / captura del episodio
- Fecha de estreno del capítulo

Una vez obtenidos todos los episodios de cada temporada y asignados a sus temporadas, se persisten y ya tan solo queda actualizar la petición como aprobada y finalizada y devolver como respuesta las partes que han ido bien y las que han ido mal de todo el proceso.

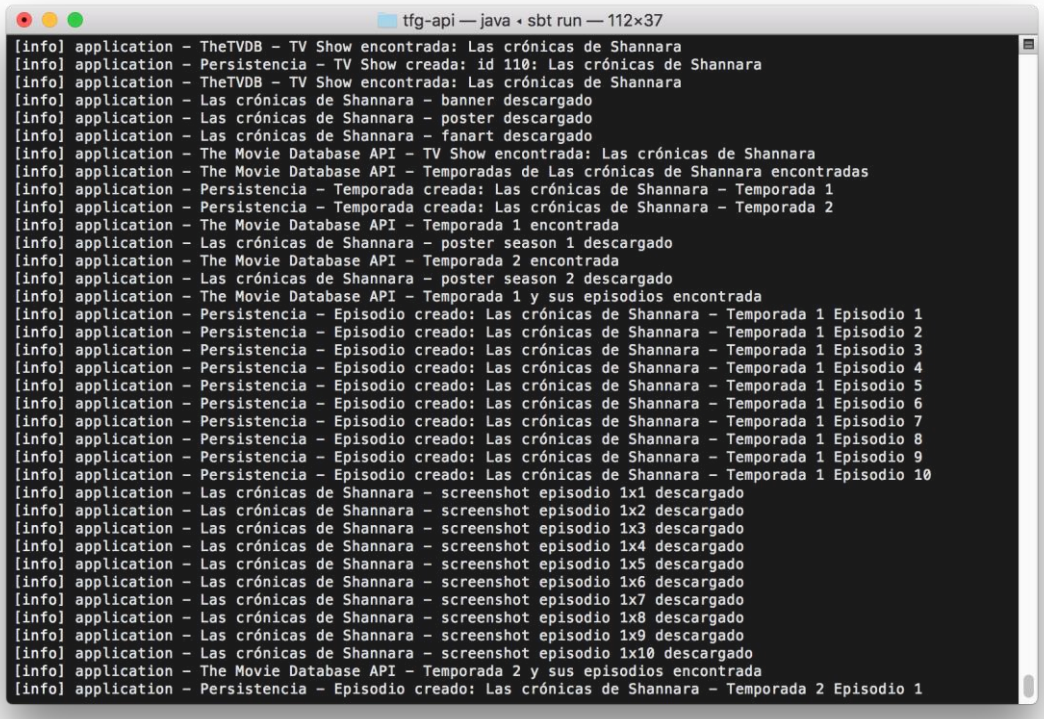


Ilustración 81 - Log de obtención de una serie

Si cualquier parte fuera mal, se puede volver a comprobar la petición o también se puede ejecutar por separado parte a parte para redescargar lo que hiciera falta desde el panel de administración.

Toda esta parte está implementada en distintos Servicios según su modelo y servicio externo (TvShow, Season, Episode, TVDB, TMDB, etc.), y dentro de cada Servicio está todo muy modularizado para mantener el código de manera eficiente, y poder reutilizar funciones para llamarlas individualmente por si hiciera falta solo una parte del proceso, o redescargar ciertas partes como se ha indicado antes.

De esta manera, si un servicio externo fallara, se podría implementar fácilmente otro que lo sustituya, incluso se podría implementar a la vez otros, para estar de sustitución automática en caso de que en cualquier momento uno falle (propuesta de característica futura).

Y así es cómo funciona la obtención de datos de una serie completa desde que un administrador la acepta. No se introduce ni un solo dato de serie a mano, ya que está todo totalmente automatizado de manera que el sistema crezca según se vayan solicitando series y se vayan aceptando.

La obtención de una serie al completo tardará en medida de los recursos de los que disponga la misma y de la velocidad de conexión tanto del servidor como del servicio externo. No es lo mismo obtener todos los datos de una serie que tiene tan solo una temporada con ocho episodios, que obtener una serie con quince temporadas y veinte episodios por temporada.

8.2.9. Popularidad, tendencia y mejor valoración

En esta vista se puede ver dos listados: uno de las series más populares y otro de las series mejor valoradas.

Estos listados son como los anteriores, responsive y con filtro de búsqueda, ordenación por el campo que se quiera y paginación.

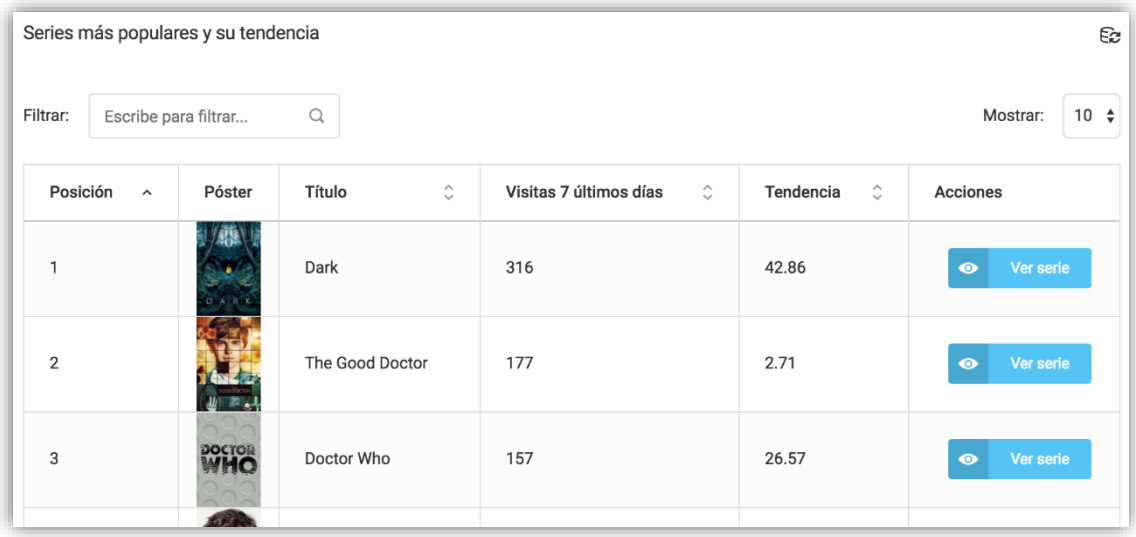


Ilustración 82 - Administración: popularidad y tendencia

En el listado de la ilustración anterior se muestran las series más populares que son las series que más vistitas están recibiendo en los últimos siete días junto a su tendencia, la cual es una media de los dos últimos días enfrentada al resto.

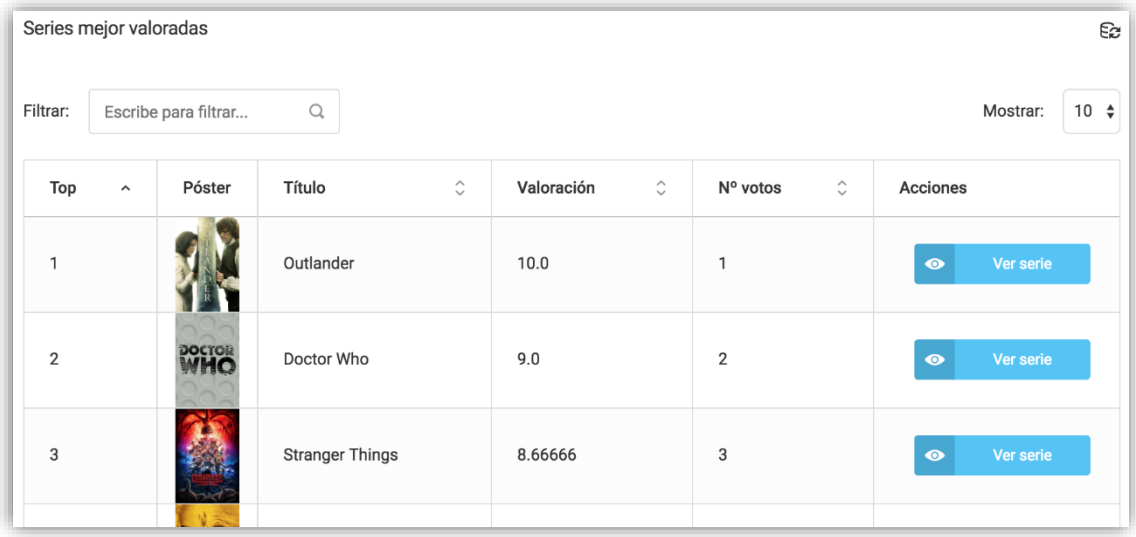


Ilustración 83 - Administración: mejor valoradas

En este, se muestran las series mejor valoradas donde se ve la media y el número de votaciones recibidas por parte de los usuarios.

En ambos listados se dispone de la acción **Ver serie** que enlaza con la vista de serie.

8.3. Front-end: Cliente móvil (iOS/Android)

El cliente para smartphones tanto para iOS como para Android, comparten implementación, ya que se ha realizado con React Native como ya se ha explicado. Las vistas en ambas plataformas son idénticas, salvo porque en cada una de ellas se utiliza la fuente de texto nativa, los iconos nativos, los indicadores de actividad nativos y los botones también nativos. Por lo demás, es exactamente igual, dicho en singular, porque no existen dos vistas por cada una.

8.3.1. Identificación y registro

En estas dos vistas un usuario se puede registrar o identificar rellenando los datos del formulario. Tiene validación de datos antes del envío de estos.

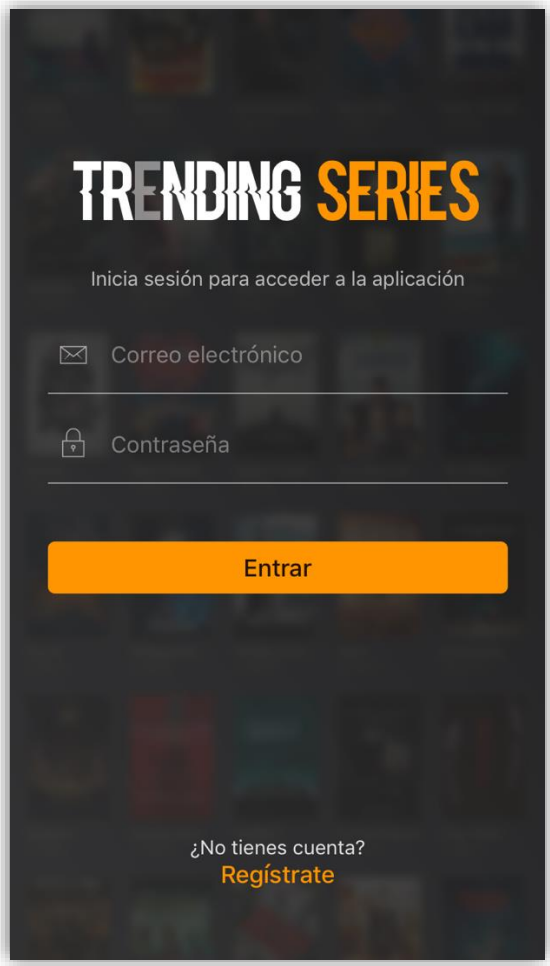


Ilustración 84 - Cliente: identificación

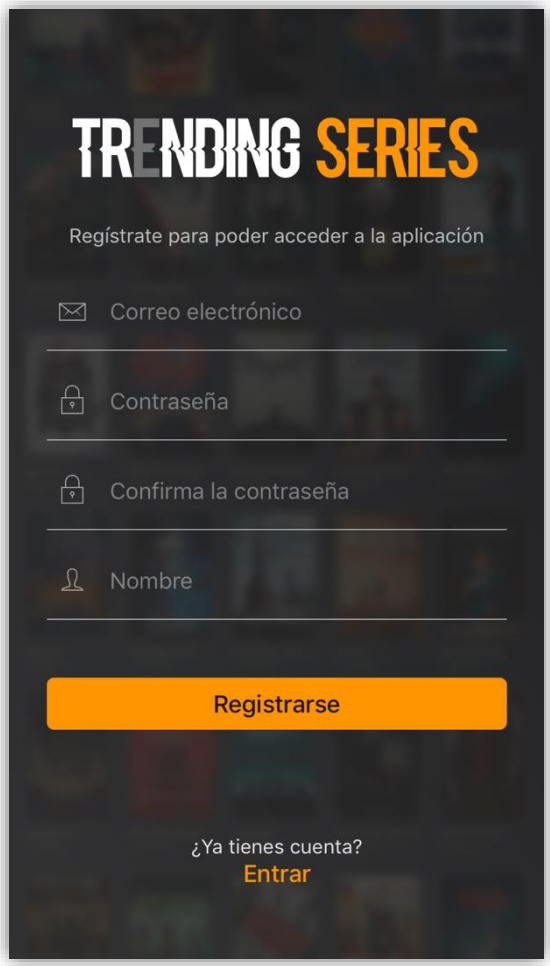


Ilustración 85 - Cliente: registro

El título, **TRENDING SERIES**, contiene una animación en la primera “E”, de parpadeo como si fuera una señal televisiva que está fallando, con rangos de parpadeo, intensidad, y opacidad aleatorios.

8.3.2. Principal

En esta vista un usuario puede ver las series más populares, las series mejor valoradas y tendencias en Twitter, en 3 listas de scroll horizontal, además de poder acceder a los listados completos de series más populares y mejor valoradas pulsando en **Ver todo**.

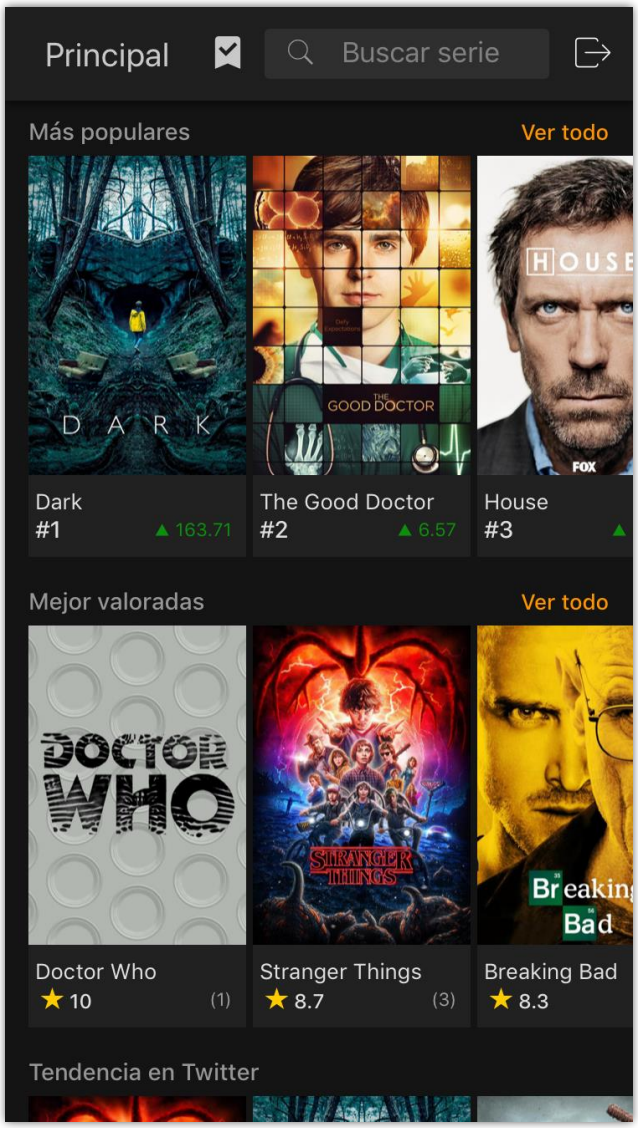


Ilustración 86 - Cliente: principal

Se puede acceder a Mis series con el botón del marcador que nos llevará a la vista de series que sigue el usuario, y mediante el botón de salir se puede cerrar sesión.

El cuadro de texto es el campo de búsqueda, pulsando en él, nos llevará a la vista de Búsqueda. Como se ha explicado en el apartado de diseño, se ha implementado una animación de expansión/contracción de este cuadro de búsqueda.

8.3.3. Búsqueda

Aquí un usuario podrá buscar series en el sistema para ver las que hay o entrar a cualquiera de ellas y ver su información, seguirlas, votarlas, etc.

Lo primero que se encuentra en la parte superior es un cuadro de texto de búsqueda, que a su izquierda contiene el botón para ir hacia atrás a la vista anterior, en este caso, la vista principal.

Se va a suponer que se quiere buscar la serie llamada “Dark”, así que se escribe el título de la serie en el cuadro de texto de búsqueda y se envía.

Se hace una llamada a la API y enseguida se muestra un listado con las series que contienen la palabra “Dark” en el sistema. Este listado es de scroll vertical que se esconde debajo del cuadro de texto de búsqueda.

Debajo del cuadro de texto de búsqueda aparece cuántos resultados se ha encontrado que coincidan con las palabras buscadas.

También se observa que para cada serie que se muestra en el listado aparece su imagen banner identificativa, su nombre, su año de estreno y su media de votaciones por parte de los usuarios.

Si se pulsa sobre cualquiera de las series, nos llevaría a la vista de serie de la misma.

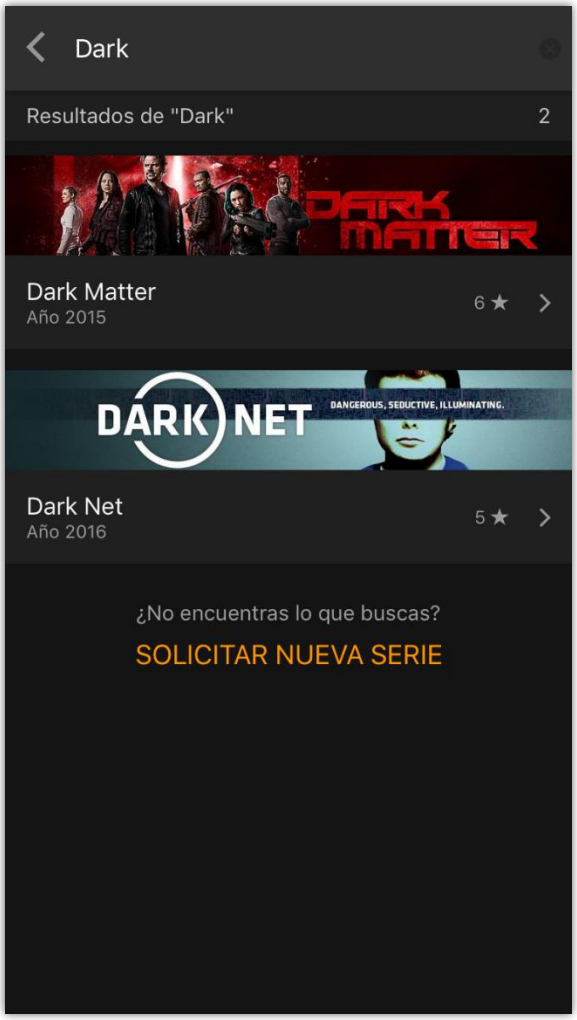


Ilustración 87 - Cliente: búsqueda

Si la serie que se busca no se encuentra en el listado, debajo del mismo hay un botón que sugiere al usuario **solicitar** una **nueva serie**.

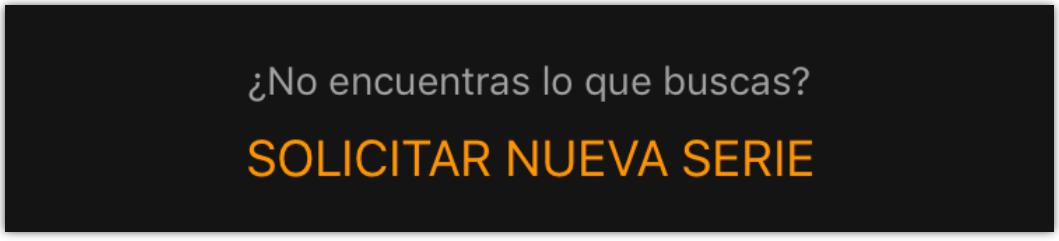


Ilustración 88 - Cliente: búsqueda - botón solicitar nueva serie

8.3.4. Solicitud de una nueva serie

Un usuario puede aquí buscar en servicios externos series por nombre.

Como en la búsqueda de la serie “Dark” en el sistema en la vista anterior no se ha encontrado, pulsando **SOLICITAR NUEVA SERIE** nos llevaría a esta vista, con el campo de búsqueda relleno con la palabra “Dark” automáticamente como se ve en la imagen.

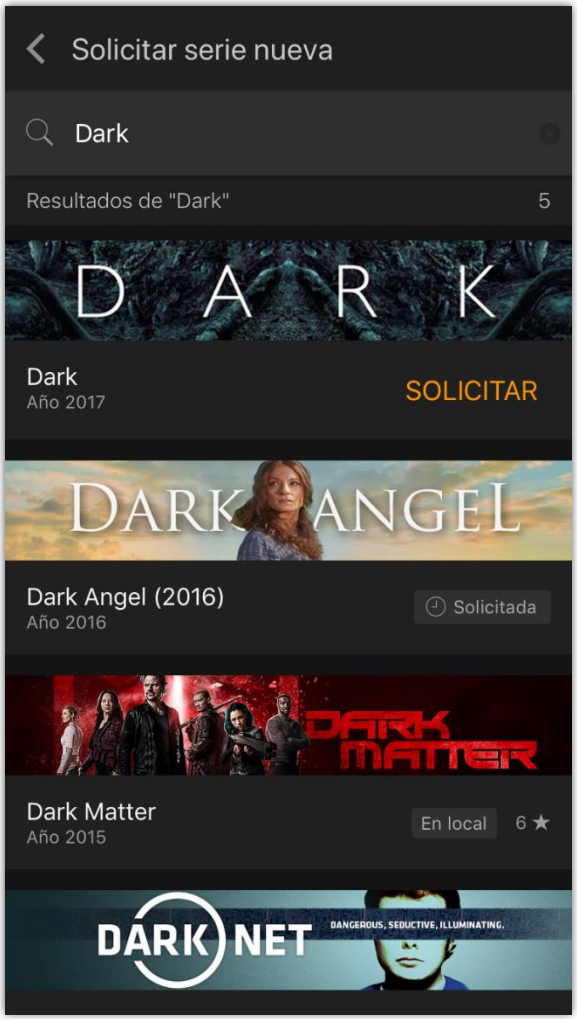


Ilustración 89 - Cliente: solicitar serie nueva

Al pulsar enviar, se hace una petición a la API, y a su vez la API hace una petición a un servicio externo, esta vez a TheTVDB, para obtener las series con dicho nombre y para contrastarlas con las que se tienen en el sistema.

Una vez contrastadas, envía los resultados en la respuesta al cliente, y una vez aquí, muestra el listado de series encontradas.

Como se puede observar, gracias al contraste que ha hecho el sistema, vemos las series que no están en el sistema que se pueden solicitar con un botón **SOLICITAR**, las series que están pendientes de aprobación o rechazo con un tag que contiene un reloj y la palabra **Solicitada**, y las que ya están en el sistema aparecen con un tag que contiene la palabra **En local** y muestra la votación media por parte de los usuarios.

Si se pulsa en la serie que indica que ya está en local, nos llevaría a la vista de una serie de la misma. Si se pulsa en la solicitada, nos indicaría que ya está solicitada a la espera de que un administrador la revise.

La búsqueda externa para la solicitud de la serie “Dark” ha sido un éxito ya que como se puede observar, el primer resultado es el de la serie que se busca. Al pulsar sobre este resultado, salta un modal pidiendo confirmación de solicitud, y al confirmar, un mensaje del resultado de la solicitud.

Ilustración 91 - Cliente: solicitar serie nueva - confirmar

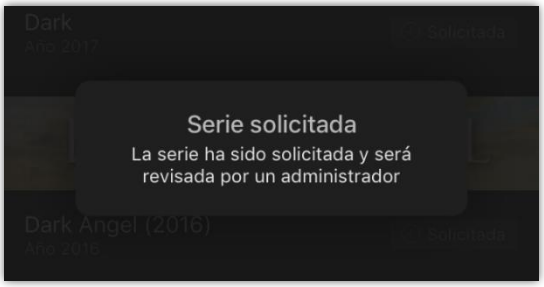


Ilustración 90 - Cliente: solicitar serie nueva - confirmado

8.3.5. Serie

Esta es la vista de una serie, que contiene toda la información relevante de la serie incluyendo datos generales, imágenes, valoraciones, temporadas, etc.

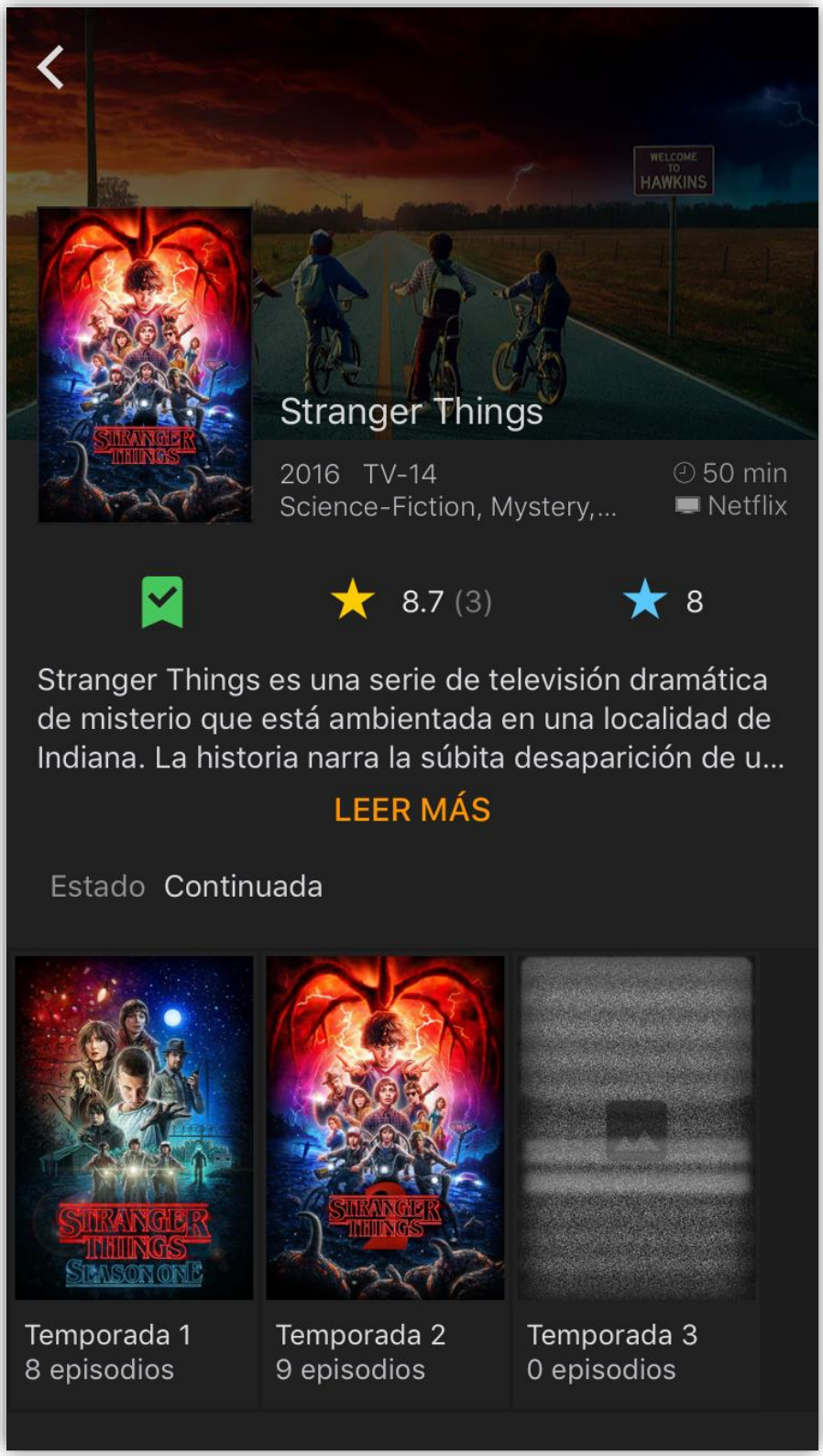


Ilustración 92 - Cliente: serie

En la parte superior de esta vista se encuentra una cabecera que contiene póster principal de la serie, nombre de la serie, año de estreno del primer capítulo, el rating de edad, el género, la duración media de los episodios y la cadena oficial y producción.

El fondo de esta cabecera es el fanart oficial actual con un degradado negro y un efecto parallax de scroll vertical que se especificó en la parte de diseño. Este efecto parallax hace que cuando se hace scroll-down, el contenido de la vista suba, pero la imagen fanart sube a un ritmo más lento, de manera que da la sensación de que esta imagen tiene una profundidad mayor al resto del contenido.

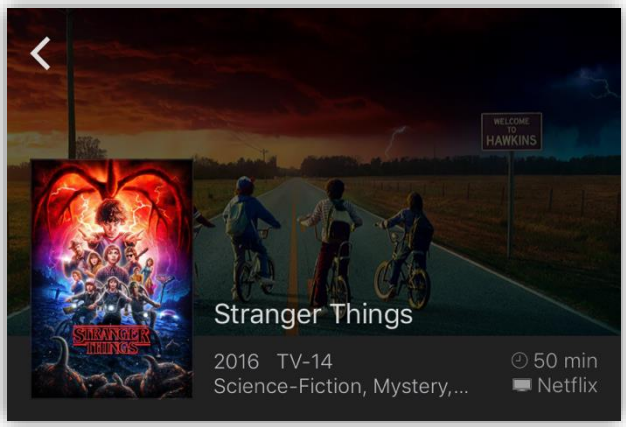


Ilustración 93 - Cliente: serie - cabecera

El siguiente apartado es seguir la serie y valoraciones de la serie.

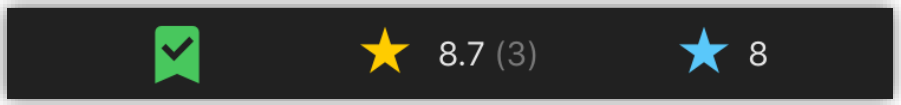


Ilustración 94 - Cliente: serie - acciones

- El marcador es el botón para seguir o dejar de seguir la serie. Si no la seguimos se muestra en gris y con un signo de suma dentro. Si se pulsa la serie, se marca como seguida y se pone verde y con un signo de check dentro, como se muestra en la ilustración anterior.
- La estrella amarilla corresponde con la nota media según la votación de los usuarios que además contiene un contador de votaciones totales. Si nadie ha votado la serie, la estrella aparece vacía.
- La estrella azul corresponde con la nota que el usuario activo ha calificado la serie. Si el usuario no ha votado la serie, la estrella aparece vacía. Esta estrella y nota es un botón, al pulsar sobre ella nos aparece un modal para realizar, modificar o eliminar la valoración.

Ilustración 95 - Cliente: serie - votar

Entonces, si se pulsa en la estrella azul aparece un modal.

En el modal aparece el título de la serie, el póster, un número con la votación actual y el slider de estrellas para poder elegir la votación arrastrando el dedo sobre él slider. También contiene dos botones de acción, **Guardar voto** el cual guarda la votación que esté en el slider y **eliminar voto**, la cual borra la votación que se tenga en el sistema por parte del usuario.

Si se pulsa guardar voto o eliminar voto, aparecerá un indicador de actividad animado y posteriormente un texto animado (degradado de entrada o degradado de salida) indicando la respuesta de la API, y se cerrará automáticamente el modal.

Arriba a la derecha del modal, se encuentra el típico botón en cruz para cerrar el modal manualmente, acción que también se puede realizar pulsando en cualquier parte de la pantalla que no sea el modal.

Debajo de las estrellas de valoración, se muestra la sinopsis principal de la serie.

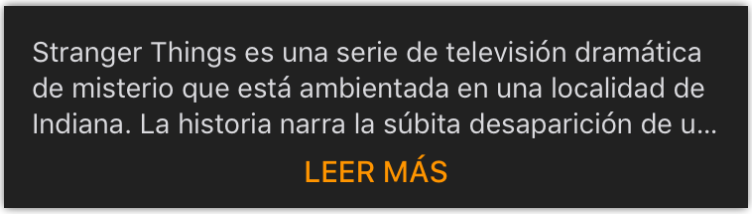


Ilustración 96 - Cliente: serie - sinopsis contraida

Si esta sinopsis tiene una longitud de más de tres líneas, aparece un botón **LEER MÁS** que al pulsarlo desplegará el resto de líneas de la sinopsis.

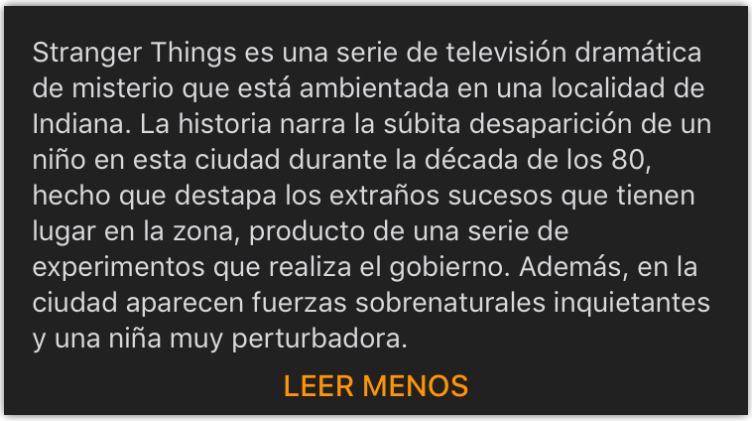


Ilustración 97 - Cliente: serie - sinopsis expandida

Del mismo modo, al expandir la sinopsis, el botón cambia el texto a **LEER MENOS** que realiza la acción contraria (contraer la sinopsis a tres líneas).

Lo siguiente que aparece es el estado actual de la serie, pudiendo ser o continuada o finalizada. Continuada indica que la serie está actualmente emitiéndose o ha sido renovada para una nueva temporada. Finalizada indica que la serie ha finalizado o ha sido cancelada.

Y, por último, las temporadas. Estas se muestran en un listado de scroll horizontal con una animación fluida y vistosa, agradable al tacto y a la vista.

Cada elemento de la lista es un botón de temporada, componente que se ha implementado en React Native usando componentes React Native y componentes nativos tanto de iOS como de Android. Un botón de temporada muestra el póster, número de temporada y cantidad de episodios de dicha temporada.

Ilustración 98 - Cliente: serie - temporadas

Si una temporada no tiene póster, como es el caso de la tercera temporada de Stranger Things como se puede observar en la imagen anterior, se muestra un placeholder en su lugar.

Si se pulsa sobre una temporada, se pasa a la vista de una temporada, mostrando su información.

8.3.6. Temporada

Al acceder a una temporada se carga esta vista, en la que se muestra toda la información relevante de la temporada. Es similar a la vista de una serie.

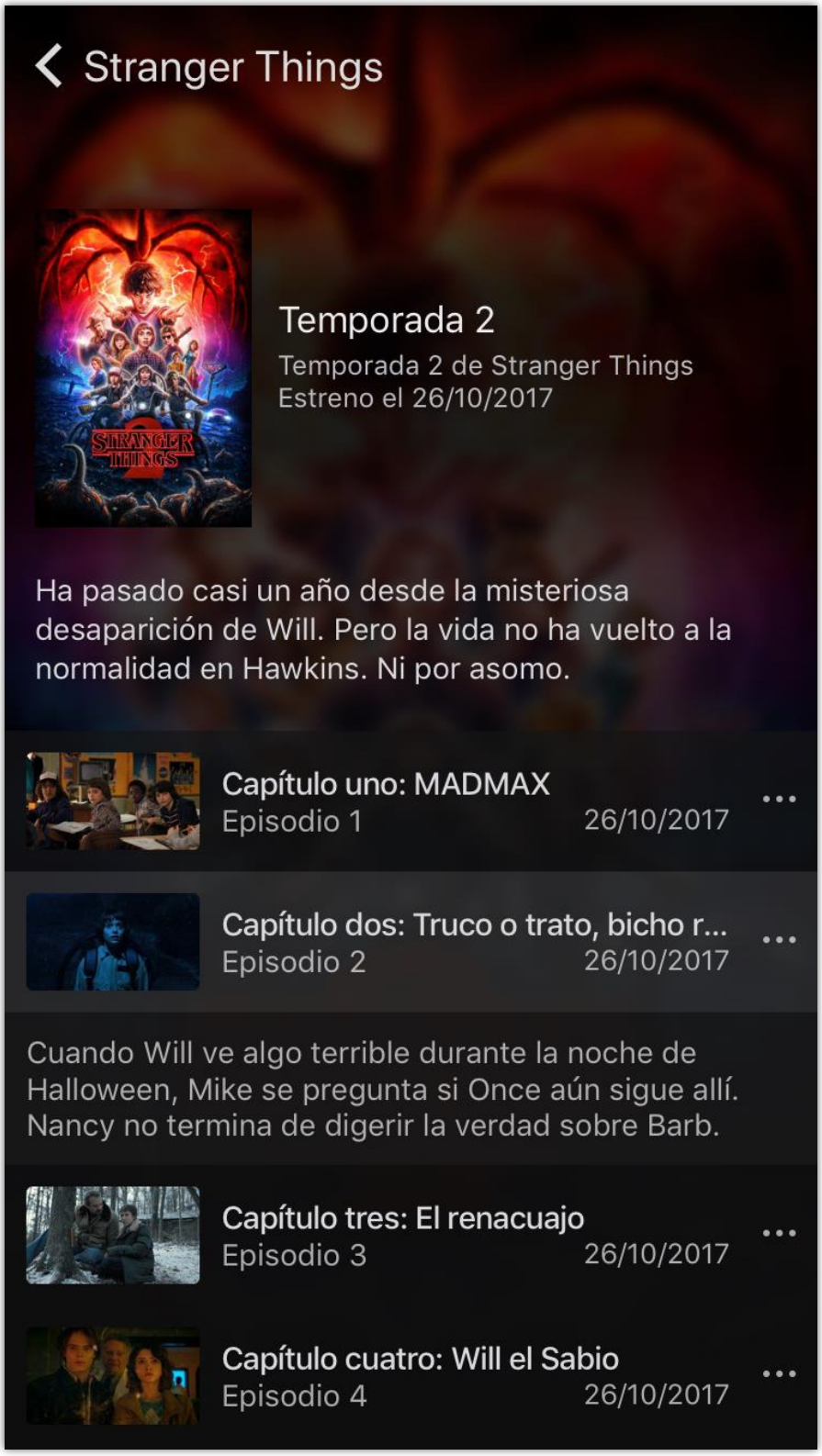


Ilustración 99 - Cliente: temporada

Como ya se explica en el apartado de diseño, el fondo de esta vista, es un fondo de tamaño completo del póster de la temporada ya que ocupa el 100% de la vista, al que se le ha dado un efecto blur, un degradado de transparente a negro verticalmente de arriba abajo, y que además cuenta con el efecto parallax cuando se hace scroll vertical implementado en otras vistas.

En la parte cabecera se encuentra la información principal de la temporada incluyendo el póster de la misma, título, número de la temporada y la fecha de estreno.

Ilustración 100 - Cliente: temporada -

Lo siguiente que se observa es la sinopsis de la temporada, que al igual que en la vista de una serie, es expandible/contraíble.

Contiene también un listado de scroll vertical que contiene componentes episodio contraíble con la información de los episodios.

Este componente, al igual que el componente botón de temporada, ha sido implementado para su reúso. En principio, muestra una captura del episodio, nombre, número y fecha de estreno del episodio (ej.: episodio 1).

Si se pulsa sobre un episodio, este se expande hacia abajo y muestra, además, la sinopsis del mismo, dándole un color más claro a la parte del título del episodio de manera que quede claro que está activo (ej.: episodio 2).

Ilustración 101 - Cliente: temporada - episodios

Si se vuelve a pulsar sobre un episodio expandido, se contrae para dejarlo en su estado inicial.

A la derecha de cada episodio, hay un botón *Más* de 3 puntos. Este botón sirve para realizar distintas acciones, aunque de momento la que está operativa es marcar el episodio como visto o como no visto (depende de su estado actual). Así pues, si se pulsa, se abrirá un Modal que preguntará para confirmar la acción, algo que se verá en la parte de seguimiento de series.

8.3.7. Series populares y series mejor valoradas

En esta sección se puede ver un listado de las series más populares del sistema. Basado en los datos que genera con cada petición de cada serie en la API, se puede saber cuáles son recientemente las más populares en el sistema.

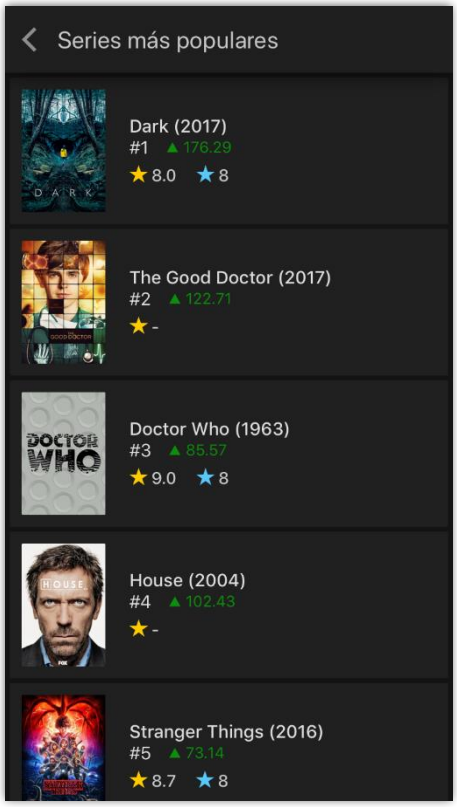


Ilustración 102 - Cliente: más populares

En la misma línea que el listado anterior. El listado está en el orden de más popular a menos.

Además de poder ver, como en la vista anterior, el póster, nombre de la serie y valoraciones, también se puede ver el año de la serie, su posición en el top y un número que refleja su ganancia de popularidad en visitas basada en una tendencia de los últimos días.

En cuanto al listado de las series mejor valoradas, es básicamente la misma vista, solo que aparecen de mejor valorada a peor valorada el listado de series.

También se incluye el año de estreno de la serie y sus valoraciones, tanto la media como la del usuario.

Pulsando en cualquiera de los elementos nos llevará a la vista de serie de la misma.

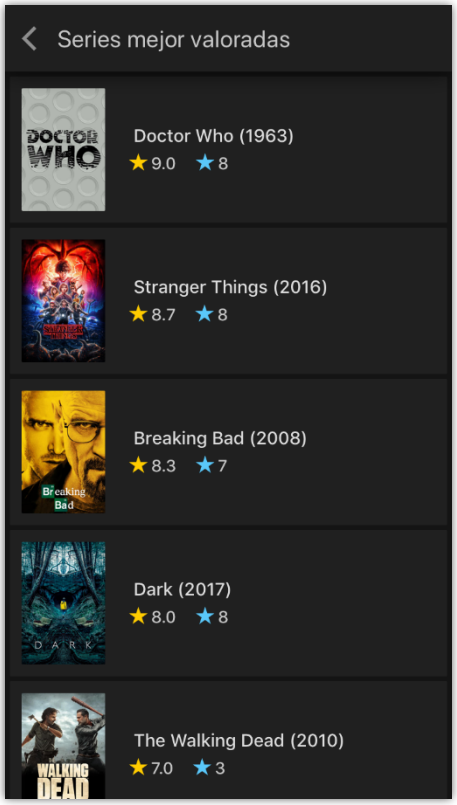


Ilustración 103 - Cliente: mejor valoradas

8.3.8. Seguimiento de series

Gracias al marcado de series como seguidas en conjunto con la marcación de episodios como vistos y no vistos, se puede realizar con seguimiento visual con muy buen resultado, de manera que en los posters de las series se ha decidido mostrar un indicador de cuántos episodios el usuario no tiene marcado como vistos, para aquellas series que tiene marcadas como seguidas.

Mis Series

Esta vista, muy similar a las listas anteriormente descritas, contiene las series que el usuario ha marcado con el botón de marcapáginas visto en la vista de una serie.

Se puede encontrar la nota media y la puntuación personal del usuario identificado. Además, a la derecha de cada serie, aparece el botón de marcador para poder dejar de seguir la serie desde este mismo listado.

Pulsando en el marcador de una serie, hará que se deje de seguir la serie, y el elemento de la lista desaparecerá con un efecto de desvanecimiento mientras que los elementos inferiores subirán también de manera animada poco a poco hasta ocupar el lugar que se ha quedado vacío.

Para poder volver a seguir una serie que se ha dejado de seguir, hará falta ir a su vista de serie para pulsar de nuevo el marcador de seguir serie.

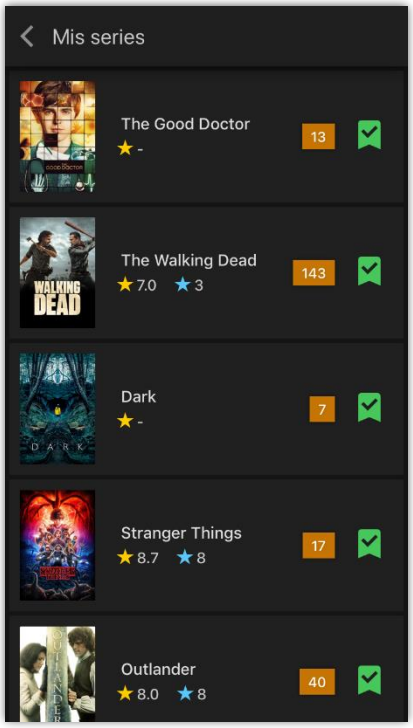


Ilustración 104 - Cliente: mis series

Indicadores de episodios

Como se ha dicho antes, una vez tenemos series marcadas como seguidas, aparecerán en sus posters el indicador numérico de cuántos episodios el usuario no tiene marcados como vistos.

Ilustración 105 - Cliente: indicador en póster

Se puede observar el indicador en la esquina superior derecha del póster de la serie *Dark*, indicando al usuario que tiene siete episodios sin ver todavía.

Deberá ir a esos episodios y marcar como vistos para que el indicador desaparezca.

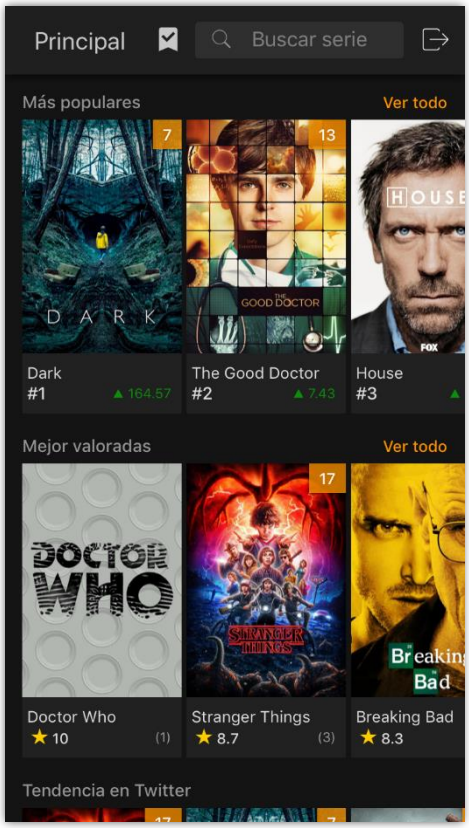


Ilustración 106 - Cliente: indicadores vista principal

Se puede encontrar marcadores en prácticamente todos los posters de todas las vistas, de manera que el usuario sepa para las series que sigue, qué episodios le faltan. Si quisiera que se deje de mostrar estos indicadores en una serie, bastaría con dejar de seguir la misma.

Se puede encontrar en los listados de la vista principal (ilustración de la izquierda), en Mis series, en la vista de una serie, en la vista de una temporada…

En el caso de la vista de una serie, hay un indicador en el póster principal de la serie indicando el número de episodios total de la serie que el usuario no ha visto, y también un indicador en el póster de cada temporada indicando el número de episodios sin ver de cada una de ellas.

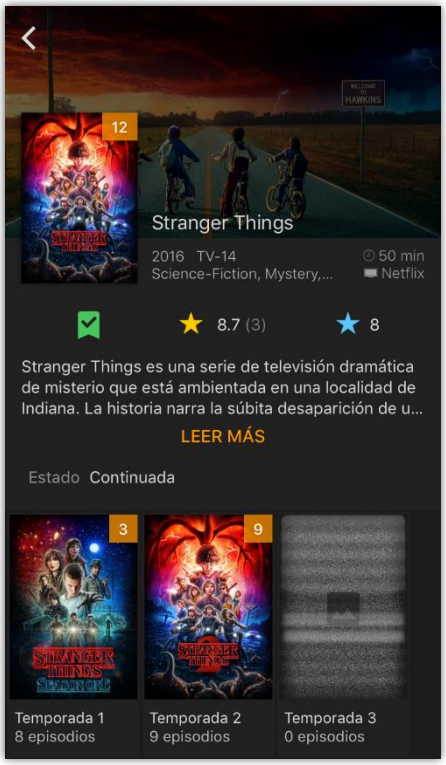


Ilustración 107 - Cliente: indicadores vista serie

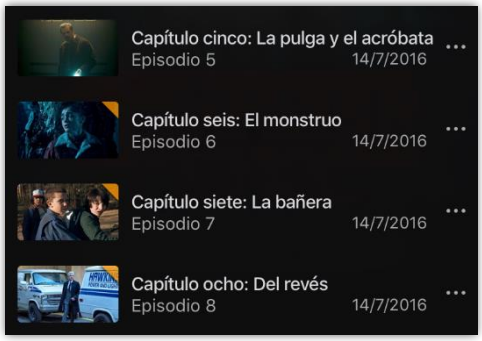


Ilustración 108 - Cliente: indicadores episodios

Y en el caso del listado de episodios, aparece un triángulo en la esquina superior derecha de la captura de imagen del mismo indicando que el episodio está como no visto (ej.: episodios 6, 7, 8)

Si un episodio se marca como visto, desaparecerá el triángulo.

9. Seguridad

En este apartado de seguridad, se hará un repaso de las medidas de las que dispone cada parte del proyecto para asegurar en todo momento los datos de los usuarios, los datos del sistema y cualquier intrusión, alteración o pérdida de los mismos.

9.1. Contraseñas

La contraseña un usuario, es la clave más importante del mismo, ya que es la que le proporciona el acceso al sistema. Para asegurar su integración y seguridad, se ha llevado diversas medidas en un proceso de encriptación determinado inspiradas en el *Doing it Right* de Taylor Hornby[33].

En primer lugar, el sistema nunca debe conocer la contraseña del usuario, ni siquiera resumida con una función hash[34]. Así que, se realiza un resumen de la contraseña utilizando la función hash SHA512[35], y se envía tan solo la primera mitad de este resumen, de manera que el servidor o cualquiera que pueda interceptarla durante la transferencia de la misma (*man-in-the-middle*[36]), no pueda conocerla al completo, además de que esta comunicación es asegurada por HTTPS/SSL.

En segundo lugar, al llegar la mitad del resumen, esta pasa por el proceso de encriptación PBKDF2[37] con HMAC[38] SHA512 (el más reciente que soporta Java), junto a una salt para evitar que, aunque dos contraseñas sean iguales, encriptadas sean distintas. El proceso de encriptación es el siguiente:

- Generación de Salt[39] de 24 bytes mediante SecureRandom[40]
- Resumen hash de 18 bytes mediante PBKDF2 con HMAC SHA512 de 64.000 iteraciones
- Unión de todas las partes separadas por ":" con el siguiente formato
    - alg – algoritmo utilizado, en este caso sha512
    - PBKDF2_ITERATIONS – iteraciones de PBKDF2 (64.000 en este caso)
    - hashSize – tamaño del hash resultante
    - saltBase64 – salt generada codificada en base 64
    - hashBase64 – hash resultante codificado en base 64
    - `alg:PBKDF2_ITERATIONS:hashSize:saltBase64:hashBase64`

La cadena resultante de la unión anterior, es lo que se guarda en la base de datos en un solo campo.

Para verificar una contraseña, el sistema necesita primero obtener el hash correcto almacenado en la base de datos, de donde sacará los parámetros unidos anteriormente. Una vez el sistema tiene todas las partes, somete a la contraseña introducida la misma encriptación. Finalmente, ambos hashes se comparan para ver si son iguales.

Para hacer esta comparación, se ha implementado una función que compara dos vectores en *length constant time*[41], para evitar casos de fuerza bruta y otros ataques basados en el tiempo de proceso.

---

[33] Doing it Right – Guía referencial de cómo aumentar la seguridad de las contraseñas y de qué no hacer
[34] Función hash – Función de resumen computable mediante un algoritmo determinado
[35] SHA512 – Función hash perteneciente a SHA-2 de 512 bits
[36] Man-in-the-Middle – Ataque que permite acceder y/o manipular situándose entre el origen y el destino
[37] Password-Based Key Derivation Function 2 – Función derivada con coste computacional variable (RSA)
[38] Hash-Based Message Authentication Code – implica función has con clave secreta para verificar datos
[39] Salt – Sal o semilla, compuesta de bits aleatorios para derivar una clave o contraseña aleatoriamente
[40] SecureRandom – Clase que proporciona robustos generadores criptográficos de números aleatorios
[41] Length constant time – Comparación de dos elementos de principio a fin, aunque se encuentre el fallo durante el proceso

9.2. JSON Web Tokens

Es un estándar abierto[42] basado en JSON para la creación de tokens de acceso que contienen una serie de claims de manera segura entre dos partes.

Dicho de otro modo, el sistema genera un pase para un usuario determinado con unos privilegios o características determinadas, y solo este usuario podrá utilizarlo para acceder a las partes del sistema que el pase le permita.

Los JWT se componen principalmente de tres componentes: Header, Payload y Signature.

- Header – incluye datos referentes al algoritmo y tipo de token y se codifica en base 64
  - alg – algoritmo utilizado
  - typ – tipo de token
- Payload – alberga los distintos claims y se codifica en base 64
- Signature – se concatena el header y payload codificados en base 64 con un "." y se encripta con HMAC SHA256 con a una clave secreta

Juntando las tres partes concatenadas con un "." se obtiene el JWT completo.

Así pues, el algoritmo que se usa en este caso en HMAC SHA256, el tipo de token es JWT y los claim que se han implementado para el payload son el email del usuario y el rol.

Un ejemplo claro de toda la estructura sería:

Sin codificar

HEADER

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD

```
{
  "emailClaim": "roberto@email.com",
  "rolClaim": "a"
}
```

SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```

Codificado

```
eyJhbGciOiJIUzI1NiIsInR5cCI6Ik
pXVCJ9.eyJlbWFpbENsYWltIjoicm9
iZXJ0b0BlbWFpbC5jb20iLCJyb2xDb
GFpbSI6ImEifQ.DRkYKEIOhZeRWKtr
-TFV4MiH6ZMCDWd0QGYDysHHjq0
```

[42] RFC 7519 – JSON Web Token estándar

9.3. Play Framework Security y roles

Play, además de la seguridad que implementa y va mejorando en cada, dispone de una característica de seguridad basada en las acciones definidas en los controladores. Se realiza marcando las acciones con la etiqueta `@Security.Authenticated`.

Esta etiqueta añade una comprobación a cualquier acción a la que le pongamos dicha anotación. Por defecto, esa comprobación llamará al método `getUsername(Http.Context ctx)` de la clase `Security.Authenticator` la cual intentará recuperar el nombre del usuario de la cookie de sesión. Si la sesión contiene el nombre de usuario, la acción continuará con su ejecución como lo normalmente lo haría. Si el nombre de usuario no está, directamente se devuelve una respuesta con un error 401, sin ejecutarse la acción.

Por suerte, se puede implementar un Authenticator propio que compruebe lo que se requiera. En este caso, se requiere que se compruebe que la petición contiene un JWT en la cabecera, y que además ese JWT pase la verificación, que sea un usuario del sistema y que es el rol adecuado. Para ello, solo hay que implementar una clase que herede de `Security.Authenticator` y sobreescribir los métodos `getUsername(Http.Context ctx)` y `onUnauthorized(Http.Context ctx)`.

En el primer método, se hará las comprobaciones del JWT y en el segundo se implementará la respuesta genérica de acceso no autorizado.

Como en el sistema desarrollado existen dos roles para usuarios (normales y administradores), se ha creado un Authenticator general llamado **Roles**, que será el encargado de comprobar las acciones a las que pueden acceder tanto usuarios corrientes como administradores, y dos Authenticators que heredan de Roles, llamados **User** para las acciones que solo pueden realizar los usuarios normales y **Administrator** para las acciones que solo pueden realizar los administradores.

En la clase padre Roles, se ha implementado una función que en vez de obtener el nombre de usuario de la cookie como hacía por defecto Play, lo que hace es intentar obtener el JWT de las cabeceras de la petición, puesto que el JWT debe ir en la cabecera `Authorization`. Las hijas, al heredar de esta, disponen de esta función como la de verificación del JWT.

Una vez implementado nuestros Authenticators, solo queda anotar las acciones con el rol que se necesite para acceder a ellas, anotándolas con Roles en caso de que puedan acceder ambos roles, con User en el caso de que puedan acceder tan solo usuarios corrientes o con Administrator en caso de que sean acciones que solo puedan realizar administradores. En el caso de que una acción no necesite autenticación ni rol, basta con no anotarla con ningún Authenticator.

```
@Security.Authenticated(Roles.class)
public Result tvShowById(Integer id) {…}

@Security.Authenticated(Administrator.class)
public Result createTvShow() {…}

@Security.Authenticated(User.class)
public Result voteTvShow(Integer id) {…}

// sin authenticator
public Result login() {…}
```

9.4. JPA

La API de persistencia de java añade una capa de seguridad al acceso de los datos, evitando de esta manera ciertos ataques como por ejemplo el SQL injection[43] ya que el sistema trata con objetos POJO y no con accesos y consultas directas a la base de datos.

Además de esto, JPA funciona con JPQL, un lenguaje de consulta orientado a objetos independiente de la plataforma donde se use, y aunque guarda cierta similitud con SQL, opera con objetos entidad de JPA en lugar de hacerlo directamente con las tablas de la base de datos.

Y, por si fuera poco, JPQL admite parámetros con nombre, de manera que se puede definir en el código fuera de la propia consulta, con lo que hace que el SQL injection sea más difícil todavía.

9.5. React Native

En React Native se hará referencia a la manera en que se trata la contraseña tanto en la identificación como en el registro y a la manera en la que la aplicación almacena el JWT.

Tanto en la identificación como en el registro, la contraseña cumple con los requerimientos de la API que se ha implementado, es decir, se aplica la función hash SHA512 gracias al paquete **bcrypt** y se envía la primera mitad en la petición a la API. Esta comunicación con la API debe ser cifrada mediante una petición a través de HTTPS/SSL.

En cuanto al almacenaje del JWT una vez que un usuario se ha identificado, se hace uso del componente nativo de React Native **AsyncStorage**[44], que será el encargado de almacenar datos necesarios como el JWT, el id el usuario y nombre del usuario.

AsyncStorage es asíncrono, de tipo clave-valor y es global en toda la aplicación. Es lo que se usa en vez de LocalStorage[45].

Hay que tener en cuenta que, en iOS, AsyncStorage se traduce por el código nativo de iOS que almacena pequeños valores en un diccionario serializado y grandes valores en fichero separados, lo cual garantiza totalmente su seguridad. En Android utilizará RocksDB[46] o SQLite[47] en función de lo que esté disponible.

Así que, cuando la aplicación se abra, comprobará si existe en AsyncStorage un JWT, y si existe lo comprobará con la API para ver si sigue vigente o es correcto. Si fuera correcto, la aplicación funcionaría sin problemas. En el caso de que el JWT no fuera correcto o de que no se encontrara ninguno en AsyncStorage, le redirigiría a la vista de Login.

[43] SQL injection – Método de infiltración de código intruso en las entradas para realizar operaciones sobre la base de datos
[44] AsyncStorage – Sistema de almacenamiento simple, asíncrono del tipo clave-valor de React Native
[45] LocalStorage – Sistema de almacenamiento en la web (navegador), en vez de en cookies
[46] RocksDB – Sistema de almacenamiento del tipo clave-valor persistente para entornos veloces
[47] SQLite – Motor de bases de datos SQL autónomo

10. Testing: pruebas

Para asegurar la mejor implementación y funcionamiento posible dentro de lo que quepa, es muy importante el testeo de las aplicaciones. En este caso, se ha llevado un testeo constante, de implementación de nuevos tests y mantenimiento de los acumulados de la API desarrollada.

10.1. JUnit

Como la API se ha desarrollado haciendo uso de Play Framework usando Java como lenguaje principal, la manera más fácil compatible con este framework es mediante JUnit. Con JUnit se ha podido implementar tests que han intentado abarcar la máxima cobertura posible de código, centrándose en los DAOs y Services.

Se ha implementado tests tanto unitarios como de integración, para probar también los servicios externos como TVDb o TMDb.

Para algunos tests unitarios, se ha tenido que utilizar mocks de ciertas dependencias externas para aislar completamente las unidades.

Siempre que se implementa un ticket nuevo, se implementan tests que prueban ese nuevo código, y antes de hacer cualquier commit, se pasa tanto esos nuevos tests como los anteriores para comprobar el buen funcionamiento e integración.

Estos tests son de nuevo ejecutados por el servicio de integración continua del que ya se ha hablado (Travis CI) y después de hacer cualquier merge, también se vuelven a pasar en local.

En el tramo final del proyecto, se cuenta con 139 tests implementados con JUnit.

10.2. Pruebas de usabilidad

Para los clientes móviles, la parte de administración y también para la API, aunque tenga tests, se ha realizado diversas pruebas de usabilidad mediante usuarios lo más reales posible con fin de detectar errores y fallos de usabilidad que no se pudieran detectar con los tests.

11. Repositorios y puesta en marcha

A continuación, podemos encontrar toda la información relativa a los repositorios del proyecto y todo lo necesario para la puesta en marcha de todo lo conseguido y explicado aquí.

11.1. Repositorios

Como ya se ha mencionado, los repositorios están albergados en GitHub

- API y Administración: https://github.com/DarkHollow/tfg-series-playAPI
- Cliente para iOS y Android: https://github.com/DarkHollow/tfg-react-native-Client

11.2. Preparación del entorno

Para la preparación del entorno, necesitaremos instalar las siguientes dependencias o tenerlas externamente preparadas

- Java 1.8: Oracle Java SE Downloads
- MySQL: MySQL Downloads
- Node.js: Node.js Downloads
    - Create React Native App:

```
$ npm install -g create-react-native-app
```

- Xcode: Xcode en Apple Developer
- Android Studio: Android Studio en Android Developer

Java y MySQL será necesario para poner en marcha la API en Playframework y Node.js para los clientes móviles en React Native. Es esencial que, una vez instalado Node.js, se instale mediante el mismo el paquete Create React Native App.

Tanto si tenemos MySQL instalado en local o se hará uso de una base de datos MySQL externa, hay que crearla y conocer los datos de acceso, nombre de base de datos, URL y puerto.

Por último, si se quiere construir la aplicación tanto para iOS como para Android, se necesita las herramientas necesarias para ello, las cuales son Xcode para iOS y Android Studio para Android. En el caso de Android Studio, además, se deberá comprobar que se instalen los siguientes componentes del mismo:

- Android SDK
- Android SDK Platform
- Performance (Intel ® HAXM)
- Android Virtual Device

Por defecto, las aplicaciones React Native para Android utilizan la versión Android 6.0 (Marshmallow), así que será necesario que desde Android SDK se instalen las siguientes dependencias:

- Android 6.0 (Marshmallow)
    - Google APIs
    - Android SDK Platform 23
    - Intel x86 Atom_64 System Image
    - Google APIs Intel x86 Atom_64 System Image
- SDK Tools
    - Android SDK Build-Tools 23.0.1

Para mayor comodidad, las herramientas de React Native necesitarán variables de entorno para encontrar las herramientas y SDK de Android:

```
export ANDROID_HOME=$HOME/Library/Android/sdk
export PATH=$PATH:$ANDROID_HOME/tools
export PATH=$PATH:$ANDROID_HOME/platform-toolsD
```

Si se fuera a probar la aplicación en dispositivos virtuales, para iOS, con Xcode es suficiente, automáticamente se instalará el iOS Simulator. Para Android, deberemos crear un Android Virtual Device desde el Android Virtual Device Manager usando la versión de Android anteriormente descrita e instalada.

11.3. Puesta en marcha en modo debug

Una vez descargados los repositorios y el entorno preparado solo queda ponerlos en marcha.

11.3.1. Puesta en marcha de la API

Para poner en marcha la API bastará con entrar al directorio raíz de la misma y ejecutar

```
$ sbt run
```

Play se encargará de descargar todas las dependencias necesarias y pondrá en marcha el proyecto en el puerto 9000.

Para acceder a la parte de administración accedemos a la siguiente dirección (si es en local) http://localhost:9000/admin/login

Para acceder a la documentación Swagger y probar cualquier `endpoint` accedemos a http://localhost:9000/api/docs

La ruta principal de los `endpoint` es
http://localhost:9000/api

11.3.2. Puesta en marcha de los clientes para iOS y Android

Para poner en marcha los clientes en modo debug con un simulador de iOS y/o Android o bien instalarlos en dispositivos físicos deberemos tener o los simuladores o los dispositivos en ejecución o conectados y ejecutar en el directorio raíz del proyecto:

Para instalarlo en iOS Simulator o en un dispositivo iOS conectado

```
$ npm run ios
```

Para instalarlo en una Android Virtual Device o en un dispositivo Android conectado

```
$ npm run Android
```

Al igual que con Play con sbt, Node.js se encargará de descargar todas las dependencias del proyecto y ponerlo en marcha.

Si todo ha ido como debería, tanto en iOS como en Android debería de estar ya la aplicación abierta, con la vista de identificación.

Esta es solo una manera de ejecutar la aplicación. También se puede hacer uso de Xcode o Android Studio para abrir directamente la aplicación y ejecutarla o modificarla. Otra manera es con Atom[48] junto a Nuclide[49] o con Expo Client[50].

[48] Atom – Editor de texto open source muy personalizable y con gestor de paquetes y una comunidad enorme detrás
[49] Nuclide – Colección de características para Atom para proporcionar funcionalidad IDE en distintos lenguajes de programación y tecnologías
[50] Expo Client – Toolchain open source que construye aplicaciones React Native y las abre directamente en el cliente para iOS o Android instalado en cualquier dispositivo

12. Documentación

Además de esta memoria y de los propios readme de cada uno de los proyectos, para el proyecto de la API y administración web implementado en Play se ha configurado una documentación generada con Swagger UI[51].

Con Swagger UI no solo se puede ver los distintos recursos y endpoints, sino que **se pueden probar** en la propia documentación.

En caso de tener el proyecto Play ejecutándose en localhost, se puede navegar por esta documentación accediendo a

http://localhost:9000/api/docs

Esta documentación muestra el listado total de endpoints del que dispone la aplicación Play, incluyendo los públicos, los de usuarios registrados y los de administrador.

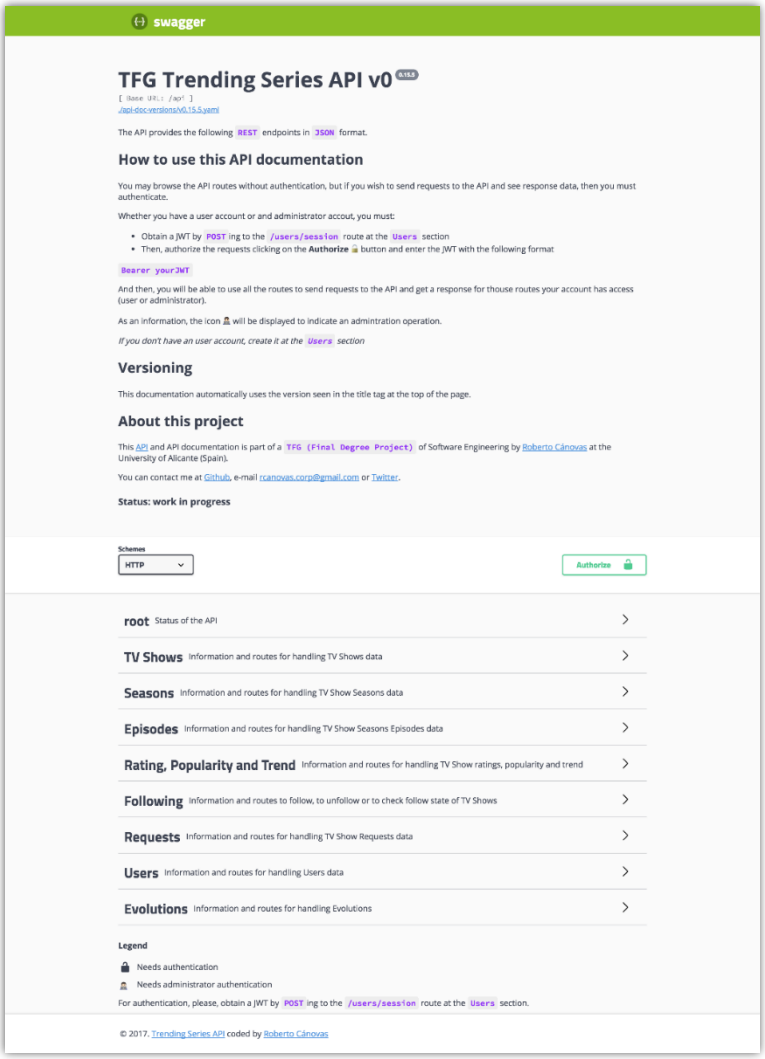


Ilustración 109 - Documentación Swagger

[51] Swagger UI, documentación y visualizador de los recursos de una API para una fácil y rápida implementación de clientes

Lo primero que uno se encuentra en esta documentación, es la versión de la misma, información sobre el proyecto y cómo utilizarla correctamente.

Indica que para los endpoints que requieren autenticación de usuario indicados con un candado, es necesario obtener el JWT de identificación y colocarlo pulsando en el botón **Authorize** junto con la palabra *Bearer* delante, ya que es lo que se haría colocando el JWT en la cabecera de las peticiones, Swagger UI lo hace por nosotros.

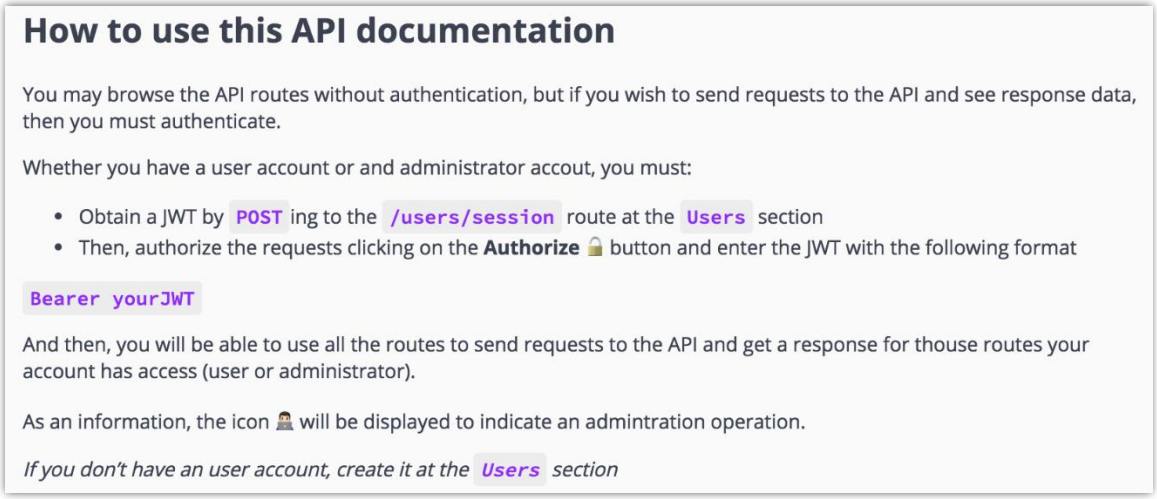


Ilustración 110 - Documentación Swagger: How to use it

Así que, yendo a la sección **Users** podemos encontrar tres endpoints públicos, evidentemente. Si uno no tiene cuenta, accede al primer endpoint y ahí puede encontrar toda la información sobre esta petición.

Se incluye absolutamente toda la información de la petición como los campos que acepta de entrada, ejemplo de entrada, lo que se requiere o no, todas las respuestas que puede devolver con ejemplos, códigos HTTP, el tipo de datos que devuelve, etc.

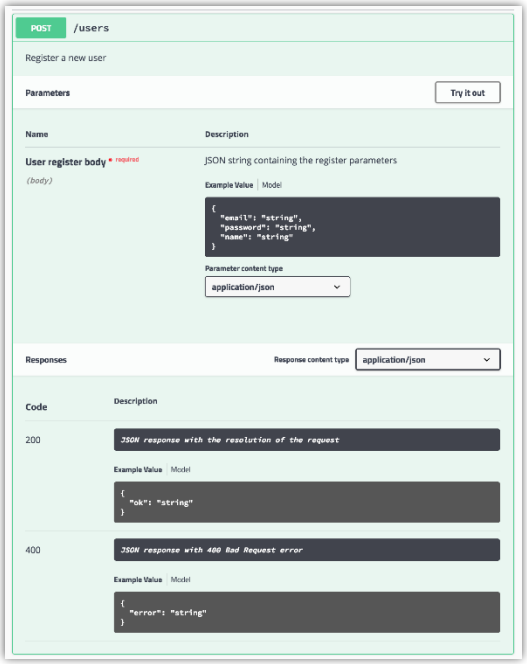


Ilustración 111 - Documentación Swagger: Información petición

Si se pulsa el botón **Try it out**, la vista cambia para poder probar este endpoint, de manera que se puede escribir los campos que acepta la petición, en este caso siendo email, password y name:

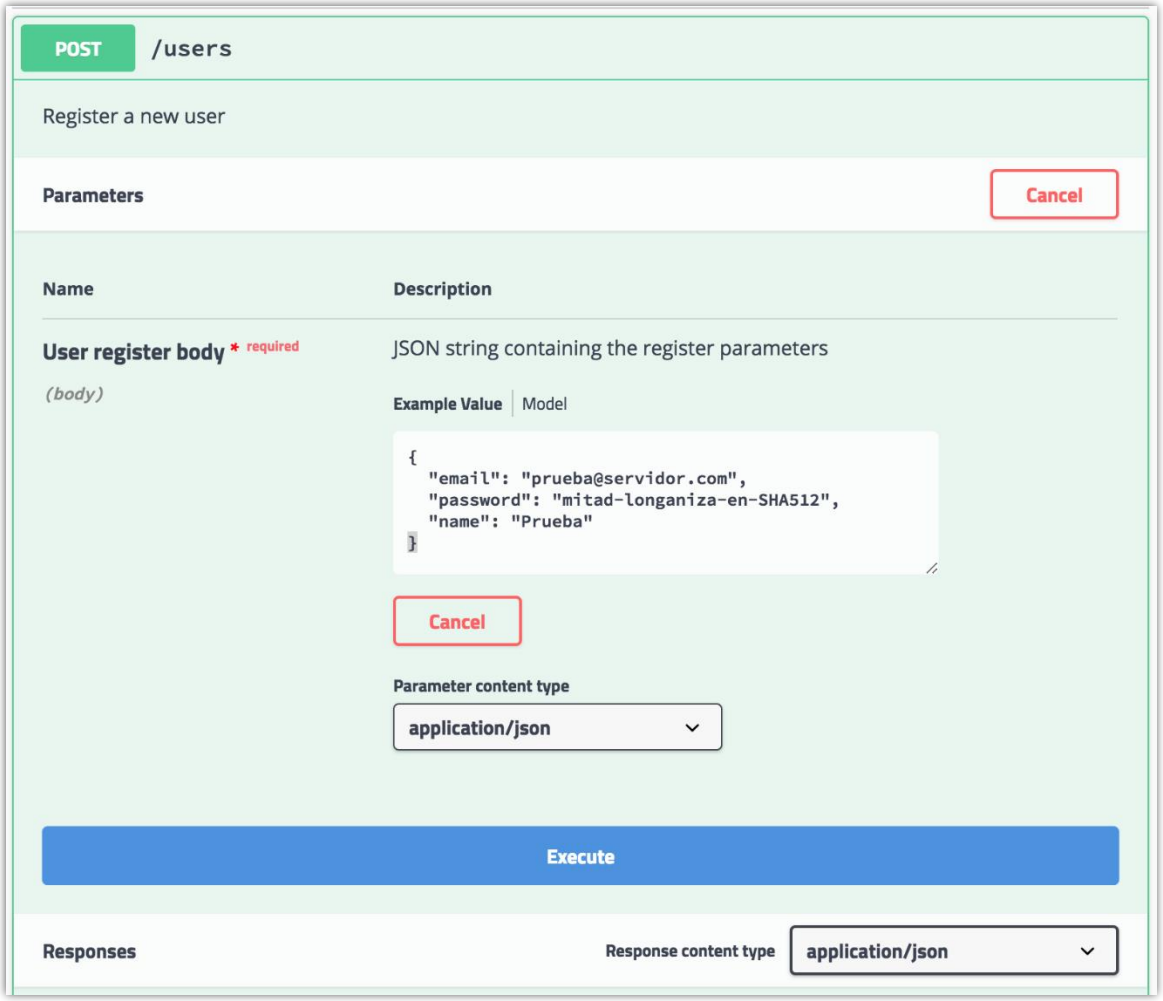


Ilustración 112 - Documentación Swagger: Try it out

Ahora ya, si se pulsa **Execute**, se envía la petición a la API que está en ejecución, y obtendremos una respuesta de esta e incluso la petición en formato *curl*:

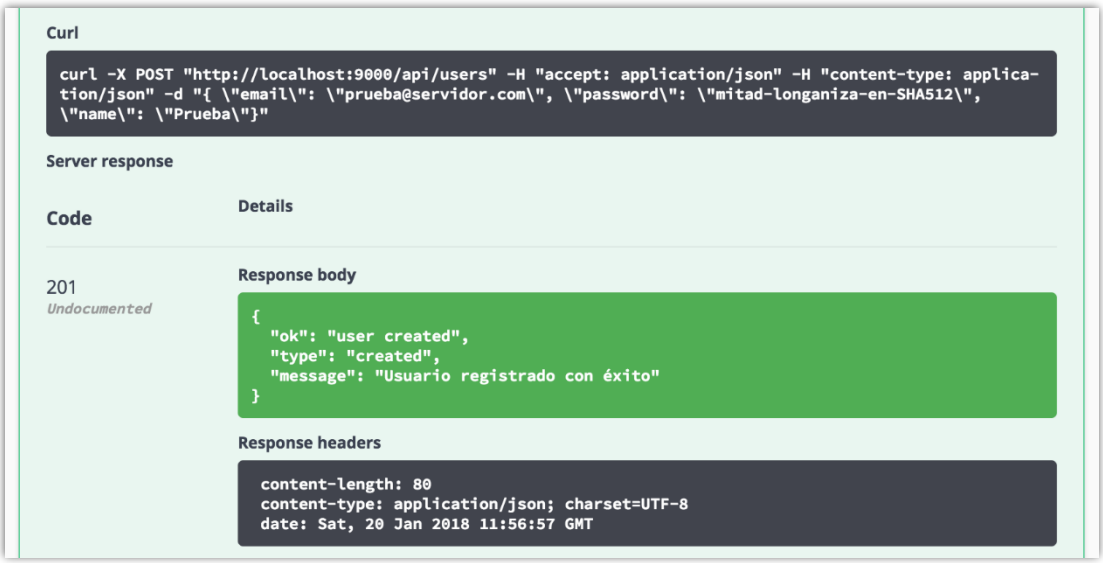


Ilustración 113 - Documentación Swagger: respuesta

Si ya se tiene una cuenta o se acaba de crear una con lo explicado anteriormente, se accede al endpoint *POST /users/session* para identificarse y obtener un JWT de acceso. De la misma manera que hemos hecho la petición de registro.

Una vez uno se identifica de la manera anterior, recibe en la respuesta el JWT. Ahora ese JWT se debe de incluir en la cabecera llamada **Authorization**, precediéndolo con la palabra *Bearer*.

Para que Swagger UI incluya esta cabecera con el JWT obtenido en el momento de la identificación automáticamente en todas las peticiones que se hagan desde esta documentación, se debe pulsar el botón **Authorize** y escribir *Bearer* 'JWT' en el campo.

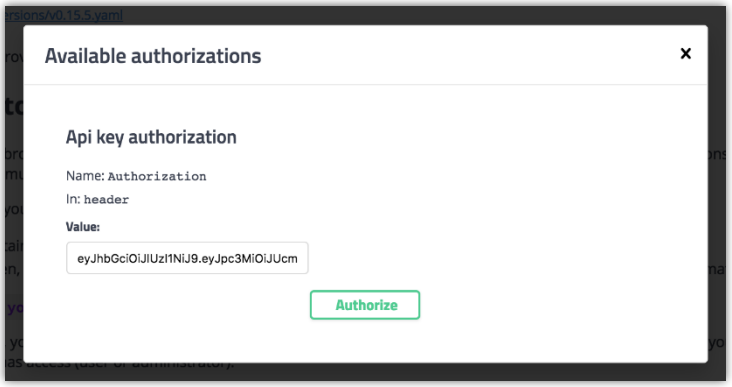


Ilustración 114 - Documentación Swagger: Authorize

Una vez hecho esto, en todas las peticiones que necesiten autenticación indicadas con un candado, incluirán esta cabecera con el JWT de identificación.

Ahora sí, se verá un ejemplo explicando con más detalles la interfaz. Si se elige **TV Shows**, se puede observar lo siguiente:

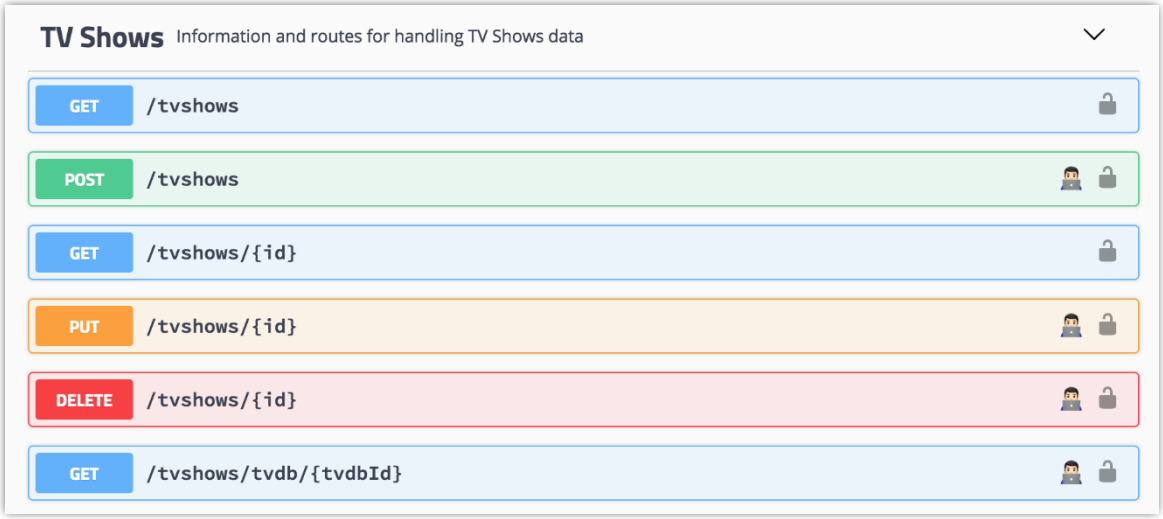


Ilustración 115 - Documentación Swagger: TV Shows

En el lado izquierdo se puede observar qué tipo de método HTTP de servicios REST es el de cada endpoint y a su lado, la URL relativa de cada acción.

En el lado derecho se observan dos iconos:

- Candado: indica que el endpoint requiere que el usuario esté identificado (JWT en la cabecera)
- Persona con portátil: indica que el endpoint requiere que el usuario sea administrador

Como ya se ha explicado en primer lugar el proceso de registro y de identificación mediante la documentación Swagger, omitiremos este proceso.

El primer endpoint de TV Shows es *GET /tvshows* el cual, como indica la documentación, devuelve información básica sobre las series y se puede filtrar por nombre. Si se incluye en el parámetro de búsqueda la palabra *Stranger* y se pulsa en **Execute**, se puede observar lo que devuelve la API:

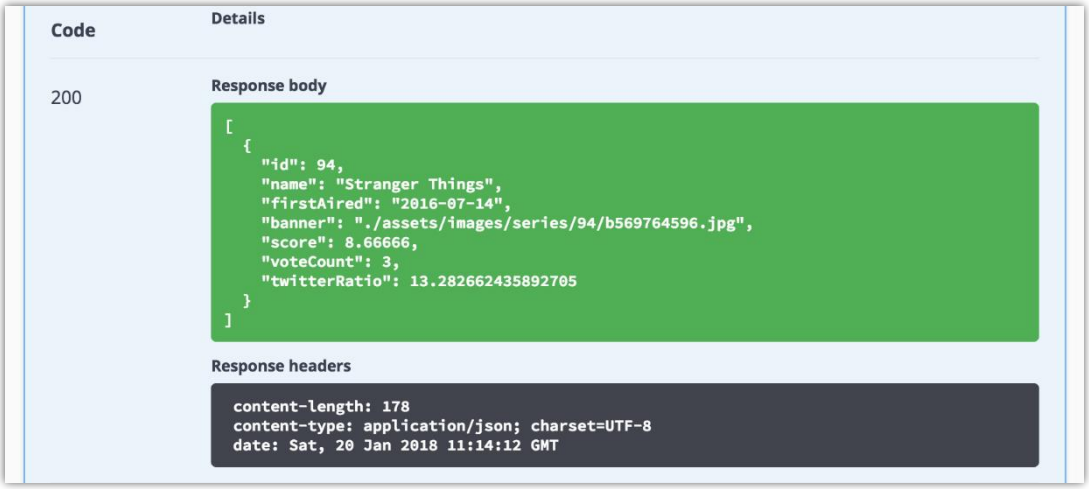


Ilustración 116 - Documentación Swagger: respuesta GET /tvshows con parámetro

Y así, con el resto de endpoints que se quieran probar de la documentación.

Esta documentación requiere que se definan todas las peticiones y respuestas que se puedan dar por parte de la API para que pueda ser generada, y ha sido generada según el proyecto iba creciendo, teniendo tantas versiones de la documentación como funcionalidades se han ido añadiendo.

Gracias a este tipo de documentación tan vistosa y probable, la implementación de un cliente se hace mucho más sencilla y llevadera, pues sabes en cualquier momento dónde se tiene que hacer la petición, con qué requerimientos, etc., y además saber qué esperar de cada petición, códigos de respuesta HTTP, etc.

Existen herramientas que, dada una documentación como esta, genera automáticamente las peticiones a cada uno de los endpoints gestionando cada respuesta automáticamente, para partir de una base sólida basada en la documentación dispuesta a ser modificada para llevar a cabo la creación del cliente.

Además de esta documentación a la que se puede acceder si está el proyecto de Play en marcha, se ha generado otra documentación Swagger a la que se ha denominado **documentación Swagger offline**, en la que se ha deshabilitado los botones de **Try it out** ya que se podrá acceder a ella sin tener el proyecto Play en ejecución y a través de GitHub Pages[52] que acceder directamente al repositorio al haber colocado esta documentación offline en el directorio */docs*:

https://darkhollow.github.io/tfg-series-playAPI

[52] GitHub Pages – Sitios web alojados directamente desde repositorios de GitHub

13. Conclusiones y mejoras

Una vez se ha finalizado con la implementación de este proyecto, se ha hecho una retrospección global y se concluye con que se ha cumplido con más objetivos de los que se pensaba en un primer momento. Si es cierto que hay ciertas características que se han quedado fuera, pero solo porque durante el desarrollo del proyecto han surgido nuevas características de mayor prioridad, lo cual es algo muy natural en este tipo de proyectos.

Por una parte, se ha desarrollado una API REST estable, segura y totalmente funcional la cual está debidamente documentada. A partir de esta API, se puede implementar cualquier cliente totalmente independiente que se quiera, y, además, gracias a la implementación de la documentación Swagger UI, este proceso les será muy fácil a cualquiera que se aventure a implementar un cliente.

Es muy interesante el hecho de que esta API es creciente, de manera que cuantos más usuarios la utilicen, mayor será la cantidad de recursos y datos disponibles.

La elección de Play Framework ha facilitado el trabajo inicial ya que se tenía conocimientos previos adquiridos en la asignatura de *Metodologías ágiles de diseño software* con este framework y con API REST en la asignatura de *Aplicaciones distribuidas en Internet*.

Por otra parte, se ha desarrollado una aplicación web de administración para esta API, en la que un administrador tendrá en su control los recursos de la misma. Siendo muy intuitiva y fácil de llevar, el trabajo del administrador es tan sencillo que ni siquiera tiene que introducir ni un solo dato, todo se realiza mediante acciones que se han automatizado de manera que, para obtener todos los datos, temporadas y episodios de una serie, tan solo hay que pulsar un botón y el sistema se encargará de todo.

Y, por último, se ha desarrollado un cliente doble smartphones para iOS y para Android en el que se hace uso de todas las funcionalidades de las que dispone la API implementada. Puede servir como un claro ejemplo de lo que se puede hacer con la API, aunque todavía se puede hacer mucho más con la cantidad de datos que se van generando.

En esta última parte, se ha apostado por React Native, tecnología relativamente nueva de la que no se tenía conocimientos previos, lo cual ha sido un reto y una grata satisfacción el haber aprendido a utilizarla y llevar a cabo la implementación de una aplicación completa.

Todo llevo implementado con la seguridad aplicada aprendida en asignaturas como *Seguridad en el Diseño Software*, de manera que tanto la API, como las contraseñas de los usuarios, la administración y el envío de información crítica desde el cliente al sistema y viceversa sea lo más seguro posible.

La gestión y flujo de trabajo del proyecto haciendo uso de Trello y GitHub sobre un sistema Kanban + Scrum + GitFlow + Travis CI ha sido de lo más satisfactorio del proyecto.

Intentando no dejar ninguna asignatura fuera, se ha hecho uso de conocimientos de testing adquiridos en Planificación y Pruebas de Sistemas Software, o la arquitectura clara de MVC que tanto se ha visto durante todo el transcurso del grado, bases de datos, análisis y especificación, etc.

Se podría seguir describiendo conocimientos aprendidos durante todos estos años aplicados en este proyecto, pero entonces, no acabaríamos nunca.

Por último, como cualquier proyecto, tiene margen de mejora y de futuras características, algunas descritas desde hace ya un tiempo y otras ideadas una vez el proyecto ha finalizado:

- Calificar episodios (calificación media de temporada)
- Gente (actores y actrices, directores, guionistas, etc.
- Servicios externos suplentes (en caso de que los principales fallen)
- Comentar episodios (mostrar comentarios hasta el último episodio marcado como visto
- Críticas
- Trailers de series y temporadas
- Noticias
- Notificaciones push de las series marcadas como seguidas
- Amigos y almas gemelas (gustos parecidos)

14. Referencias y bibliografía

Proyectos

- API y administración
    - Repositorio: https://github.com/DarkHollow/tfg-series-playAPI
    - Documentación: http://darkhollow.github.com/tfg-series-playAPI
- Cliente front-end
    - Repositorio: https://github.com/DarkHollow/tfg-react-native-Client

Tecnologías y herramientas utilizadas

- Trello: https://trello.com
- GitHub: https://github.com
- GitHub Pages: https://pages.github.com
- Travis CI: https://travis-ci.org
- Slack: https://slack.com
- Play Framework: https://www.playframework.com
- React Native: https://facebook.github.io/react-native
- Java 8: https://docs.oracle.com/javase/8/docs/api
- Javascript
    - ES7
    - JSX
    - Flow
    - Polyfills
- JPA: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
- MySQL: https://www.mysql.com
- Twitter4j: http://twitter4j.org/en/index.html
- Swagger UI: https://swagger.io/swagger-ui
- JWT: https://jwt.io
- Node.js: https://nodejs.org/es
- Scala: http://www.scala-lang.org
- sbt: https://www.scala-sbt.org
- Xcode: https://developer.apple.com/xcode
- iOS Simulator: https://goo.gl/CZ4YEq
- Android Studio, SDK, tools y AVD: https://developer.android.com/studio
- IntelliJ IDEA Ultimate by JetBrains: https://www.jetbrains.com/idea
- WebStorm by JetBrains: https://www.jetbrains.com/webstorm
- Atom: https://ide.atom.io
- TheTVDB: https://www.thetvdb.com
- TheMovieDb: https://www.themoviedb.org
- Adobe Photoshop CC 2017: https://www.adobe.com/es/products/photoshop.html
- Wacom Intuos Art: https://www.wacom.com/es-es/products/intuos-art

Material de apoyo

- Scrum Manager – Guía de formación: http://scrummanager.net/files/scrum_manager.pdf
- Kanban – Artículo de Atlassian: https://es.atlassian.com/agile/kanban
- GitFlow – A successful Git branching model por Vicent Driessen
- React Native Docs – https://facebook.github.io/react-native/docs/getting-started.html
- Guía de seguridad – Doing it Right por Taylor Hornby
- EC2017 https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

- Artículos de ayuda en https://medium.com sobre React Native
- Apuntes de Metodologías Ágiles de Desarrollo Software de la UA
- Apuntes de Seguridad en el Diseño Software de la UA
- Apuntes de Planificación y Pruebas de Sistemas Software de la UA
- Apuntes de Aplicaciones Distribuidas en Internet de la UA